

Online Data Stream Classification in the Presence of Concept Drift and Class Imbalance

By

Chun Wai Chiu

A thesis submitted to
the University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
March 2022

UNIVERSITY OF
BIRMINGHAM

University of Birmingham Research Archive

e-theses repository

This unpublished thesis/dissertation is copyright of the author and/or third parties. The intellectual property rights of the author or third parties in respect of this work are as defined by The Copyright Designs and Patents Act 1988 or as modified by any successor legislation.

Any use made of information contained in this thesis/dissertation must be in accordance with that legislation and must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the permission of the copyright holder.

ABSTRACT

Online data stream learning requires to process each training example upon arrival. Machine learning algorithms designed to process data in this manner are particularly suitable for this digital era, as the volume and the incoming speed of data are growing everyday. However, the environments of most real-world applications in online data stream learning are usually non-stationary, meaning that their underlying distribution may change over time (concept drift). Also, in real-world classification task, the number of examples from different classes are unlikely to be equal (class imbalance). Existing literature of concept drift adaptation and class imbalanced learning have been well-established in the last years. Yet, there still exists little work to address the joint challenge of them. This thesis contributes to this area of study by investigating how to deal with concept drift from a memory perspective and investigating how to create synthetic examples to aid in learning the skewed class while able to adapt to concept drift in class imbalanced data streams. The main contributions of this thesis are:

- A study of the impact of different memory management strategies to the predictive performance.
- A novel diversity-based memory management strategy that can maximise the chance of having relevant past knowledge to exploit in dealing with concept drift.
- A novel framework called Concept Drift Handling Based on Clustering in the Model Space (CDCMS), which can deal with multiple types of concept drifts in class balanced

data streams by exploiting relevant past knowledge.

- A set of novel memory strategies for class imbalanced learning that can maintain and recover relevant past knowledge to deal with concept drift in class imbalanced data stream. From that, a novel framework called Concept Drift Handling Based on Clustering in the Model Space for Class Imbalanced Learning (CDCMS.CIL) is proposed.

- A novel approach called Synthetic Minority Oversampling based on Stream Clustering (SMOClust), which creates synthetic examples based on stream clustering to aid in learning the skewed class, while able to adapt to concept drifts.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my supervisor Dr. Leandro L. Minku. I would like to thank him for his guidance, inspiration, and support. He can always give me good insights and guide me through the difficulties. Without his valuable advice and help, this thesis would never be completed. To me, he is not just my superior but also my mentor. I look up to him not only for his professionalism in research but also his persistence in teaching. He can always find a way to explain difficult concepts to be very easy to understand. His attitude in both researching and teaching have a great influence to me.

I would also like to thanks the thesis group committees, Prof. Ata Kabon and Prof. Hamid Dehghani for their valuable comments, advice and discussion. I also thank them for their encouragement and understanding when my research reach a bottleneck.

Thanks Dr. Shuo Wang for sharing the source code of the Prequential Multi-class AUC method and real-world data sets as well as the valuable discussions, and to Dariusz Brzezinski for the guidance on how to use the artificial class imbalanced data stream generator and the method to estimate the characteristics of real-world data streams. Also, I would like to thank Dr. Shuo Wang once again and Prof. Mohammed Gaber for agreeing to be the examiners of my Ph.D viva.

Apart from thanking people who supported my research technically, I would also like to thank the administrative team of the School of Computer Science. I particularly thank to Sarah Brookes for her support in administrative work throughout my Ph.D study. I

also thank Dr. Peter Hancox for the research skills he taught and Dr. Mark Lee for his understanding and for his role as Head of School.

I would also like to express my sincere gratitude to my family and my girlfriend. Thanks for always being here encourage me and to understand my difficulties. Thanks very much to Elaine Yuen Ling Yu, Wallace Wing Hung Chiu, Emily Tsz Ying Chiu and Carol Wai Shan Siu for all your love and support for many years.

Lastly, I would like to thank all the friends I met in the UK and I knew Hong Kong. Thanks Alvin Ma for always sharing faith, ups and downs in life, taught me badminton skills and took me travel around the UK. Those are precious moment that I will never forget. Thanks Jonathan Lee and Elpis Lam for always be around, sharing faith, life and hobbies. Thanks Timothy Chong, Sammy Ng, Jonathan Chan, and Wilson Lau for good company since undergraduate study. I also thanks people from my spiritual home in the UK - Leicester Chinese Christian church. Special thanks to Raymond Tang, Michelle Tang and Pastor Esther for taking care of me during the time I studied in the UK. Also, thanks Dr. Yiwen Tan for sharing your experience in Ph.D study. Many thanks also to all my friends who shared joy with me and helped me get through any difficulties in life. Most importantly, I thank my Heavenly Father for putting these great people in my life. You made me who I am today.

Contents

Page

1 Introduction **1**

1.1 Definitions and Problem Formulation 7

1.1.1 Offline, Incremental and Online Learning 7

1.1.2 Class Imbalance 11

1.1.3 Concept Drift 15

1.2 Research Questions 20

1.2.1 Diversity-based Memory Strategies for Concept Drift Adaptation in Class Balanced Data Streams 21

1.2.2 Diversity-based Framework for Concept Drift Adaptation in Class Imbalanced Data Streams 24

1.2.3 Synthetic Minority Oversampling for Learning Drifting Class Imbalanced Data Streams 25

1.3 Thesis Contribution and Organisation 26

2 Background **29**

2.1 Hoeffding Tree 30

2.2 Diversity 32

2.3 Synthetic Minority Oversampling Technique (SMOTE) 34

2.4 Data Stream Clustering 36

3 Related Works in Dealing with Concept Drift and Class Imbalance in Data Stream Classification **41**

3.1 Approaches for Learning Drifting Class Balanced Data Streams 42

3.1.1 Explicit Concept Drift Adaptation Approaches for Class Balanced Data Stream Learning 44

3.1.2 Implicit Concept Drift Adaptation Approaches for Class Balanced Data Stream Learning 54

3.2 Approaches for Learning Stable Concept Class Imbalanced Data Streams . . 57

3.2.1 Gaussian Hellinger Very Fast Decision Tree (GH-VFDT) and Hellinger Distance Very Fast Decision Tree (HD-VFDT) 60

3.2.2 OnlineUnderOverBagging and OnlineSMOTEBagging 61

3.2.3 Online Sequential Extreme Learning Machine (OS-ELM) with Improved SMOTE 62

3.3 Approaches for Learning Drifting Class Imbalanced Data Streams 64

3.3.1 Explicit Concept Drift Adaptation Approaches 64

3.3.2 Implicit Concept Drift Adaptation Approaches 71

3.4 Evaluation Methods for Data Stream Learning 76

3.5 Chapter Summary 83

4 Diversity-based Memory Strategies for Concept Drift Adaptation in Class Balanced Data Streams **94**

4.1 Diversity-based Memory Management 97

4.1.1 Description of Diversity Pool (DP) 98

4.1.2 Design of the Experiment 105

4.1.3 Analysis 111

4.1.4 Summary of Diversity-based Memory Management Study 122

4.2 Diversity-based Memory Recovery 123

4.2.1 Description of Concept Drift Handling Based on Clustering in the Model Space (CDCMS) 124
4.2.2 Design of the Experiment 136
4.2.3 Analysis 141
4.2.4 Experiments to Analyse the Time-Memory-Accuracy Relationship in CDCMS 157
4.2.5 Summary of Diversity-based Memory Recovery Study 160
4.3 Summary and Discussion 161

5 Diversity-based Framework for Concept Drift Adaptation in Class Imbalanced Data Streams 164

5.1 Description of Concept Drift Handling Based on Clustering in the Model Space for Class Imbalanced Learning (CDCMS.CIL) 168
5.1.1 Clustering in the Model Space for Class Imbalanced Learning 178
5.1.2 Diversity-based Memory Management Strategy for Class Imbalanced Learning 186
5.2 Design of the Experiment 189
5.2.1 Data streams 190
5.2.2 Experiment Setup 194
5.3 Analysis 199
5.3.1 Comparison between CDCMS.CIL and CDCMS 199
5.3.2 Comparing Predictive Performance of Different Versions of CDCMS.CIL 209
5.3.3 Comparing CDCMS.CIL against Existing Approaches 223
5.4 Summary and Discussion 238

6 Synthetic Minority Oversampling for Learning Drifting Class Imbalanced Data Streams 240

6.1 Description of Synthetic Minority Oversampling based on Stream Clustering (SMOClust) . . . 244
6.1.1 Generating a synthetic minority class example for oversampling using the micro-clusters . . . 249
6.1.2 Sampling from a Micro-cluster with the Highest Probability at a Designated Location . . . 254
6.1.3 Uniform Sampling inside a Hyper-Sphere . . . 259
6.2 Design of Experiment . . . 261
6.2.1 Data streams . . . 262
6.2.2 Experiment Setup . . . 270
6.3 Analysis . . . 278
6.3.1 Results with Artificial Data Streams . . . 279
6.3.2 Results with Real-world Data Streams . . . 308
6.4 Summary and Discussion . . . 311

7 Conclusion and Future Work 314

7.1 Diversity-based Memory Strategies for Concept Drift Adaptation in Class Balanced Data Streams . . . 317
7.1.1 Contributions . . . 317
7.1.2 Future Work . . . 319
7.2 Diversity-based Framework for Concept Drift Adaptation in Class Imbalanced Data Streams . . . 321
7.2.1 Contributions . . . 321
7.2.2 Future Work . . . 322
7.3 Synthetic Minority Oversampling for Learning Drifting Class Imbalanced Data Streams . . . 324
7.3.1 Contributions . . . 324

7.3.2 Future Work 325

A Percentage Differences of Sine, Agrawal, SEA and STAGGER Concepts 327

B All Performance Plots of DP 329

C All Performance Plots of CDCMS 334

D Plots of CDCMS's Time, Memory and Accuracy Relationship on Sine1-abrupt, Sine1-gradual, Sine2-abrupt and Sine2-gradual Streams 339

E All Performance Plots of the Comparison between CDCMS.CIL$^{\text{Preq. Acc.}}$ and CDCMS 343

F All Performance Plots of the Comparison between Different Versions of CDCMS.CIL 354

G All Performance Plots of the Comparison between CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ and Existing Approaches with and without Drift Detector 365

References 381

List of Figures

1.1 Illustration of Minority Class Example Types 14

1.2 Illustration of Concept Drift Types 17

3.1 Example of ROC Curve and the Area Under ROC Curve (AUC) 82

4.1 Prequential Accuracy Using Different Memory Strategies over Artificial Data Streams (Agrawal3-4,6) 112

4.2 Prequential Accuracy of DP and Existing Approaches over Artificial Data Streams (Agrawal1,5 and SEA1) 121

4.3 Prequential Accuracy in Sine2-gradual, Agr3-gradual, SEA1-gradual and SEA2-gradual streams 145

4.4 Prequential Accuracy in Sine1-gradual stream 148

4.5 Prequential Accuracy in Sine1-abrupt and Sine2-abrupt Streams 152

4.6 Prequential Accuracy with Real-World Data Streams 155

4.7 Overall runtime, memory and accuracy of CDCMS for different maximum total numbers of models in Agrawal3-gradual. The different coloured lines represent different sliding window sizes. 159

5.1 Class diagram of Class LearnerWithInfo (The First Method is the Constructor) 172

5.2 Class diagram of Class EnsembleWithInfo (The First Method is the Constructor) 174

5.3 Illustration of Comparing Similarities between Base Learners based on the Most Recent Sliding Window of Examples 178

5.4 Illustration of Comparing Similarities between Base Learners based on Micro-Clusters . . . 180
5.5 Illustration of the Clustering in the Model Space Strategy for Class Imbalanced Learning . . . 183
5.6 Illustration of the Diversity-based Memory Management Strategy for Class Imbalanced Learning . . . 187
5.7 CDCMS.CIL vs CDCMS: STAGGER2 (Gradual, 9-1) . . . 203
5.8 CDCMS.CIL vs CDCMS: Sine2 (Gradual, 9-1) . . . 206
5.9 $\text{CDCMS.CIL}_{\text{OS}}^{\text{Preq. Acc.}}$vs $\text{CDCMS.CIL}_{\text{US}}^{\text{Preq. Acc.}}$vs $\text{CDCMS.CIL}^{\text{Preq. Acc.}}$: Sine1 streams (Time Decay G-Mean) . . . 215
5.10 $\text{CDCMS.CIL}_{\text{OS}}^{\text{Preq. Acc.}}$vs $\text{CDCMS.CIL}_{\text{US}}^{\text{Preq. Acc.}}$vs $\text{CDCMS.CIL}^{\text{Preq. Acc.}}$: Sine2 streams (Time Decay G-Mean) . . . 216
5.11 $\text{CDCMS.CIL}_{\text{OS}}^{\text{Preq. Acc.}}$vs $\text{CDCMS.CIL}_{\text{US}}^{\text{Preq. Acc.}}$vs $\text{CDCMS.CIL}^{\text{Preq. Acc.}}$: Sine2 gradual (0.9:0.1) stream . . . 218
5.12 $\text{CDCMS.CIL}_{\text{OS}}^{\text{Preq. Acc.}}$, $\text{CDCMS.CIL}_{\text{US}}^{\text{Preq. Acc.}}$and $\text{CDCMS.CIL}^{\text{Preq. Acc.}}$: Sine1 gradual (0.8:0.2) stream . . . 221
5.13 Comparing CDCMS.CIL against Existing Approaches: Sine1 (Time decay G-Mean) . . . 229
5.14 Comparing CDCMS.CIL against Existing Approaches: Sine2 (Time decay G-Mean) . . . 230
5.15 Comparing CDCMS.CIL against Existing Approaches: Agrawal1 (Time decay G-Mean) . . . 232
5.16 Comparing CDCMS.CIL against Existing Approaches: Airlines Stream . . . 234
5.17 Predictive Performance on NOAA Stream . . . 237

6.1 Illustration of Combining the $mc_{anchor}^{minority}$ with its 3-nearest neighbours into one micro-cluster . . . 252

6.2 Illustration of Sampling a Synthetic Minority Class Example from mc_{temp} . . 253
6.3 Illustration of Sampling from mc_{temp} 256
6.4 Periodic Class Balanced Holdout Test G-Mean Against Time Steps in Two Dimensional StaticIm1_Move7 293
6.5 Decision Areas Against Class Balanced Test Set at 70k Time Steps (Before Drift) of Two Dimensional StaticIm1_Move7 294
6.6 Decision Areas Against Class Balanced Test Set at 100k Time Steps (After Drift) of Two Dimensional StaticIm1_Move7 298
6.7 Decision Areas Against Class Balanced Test Set at 200k Time Steps (End of Stream) of Two Dimensional StaticIm1_Move7 299
6.8 Periodic Class Balanced Holdout Test G-Mean Against Time Steps in Two Dimensional StaticIm10_Rare100 302
6.9 Decision Areas Against Class Balanced Test Set at 70k Time Steps (Before Drift) of Two Dimensional StaticIm10_Rare100 303
6.10 Decision Areas Against Class Balanced Test Set at 100k Time Steps (After Drift) of Two Dimensional StaticIm10_Rare100 306
6.11 Decision Areas Against Class Balanced Test Set at 200k Time Steps (End of Stream) of Two Dimensional StaticIm10_Rare100 307

B.1 Prequential Accuracy Using Different Memory Strategies over Artificial Data Streams (Agrawal1-6) 330
B.2 Prequential Accuracy Using Different Memory Strategies over Artificial Data Streams (Agrawal7-9, SEA1-3) 331
B.3 Prequential Accuracy of DP and Existing Approaches over Artificial Data Streams (Agrawal1-6) 332
B.4 Prequential Accuracy of DP and Existing Approaches over Artificial Data Streams (Agrawal7-9, SEA1-3) 333

C.1 Prequential Accuracy of CDCMS and Existing Approaches over Artificial Data Streams with Abrupt Drifts (Sine1-2, Agrawal1-3) 335
C.2 Prequential Accuracy of CDCMS and Existing Approaches over Artificial Data Streams with Abrupt Drifts (Agrawal4, SEA1-2, STAGGER1-2) 336
C.3 Prequential Accuracy of CDCMS and Existing Approaches over Artificial Data Streams with Gradual Drifts (Sine1-2, Agrawal1-3) 337
C.4 Prequential Accuracy of CDCMS and Existing Approaches over Artificial Data Streams with Gradual Drifts (Agrawal4, SEA1-2, STAGGER1-2) 338

D.1 Overall runtime, memory and accuracy of CDCMS for different maximum total numbers of models in Sine1-abrupt. The different coloured lines represent different sliding window sizes. 340
D.2 Overall runtime, memory and accuracy of CDCMS for different maximum total numbers of models in Sine1-gradual. The different coloured lines represent different sliding window sizes. 340
D.3 Overall runtime, memory and accuracy of CDCMS for different maximum total numbers of models in Sine2-abrupt. The different coloured lines represent different sliding window sizes. 341
D.4 Overall runtime, memory and accuracy of CDCMS for different maximum total numbers of models in Sine2-gradual. The different coloured lines represent different sliding window sizes. 341
D.5 Overall runtime, memory and accuracy of CDCMS for different maximum total numbers of models in Agrawal3-gradual. The different coloured lines represent different sliding window sizes. 342

E.1 CDCMS.CIL vs CDCMS: Sine1 (Time decay G-Mean) 344
E.2 CDCMS.CIL vs CDCMS: Sine2 (Time decay G-Mean) 345
E.3 CDCMS.CIL vs CDCMS: Agrawal1 (Time decay G-Mean) 346

E.4 CDCMS.CIL vs CDCMS: Agrawal2 (Time decay G-Mean) 347
E.5 CDCMS.CIL vs CDCMS: Agrawal3 (Time decay G-Mean) 348
E.6 CDCMS.CIL vs CDCMS: Agrawal4 (Time decay G-Mean) 349
E.7 CDCMS.CIL vs CDCMS: SEA1 (Time decay G-Mean) 350
E.8 CDCMS.CIL vs CDCMS: SEA2 (Time decay G-Mean) 351
E.9 CDCMS.CIL vs CDCMS: STAGGER1 (Time decay G-Mean) 352
E.10 CDCMS.CIL vs CDCMS: STAGGER2 (Time decay G-Mean) 353

F.1 Comparing Different Versions of CDCMS.CIL: Sine1 (Time decay G-Mean) . 355
F.2 Comparing Different Versions of CDCMS.CIL: Sine2 (Time decay G-Mean) . 356
F.3 Comparing Different Versions of CDCMS.CIL: Agrawal1 (Time decay G-Mean)357
F.4 Comparing Different Versions of CDCMS.CIL: Agrawal2 (Time decay G-Mean)358
F.5 Comparing Different Versions of CDCMS.CIL: Agrawal3 (Time decay G-Mean)359
F.6 Comparing Different Versions of CDCMS.CIL: Agrawal4 (Time decay G-Mean)360
F.7 Comparing Different Versions of CDCMS.CIL: SEA1 (Time decay G-Mean) . 361
F.8 Comparing Different Versions of CDCMS.CIL: SEA2 (Time decay G-Mean) . 362
F.9 Comparing Different Versions of CDCMS.CIL: STAGGER1 (Time decay G-Mean) 363
F.10 Comparing Different Versions of CDCMS.CIL: STAGGER2 (Time decay G-Mean) 364

G.1 Comparing CDCMS.CIL against Existing Approaches: Sine1 (Time decay G-Mean) 366
G.2 Comparing CDCMS.CIL against Existing Approaches: Sine2 (Time decay G-Mean) 367
G.3 Comparing CDCMS.CIL against Existing Approaches: Agrawal1 (Time decay G-Mean) 368

G.4 Comparing CDCMS.CIL against Existing Approaches: Agrawal2 (Time decay G-Mean) . . . 369
G.5 Comparing CDCMS.CIL against Existing Approaches: Agrawal3 (Time decay G-Mean) . . . 370
G.6 Comparing CDCMS.CIL against Existing Approaches: Agrawal4 (Time decay G-Mean) . . . 371
G.7 Comparing CDCMS.CIL against Existing Approaches: SEA1 (Time decay G-Mean) . . . 372
G.8 Comparing CDCMS.CIL against Existing Approaches: SEA2 (Time decay G-Mean) . . . 373
G.9 Comparing CDCMS.CIL against Existing Approaches: STAGGER1 (Time decay G-Mean) . . . 374
G.10 Comparing CDCMS.CIL against Existing Approaches: STAGGER2 (Time decay G-Mean) . . . 375
G.11 Comparing CDCMS.CIL against Existing Approaches: Airlines, NOAA, and INSECTS Streams (Time decay G-Mean) . . . 376
G.12 Comparing CDCMS.CIL against Existing Approaches: Covtype Streams (Time decay G-Mean) . . . 377

List of Tables

3.1 Confusion Matrix for Binary Classification Problem 80

3.2 Summary of Strengths and Weaknesses of Reviewed Approaches 86

4.1 Summary of Data Streams 106

4.2 Options for Hyper-parameters 109

4.3 Chosen Hyper-parameters 110

4.4 Memory Strategy Comparison – Mean, Standard Deviations, Friedman Test and Nemenyi Test Results 113

4.5 State of Diversity Pool P in Agrawal4 stream 115

4.6 State of Diversity Pool P in Agrawal6 stream 117

4.7 Comparison against Existing Approaches – Mean, Standard Deviations, Friedman Test and Nemenyi Test Results 119

4.8 Summary of Artificial Data Streams 138

4.9 Options for Hyper-parameters 141

4.10 Chosen Hyper-parameters (Artificial Data Streams) 142

4.11 Chosen Hyper-parameters (Real-world Data Streams) 143

4.12 Average Prequential Accuracy (Standard Deviation) Across 30 Runs For Artificial Data Stream with Gradual Drifts 144

4.13 Average Prequential Accuracy (Standard Deviation) Across 30 Runs For Artificial Data Stream with Abrupt Drifts 151

4.14 Average Prequential Accuracy (Standard Deviation) Across 30 Runs For Real World Data Streams 154

5.1 Summary of Artificial Data Streams 191
5.2 Summary of Real-World Data Streams 192
5.4 Options for Hyper-parameters 196
5.5 Chosen Hyper-parameters of Existing Approaches 197
5.6 Chosen Hyper-parameters of CDCMS.CIL 198

6.1 Summary of Artificial Data Streams 263
6.2 Summary of Single Factor Drift Artificial Data Streams with Severe Imbalance Ratio 264
6.3 Summary of Real-World Data Streams 265
6.4 Characteristics of Real-World Data Streams (Values in the brackets are the median) 269
6.5 Options for Hyper-parameters 273
6.6 Chosen Hyper-parameters of Approaches without Drift Detector 274
6.7 Chosen Hyper-parameters of Approaches with Drift Detector 275
6.10 30 Runs Average G-Mean on Artificial Data Streams (A12 SMOClust vs Others) 281
6.13 30 Runs Average Prequential G-Mean on Severely Class Imbalanced Artificial Data Streams (A12 SMOClust vs Others) 286
6.15 Chosen Hyper-parameters of Approaches for 2D Data Streams 291
6.16 30 Runs Average G-Mean on Two Dimensional Version of Representative Artificial Data Streams where SMOClust Performed Better (A12 SMOClust vs Others) 292
6.17 30 Runs Average G-Mean on Two Dimensional Version of Representative Artificial Data Streams where SMOClust Performed Worse (A12 SMOClust vs Others) 301

A.1 Percentage Difference of Concepts 328

Acronyms

ADASYN Adaptive Synthetic Sampling. 58

ADWIN Adaptive Windowing. 6, 45, 49, 50, 69, 70, 91

AUC Area Under the Receiver Operating Characteristic Curve. 67

AUE Accuracy Update Ensemble. 56

BEV Bagging Ensemble Variation. 59

C-SMOTE Continuous Synthetic Minority Oversampling Technique. 65, 69, 70, 91

CDCMS Concept Drift Handling Based on Clustering in the Model Space. 23, 24, 96, 97, 124, 126–136, 138–143, 145–150, 152–158, 160, 161, 166–168, 171, 175–178, 186, 189, 190, 195–197, 199–209, 224, 225, 227, 228, 231, 233, 234, 238, 239, 315–321

CDCMS.CIL Concept Drift Handling Based on Clustering in the Model Space for Class Imbalanced Learning. 24, 25, 27, 165, 167, 168, 171–173, 175–177, 181, 184, 185, 188, 189, 196, 199–220, 222–228, 231, 233–236, 238, 239, 315, 316, 321–324, 326

CF Clustering Feature vector. 37–40

CVFDT Concept-adaptive Very-Fast Decision Tree. 43

DDD Diversity for Dealing with Drifts. 23, 45, 50–52, 86, 94–96, 130, 131, 139, 143, 148–150, 152, 153

DDM Drift Detection Method. 45, 46, 48, 49, 65, 100, 108, 120–122, 126

DDM-GMean Drift Detection Method for Online Class Imbalance using G-Mean. 196

DDM-OCI Drift Detection Method for Online Class Imbalance. 65, 66, 196

DetectA Detect Abrupt Drift. 126

DoF Degree of Drift. 49

DP Diversity Pool. 23, 95–105, 108, 111, 112, 118, 120, 121, 132, 136, 139, 143, 146, 148–150, 161

DWM Dynamic Weighted Majority. 54–57, 71, 74, 87, 93, 95, 109, 121, 122

DWMIL Dynamic Weighted Majority for Imbalanced Learning. 56, 72, 74, 75, 93

ECDD Exponentially Weighted Moving Average Charts for Concept Drift Detection. 48, 49

EDDM Early Drift Detection Method. 45, 48–50, 100, 121, 126

ELM Extreme Learning Machine. 71

ESOS-ELM Ensemble of Subset Online Sequential Extreme Learning Machine. 65, 70, 71, 91

EWMA Exponentially Weighted Moving Average. 49

FHDDM Fast Hoeffding Drift Detection Method. 126

FIFO First in First Out. 53, 86, 95, 108, 114, 117, 120, 161

G-Mean Geometric Mean. 71, 74, 77, 81, 82, 91, 93, 177, 195, 197, 200, 202, 207, 210–214, 216, 217, 219, 220, 223, 224, 226–228, 231, 233, 235, 236, 272, 277, 279, 283, 284, 288, 289, 308, 309, 343, 354, 365

GH-VFDT Gaussian Hellinger Very Fast Decision Tree. 25, 57, 59–61, 88, 189, 194, 225, 227, 228, 235

HD-VFDT Hellinger Distance Very Fast Decision Tree. 25, 57–61, 88, 190, 194, 225, 227, 228, 235

HDDM Hoeffding's inequality Drift Detection Method. 48

HDDT Hellinger Distance Decision Tree. 60

HLFR Hierarchical Linear Four Rates. 66

HT Hoeffding Tree. 29–31

HTNB Hoeffding Tree with Naïve Bayes at Leaves. 23, 31, 93, 108, 110, 118, 120, 139

HUWRS Heuristic Updatable Weighted Random Subspaces. 75

HUWRS.IP Heuristic Updatable Weighted Random Subspaces with Instance Propagtion. 72, 75, 76, 93

Learn++.CDS Learn++ for Concept Drift with SMOTE. 72–74, 92

Learn++.NIE Learn++ for Non-stationary and Imbalanced Environment. 72–74, 92

Learn++.NSE Learn++ for Non-stationary Environment. 72, 92

LFR Linear Four Rates. 65, 66

LRU Least Recently Used. 108, 112, 114, 117, 161

MC Majority Class. 31

MOA Massive Online Analysis. 31, 106

MSE Mean Squared prediction Error. 56, 57, 87, 155

NB Naïve Bayes. 31, 93

NBAdaptive Naïve Bayes Adaptive. 31

OAUE Online Accuracy Update Ensemble. 23, 54, 56, 57, 87, 95, 109, 121, 122, 139, 146, 147, 152, 153, 155–157

OB Online Bagging. 23, 25, 51, 57, 61, 67, 68, 190, 195, 225, 227, 228, 231, 235

OLIN Online Information Network. 43

ONN Online Neural Network. 57

OOB Oversampling-based OnlineBagging. 25, 26, 65, 67, 68, 90, 190, 194, 195, 224, 226–228, 231, 232, 235, 241, 242, 248, 270, 271, 283, 284, 295–297, 304, 305, 309

oOS OnlineOversampling. 26, 271, 284, 295–297, 304, 305, 309

OS-ELM Online Sequential Extreme Learning Machine. 62, 63, 70, 71, 89, 91

oSMOTEB OnlineSMOTEBagging. 91, 241, 242, 271

oUnderOverB OnlineUnderOverBagging. 26, 89, 241, 242, 271, 279, 283–285, 289, 291, 295–297, 304, 305, 309

OzaBag Online Bagging. 139, 146, 149, 154, 156

PAUC-PH Prequential Area Under Curve in Page Hinkley Test. 65–67, 196

PCA principle component analysis. 62

PCI-OSELM Online Sequential Extreme Learning Machine (OS-ELM) with Improved SMOTE. 59, 62, 63, 89

PHT Page-Hinckley Test. 45, 46, 49, 50, 67

PL Paired Learners. 49

RCD Recurrent Concept Drift. 23, 45, 52–54, 86, 94, 95, 108, 120–122, 139, 146

RLSACP Recursive Least Square Adaptive Cost Perceptron. 57

SMOClust Synthetic Minority Oversampling based on Stream Clustering. 242–245, 247–250, 253, 261, 262, 264, 267, 278–280, 282–285, 288–292, 295–297, 300, 301, 304, 305, 308–313, 316, 324–326

SMOGauNoise Synthetic Minority Oversampling based on Gaussian Noise. 26, 271, 284, 295–297, 304, 305, 309

SMOTE Synthetic Minority Over-sampling Technique. 29, 30, 34, 35, 58, 59, 61, 89, 91, 92

STEPD Statistical Test of Equal Proportions. 49

UOB Undersampling-based OnlineBagging. 25, 65, 67, 68, 90, 190, 194, 195, 224, 226–228, 231–233, 235–237, 241, 242, 248, 270, 271, 279, 283, 285, 289–291, 295–297, 302, 304, 305, 309, 325, 326

VFC-SMOTE Very Fast Continuous Synthetic Minority Oversampling Technique. 69, 70

VFDT Very Fast Decision Tree. 30, 60

WELM Weighted Extreme Learning Machines. 70, 71

Chapter One

Introduction

Machine learning is one of the research fields in Artificial Intelligence. It concerns how to develop computational models that can acquire knowledge from the environment by themselves and automatically improve their predictive performance with experience (Mitchell, 1997). Machine learning can be applied to different tasks, for example, classification, regression, pattern recognition, etc. In the last decade, it has been rapidly integrated into our lives, ranging from data mining programs of fraudulent behaviours detection, autonomous vehicles, and to help us fighting various diseases.

Traditionally, most machine learning approaches require all training data from a given task to be available before training starts. They will only be used to perform the task after the training is finished, which means they cannot perform the task while training and be further improved or changed once the training phase is finished. This kind of machine learning approaches are referred to as "offline learning" approaches (Gama, Žliobaitė, et al., 2014). However, with the fast growth of the number of digital devices in this era, the volume and the incoming speed of data are rapidly increasing. It is impractical and often infeasible to wait for the availability of all data or to store them into a machine's main memory before the machine learning approach starts to analyse them (Gama, Žliobaitė, et al., 2014). Thus,

it is important to design machine learning approaches to learn data in the form of streams (data streams), such that they can analyse the data upon arrival and perform the task while learning. This kind of machine learning approaches are referred to as "data stream learning" approaches (Ditzler, Roveri, et al., 2015).

Data stream learning approaches are more suitable than offline learning approaches in dealing with large amount of high speed incoming data, as they can perform the task while learning. Data stream learning is often performed in the form of incremental or online learning. Increment learning refers to approaches that can update their knowledge over time (Polikar et al., 2001). With strategies to adapt to concept drift, they can continuously learn from infinite batches of data. Online learning has a stricter training requirement. It requires the data stream learning approach to process each training example upon arrival and sequentially (Oza, 2005).

In most real-world applications, the underlying data distribution of the task can change over time, resulting a phenomenon called "concept drift" (Krawczyk et al., 2017). When concept drift occurs, the predictive performance of the machine learning approach starts to degrade. Depending on the speed and the severity of the drift, the machine learning approach requires different strategies to adapt to it. For example, consider a credit card fraud detection problem (Dal Pozzolo et al., 2017), where the machine learning approach learns to spot fraudulent transactions from the genuine transactions. Delaying or failing in detecting fraudulent transactions may cost companies and clients a fortune. Thus, the machine learning approach has to be continuously updated and able to make predictions in real-time. What is more, new fraud strategies may be developed over time. The machine learning approach needs to adapt to changes in a timely manner.

Another example is software defect prediction (G. G. Cabral et al., 2019), where a machine learning approach is employed by a software development company to detect which

software changes submitted by software engineers are likely to be faulty. To reduce the debugging cost, faulty changes should ideally be identified right after the submission, such that the software engineers can fix it immediately. As the number of changes in software project could be enormous, the machine learning approach is required to update continuously and make predictions in real-time. Also, the software development flow used by the company may change over time and new software engineers with different coding style may be hired. The machine learning approach needs to be capable of adapting to these changes. Other examples include real-world applications where the data arrive continuously in the form of data streams, such as software effort estimation (Minku and Hou, 2017), spam filtering (Delany et al., 2005) and soft sensors in process industry (Kadlec, Gabrys, and Strandt, 2009).

In addition, a sub category of concept drift called recurrent drift or recurrent concept is an interesting topic in this research field. It is referred to as a concept drift bringing back a concept (data distribution) that had appeared in the past (Gonçalves Jr and Barros, 2013). If we can identify it successfully, we could potentially exploit past knowledge in the system's memory to deal with it swiftly (Ditzler, Roveri, et al., 2015). This type of concept drift occurs frequently in real-world applications. For examples, the season changes cyclically in weather forecasting, reusing past development flow or hiring former software engineers in software defect prediction or software effort estimation. Thus, this thesis has a particular interest in handling this type of concept drift.

Despite identifying relevant past knowledge is pivotal in dealing with drifts, strategy to decide which past models to preserve in the memory is also important because relevant past knowledge could not be exploited if it does not exist in the memory. Existing strategies remove oldest past models (Gonçalves Jr and Barros, 2013; Minku and Yao, 2012) or models that are irrelevant to the current underling concept (Brzeziński and Stefanowski, 2014; Kolter and Maloof, 2003) are not always ideal. For example, a past concept could recurs after many

concept drifts to other distinct concepts, and it is very different from the current concept. In this case, the relevant past knowledge (model) is likely to be removed by the aforementioned strategies. However, it is hard to find any existing work to investigate the impact of these strategies to the overall predictive performance. We refer to these strategies as *memory management strategies*, while strategies to recover which past model for learning are referred to as *memory recovery strategies*. Also, we refer to any approach that exploit past knowledge to deal with concept drift through memory management strategies and memory recovery strategies as *memory-based framework*.

Besides concept drift, another common phenomenon that occurs in many data stream classification tasks is "class imbalance". It refers to a situation where the number of examples from one class is larger than the other classes (S. Wang, Minku, and Yao, 2018). In binary classification, which is the focus of this thesis, the former class is referred to as "majority class" while the latter class is referred to as "minority class". When this phenomenon occurs in a data set or a data stream, we have relatively little knowledge about the minority class as the class distribution is skewed. The machine learning approaches are likely to learn with bias towards the majority class, thus, they tend to misclassify minority class examples. Yet, very often, the minority class is the class of interest in the problem. Thus, failing in recognising minority class examples may lead to very high cost. Reusing the above real-world application examples, class imbalance can also occurs simultaneously with concept drift. For example, in credit card fraud detection, the number of genuine transactions is normally larger than the number of fraudulent transactions. While in software defect prediction, faulty changes are less likely to occur than faultless changes.

In the literature, diversity is one of the popular perspectives to deal with concept drift in data stream learning (Minku and Yao, 2012; Kuncheva and Whitaker, 2003; Minku, White, and Yao, 2010). Although there is no formal definition of diversity in the literature, it is usually referred to as a level of disagreement among a set of computational models.

Existing work have shown that applying diversity in the main ensemble of a machine learning approach has positive impact to the predictive performance (Minku and Yao, 2012; Kuncheva and Whitaker, 2003; Minku, White, and Yao, 2010). Yet, they investigated the impact of applying diversity in a given main ensemble but the potential of diversity could possibly go beyond this aspect and could used as a overall framework to handle multiple types of drifts. This thesis first presents a study to discuss the possibility of applying diversity in a memory management perspective to deal with recurrent concepts. Then, it extends this idea to create a novel diversity-based framework to handle multiple types of concept drift. For simplicity, this framework is proposed to deal with binary classification tasks. Yet, it can be extended to multi-class problems when a diversity measure for multi-class classification is employed and the clustering in the model space strategy relies on the models' prediction rather than prediction correctness.

Although there is rich literature investigating different approaches to deal with class imbalance and concept drift separately, there is still little work on investigating how to deal with their joint challenge (S. Wang, Minku, and Yao, 2018). Exploiting relevant past knowledge could also be an intuitive method to deal with concept drift in class imbalanced data streams because there exists particularly few information (examples) from the minority class of the post-drift concept. Therefore, memory strategies could be particularly relevant for drifting class imbalanced data stream problems. In this thesis, we contribute towards this area by extending the idea of using diversity as a general framework to deal with drifting class imbalanced data streams.

Nonetheless, we still have some situations where learning drifting class imbalanced data stream cannot be fully addressed only by managing the memory based on diversity. For example, at the beginning of a data stream or when there are concept drifts leading to novel concepts, exploiting relevant past knowledge through diversity becomes infeasible as no relevant knowledge exists in the system's memory. Also, the learning bias towards

the majority class needs to be resolved. With these in mind, strategy of creating synthetic minority class examples could be a possible and intuitive solution. This kind of strategy has shown to be successful in offline learning (He, Bai, et al., 2008; N. V. Chawla et al., 2002; Han, W.-Y. Wang, and B.-H. Mao, 2005; Lee, J. Kim, and S. Kim, 2017). However, they cannot be applied to data stream learning directly because they require to store all the minority class examples the system have seen into the memory, which is infeasible in data stream learning.

Very recent work attempted to apply synthetic data generation into the field of data stream learning (Bernardo, H. M. Gomes, et al., 2020; Bernardo and Della Valle, 2021). They adopted an adaptive sliding window (ADWIN (Bifet and Gavaldà, 2007)) to store the most recent batch of minority class examples and rely on ADWIN to split then drop the old window of examples when concept drift has likely occurred. However, ADWIN may not be very effective in detecting concept drift in class imbalanced data streams, because it detects drift based on the error rate of the system. Therefore, when it split and drop the old window incorrectly, the remaining window may still contain some examples (information) belonging to the concept prior to the drift. When synthesising minority class examples based on this window, noisy examples could then be created, thus hindering the predictive performance of the system. Some other existing work, such as PCI-OSELM (W. Mao, J. Wang, and L. Wang, 2015), require additional examples for initialisation before training starts. Yet, this may not always be practical because examples are expensive to collect in most real-world applications, especially the minority class examples. In the light of this, the last study of this thesis devises a novel strategy of creating synthetic minority class examples for oversampling that is suitable for data stream learning.

It is also worth noting that the proposed approaches in this thesis are designed for supervised data stream classification, i.e., the true classification label of the most recent example is available after the approach made a prediction to it.

This section is further branched as follows. Section 1.1 formulates the problem of class imbalanced data stream learning in the presence of concept drift with details and the formal definitions of the terminology mentioned in the above. Section 1.2 presents and explains the research questions answered by this thesis. Section 1.3 summarises the contributions of the thesis and presents the thesis organisation.

1.1 Definitions and Problem Formulation

This section provides a set of definitions of terms that are frequently used in this thesis and it is organised as follows. Section 1.1.1 provides the definitions of incremental, batch-based and online learning. Section 1.1.2 provides the definition of class imbalance. Section 1.1.3 provides the definition and categorisations of concept drift.

1.1.1 Offline, Incremental and Online Learning

Traditionally, most machine learning approaches operate in "offline" mode or usually called "offline learning". They first train on the training set S then perform the task on unseen data. The formal definition is as follows (Minku, 2019; Gama, Žliobaitė, et al., 2014):

Definition 1.1.1 (Offline Learning)**.** Offline learning refers to use a finite training set S of n examples to build a machine learning approach $f : X \mapsto Y$, such that this approach is able to generalise the unseen examples $(x_i, y_i) \sim P(x, y)$, where $i > n$. It is assumed that $S \sim_{i.i.d.} P(x, y)$, where $P(x, y)$ is the underlying joint probability distribution.

As it has been discussed at the beginning of this chapter, the volume and incoming speed of data are rapidly increasing. Thus, data in many real-world applications arrive in

the form of a stream, i.e. a data stream, which can be formally defined as follows (Minku, 2019; Žliobaitė, 2010; Gama, Žliobaitė, et al., 2014; Ditzler, Roveri, et al., 2015):

Definition 1.1.2 (Data Stream)**.** A data stream is a (potentially) infinite sequence of training sets $S =< s_1, s_2, ... >$, where $s_t = \{(x_i^t, y_i^t)\}_{i=1}^{n_t}$, $n_t > 0$ is the number of examples in the training set s_t, and $t > 0$ is the time step, which is a sequential identifier denoting the time moment of receiving the training set s_t. It is assumed that $s_t \sim_{i.i.d.} P_t(X, Y)$, where $P_t(X, Y)$ is the underlying joint probability distribution at time step t.

In particular, this thesis focuses on binary classification task in the context of supervised data stream learning, which can be formally defined as follows (Minku, 2019):

Definition 1.1.3 (Binary Classification Task in Data Stream Learning)**.** For any training set s_t in a data stream at time step t, each example has the format $(x_t, y_t) \in X \times Y$, where $x_t = (x_t^{(1)}, x_t^{(2)}, ..., x_t^{(m)}) \in X$. X is the input attribute space with m dimensions and Y is the class attribute space, thus, $y_t \in Y$. In binary classification tasks, the class attribute space only has two classes, i.e. $Y = \{c_0, c_1\}$, where c_i is the class label of the $(i+1)$-th class.

In the literature, two terms have been frequently used to describe different forms and requirements of data stream learning: incremental learning and online learning. However, researchers may sometimes refer them to as other meanings or describe them with different terms. For instance, Polikar et al. (2001) pointed out that the term “incremental learning" may sometimes refer to as incremental network growing and pruning, online learning, or relearning of formerly misclassified instances. Other terms have also been used in the literature to imply incremental learning, such as lifelong learning, constructive learning etc. Bruzzone and Fernández Prieto (1999) refer it to as classifiers that are able to acquire new knowledge when new training sets become available, while preserving the knowledge it acquired in the past. Utgoff (1989) gave a similar definition and defined that incremental learning

approaches revise the current concept definition, if necessary, in response to each newly observed training example and this type of approaches is appropriate for learning data stream. Ferrer-Troyano, Aguilar-Ruiz, and Riquelme (2005) defined that, in incremental learning, the whole training set is not available in advance but the examples arrive over time, usually one example at a time and not time-dependent necessarily.

In this thesis, we adopt the definition of incremental learning provided by Polikar et al. (2001), which gives a clear set of criteria that an incremental learning algorithm should follow:

1. It should be able to learn additional information from new data.
2. It should not require access to the original data, used to train the existing classifier.
3. It should preserve previously acquired knowledge. That is, it should not suffer from catastrophic forgetting.
4. It should be able to accommodate new classes that may be introduced with new data.

Small modification to the above definition is performed. In this work, we allow the access to past training data for purposes other than retraining the existing classifier. Also, we make it clear that only useful past knowledge should be persevered. The term "useful past knowledge" here does not only refer to knowledge that can help to learn the current underlying joint probability distribution of the data stream, but also can help to adapt to various future changes of the underlying distribution. Lastly, we remove the last requirement by Polikar et al. (2001), because this thesis focuses on binary classification tasks, thus, assumes that all the examples in the data stream only belong to either one of two classes.

Similarly to the term incremental learning, the term "online learning" could sometimes be ambiguous in the literature. For example, Oza (2005) defined that online learning

algorithms process each training examples once "on arrival" without the need for storage and reprocessing, and maintain a current model that reflects all the training examples seen so far. Littlestone and Warmuth (1994)'s definition to online learning is strict and also took the way of evaluation into the consideration. They refer it to as learning proceeds in a sequence of trials. In each trial, the algorithm receives an instance from some fixed domain and is to produce a binary prediction. At the end of the trial the algorithm receives a binary label, which can be viewed as the correct prediction for the instance. Gama, Žliobaitė, et al. (2014) gave a relaxed definition to online learning, which is quite similar to the definition of incremental learning. They defined that online learning predictive models can be trained either incrementally by continuous update or by retraining using recent batches of data. As we can see, the definitions of incremental learning and online learning are sometimes interchangeable but, in most cases, online learning is defined as a stricter scenario of incremental learning.

In this thesis, we adopt the definition of online learning provided by Polikar et al. (2001). Though, we perform a small modification to it. We refer online learning to as data stream learning approaches process each training example sequentially. We allow the storage of the training examples in the system's memory but the learning approach is not allowed to retrieve then reuse them for retraining purposes. Yet, reusing examples for training upon their arrival is acceptable as the examples are yet to be stored in this situation. Also, the learning approach is required to maintain a current model that reflects the training examples belonging to the current underlying joint probability distribution of the data stream rather than all the training examples seen so far. It is because underlying joint probability distribution of the data stream may be changed over time, which will be discussed in Section 1.1.3.

In contrast to online learning, other types of incremental learning approaches are allowed to learn from the data stream by batches of new training data, which usually is

referred to as batch-based or chunk-based learning. Following Definition 1.1.2, batch-based learning refers to the number of examples arriving at time t is larger than one (i.e. $n_t > 1$), while online learning refers to the examples arrive in a sequential order (i.e. $n_t = 1$). Some batch-based learning approaches allow the learning algorithm to process each batch of training data more than once or may even apply an offline learning approach for each batch of data, whereas online learning can only learn from each training example sequentially. Thus, online learning approaches are capable of performing the task in real-time while training, whereas batch-based learning approaches may not have such capability. This thesis mainly focuses on online learning, which can be formally defined as follows (Ditzler, Roveri, et al., 2015; Minku, 2019; Gama, Žliobaitė, et al., 2014; Žliobaitė, 2010).

Definition 1.1.4 (Online Learning)**.** Online learning refers to the data stream learning algorithm to process the training examples from the data stream in a sequential order, i.e., example-by-example. The algorithm is only allowed to store training examples for purposes other than training the predictive model. Reusing examples for training upon their arrival is acceptable as the examples are yet to be stored in this situation.

1.1.2 Class Imbalance

Class imbalance is another common phenomenon in real-world classification problems (Hoens, Polikar, and N. V. Chawla, 2011; S. Wang, Minku, and Yao, 2018). Typically, it refers to the situation when one of the classes is severely underrepresented in the data set or the data stream (Hoens, Polikar, and N. V. Chawla, 2011; S. Wang, Minku, and Yao, 2018). Japkowicz (2001) and He and Garcia (2009) specifically refer it to as *between-class imbalance*. Brzezinski, Minku, et al. (2021) used the term *global imbalance ratio* to describe the percentage of minority class examples in the data set, thus, referring to class imbalance.

In offline learning, class imbalance can be formally written as $|\{(x, y_i)\}| << |\{(x, y_j)\}|$,

for any $i \neq j$ and $y_i, y_j \in Y$, which means the number of examples in one class is far less than the number of examples in at least one other class in a given data set (He and Garcia, 2009; Hoens, Polikar, and N. V. Chawla, 2011; S. Wang, Minku, and Yao, 2018). In data stream learning, it is referred to as the probability of seeing an example from one class is far less than the other classes at any time step t, i.e. $P_t(y_i) << P_t(y_j)$, for any $i \neq j$ and $y_i, y_j \in Y$. Class imbalance can occur in both binary and multi-class classification problems, but we focus on binary classification in this thesis. Conventionally, the class with more example is referred to as the *majority* or *negative* class or class 0 while the class with less examples is referred to as the *minority* or *positive* class or class 1. Usually, the minority class is the class of interest in the classification problem (He and Garcia, 2009; S. Wang, Minku, and Yao, 2018), such as the fraudulent class in credit card fraud detection, attack class in intrusion detection, and faulty changes in software defect prediction. Thus, misclassifying minority class examples generally results in higher cost than misclassifying majority class examples.

Apart from class imbalance, existing work showed that *data difficulty* or *data complexity* is another key factor that deteriorates classification performance (Batista, Prati, and Monard, 2005; Prati, Batista, and Monard, 2004; Jo and Japkowicz, 2004; Brzezinski, Minku, et al., 2021). Napierala and Stefanowski (2015) summarise the data difficulty factors from the literature:

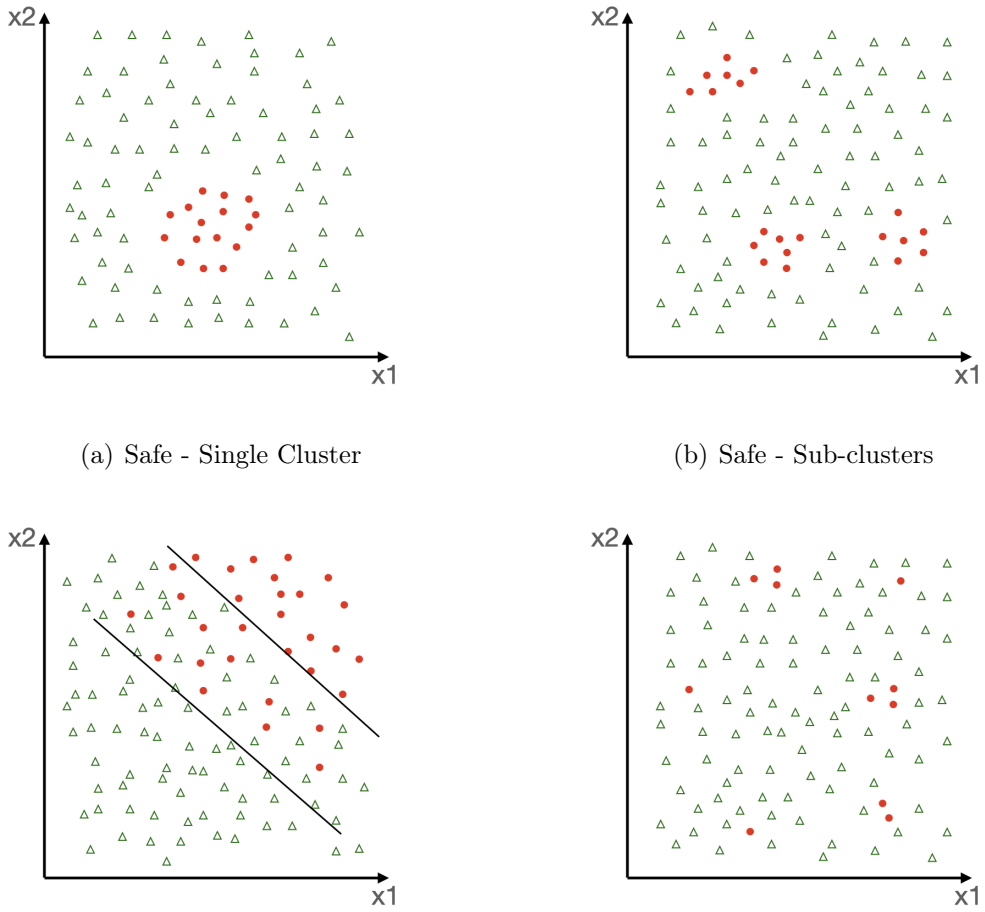
1. the decomposition of the minority class concept into several sub-concepts (sub-clusters).
2. the presence of small, isolated groups of minority examples located deep inside the majority class region;
3. the effect of strong overlapping between the classes.

The first factor in the above refers to a phenomenon called *class decomposition* where the minority class does not form a homogeneous region in the attribute (feature) space but

several small sub-clusters spreading over the space (He and Garcia, 2009). When some sub-clusters have fewer examples than the others, Japkowicz (2001) refers it to as *within-class imbalance*. Several studies investigated the impact of class decomposition to the predictive performance of popular classifiers with artificial data sets (Napierala and Stefanowski, 2015; Brzezinski, Minku, et al., 2021; Japkowicz and Stephen, 2002; Jo and Japkowicz, 2004) and showed that increasing the degree of decomposition in the minority class concept can have a higher impact than solely increasing the global imbalance ratio. Yet, severe class imbalance ratio exacerbates the challenge of class decomposition (Napierala and Stefanowski, 2015; Brzezinski, Minku, et al., 2021).

The second and the third factors in the above are related to different types of minority class example: safe, borderline, rare, and outliers (Napierala and Stefanowski, 2015; Napierala and Stefanowski, 2012; Brzezinski, Minku, et al., 2021; S. Wang, Minku, and Yao, 2018). Figure 1.1 presents the their illustration. Safe examples are located within the sub-clusters of the class. Borderline examples are scattered over the region of the decision boundaries between classes, which is related to the third factor in the above. Some studies investigated the impact of different overlapping degrees between classes to classification performance and referred such issue to as *class overlapping* (Batista, Prati, and Monard, 2005; Prati, Batista, and Monard, 2004). Rare and outlier examples are difficult to identify as they appear as groups of a few or single minority class example respectively, which are isolated and located deep inside the regions of the majority class (the second factor in the above). As the minority class is underrepresented, rare and outlier cases should not be considered as noise (Kubat and Matwin, 1997).

Note that, the aforementioned data difficulty factors and class imbalance are not mutually exclusive. For example, a given data stream can be class imbalanced with the minority class concept decomposed into several sub-clusters. At the same time, the number of borderline examples can be more than safe examples while there are some rare and outlier examples



(a) Safe - Single Cluster

(b) Safe - Sub-clusters

(c) Borderline Region

(d) Rare and Outlier examples

Figure 1.1: Illustration of Minority Class Example Types

sprinkle over the whole attribute (feature) space. Thus, when designing machine learning algorithm to deal with class imbalanced data set, it is important to take the aforementioned data difficulty factors into the consideration.

In this thesis, we focus on binary data stream classification. We refer class imbalance to as the probability of seeing an example from one class being significantly less than the other class in a data stream. Also, we adopt the categorisation of drift difficulty factors by Brzezinski, Minku, et al. (2021) because they provided a detailed study on the impact of different drift difficulty factors to classification performance of several popular classifiers

in data stream setting. These drift difficulty factors are splitting, moving, and merging minority class sub-clusters, changing the proportions of safe, borderline, rare and outlier minority class examples, and changing the class imbalance ratio of the data stream. The scope of investigating class imbalance and drift difficulty factors in Chapters 5 to 6 are specified in their corresponding introduction.

1.1.3 Concept Drift

Data Streams typically suffer changes in the underlying joint probability distribution, a phenomenon commonly referred to as *concept drift*. The underlying joint probability distribution ($p(x, y)$) is referred to as a *concept* (Krawczyk et al., 2017). In the literature, researchers may refer such phenomenon using different terms. For example, Moreno-Torres et al. (2012) refers to a change in the joint probability distribution as data set shift, while they refer to a sub category of data set shift affecting the decision boundary as concept drift. Kelly, Hand, and Adams (1999) refer to it as population drift.

Bayesian decision theory (Duda, Hart, and Stork, 2001) is commonly used to provide the formal definition of *concept* and, thus, defining *concept drift* (Žliobaitė, 2010; Ditzler, Roveri, et al., 2015; Gama, Žliobaitė, et al., 2014). In this thesis, we adopt the definitions provided by Gama, Žliobaitė, et al. (2014), where a concept is a joint probability distribution constituted by the prior class probabilities $P(Y)$ and class conditional probabilities $P(X|Y)$, i.e. $P(X, Y) = P(Y)P(X|Y)$. Thus, a concept drift is referred to as the situation where $P_{t_a}(X, Y) \neq P_{t_b}(X, Y)$, for any time steps $t_a \neq t_b$. This means that concept drift is a change affecting one or both of the following components:

- the prior probabilities of classes $P(y_i)$,
- the class conditional probabilities $P(X|y_i)$, where $y_i \in Y$.

By the Bayes theorem, a joint probability distribution (concept) can also be written as $P(X, Y) = P(X)P(Y|X)$. Thus, a concept drift can also be a change affecting one or both of the following components:

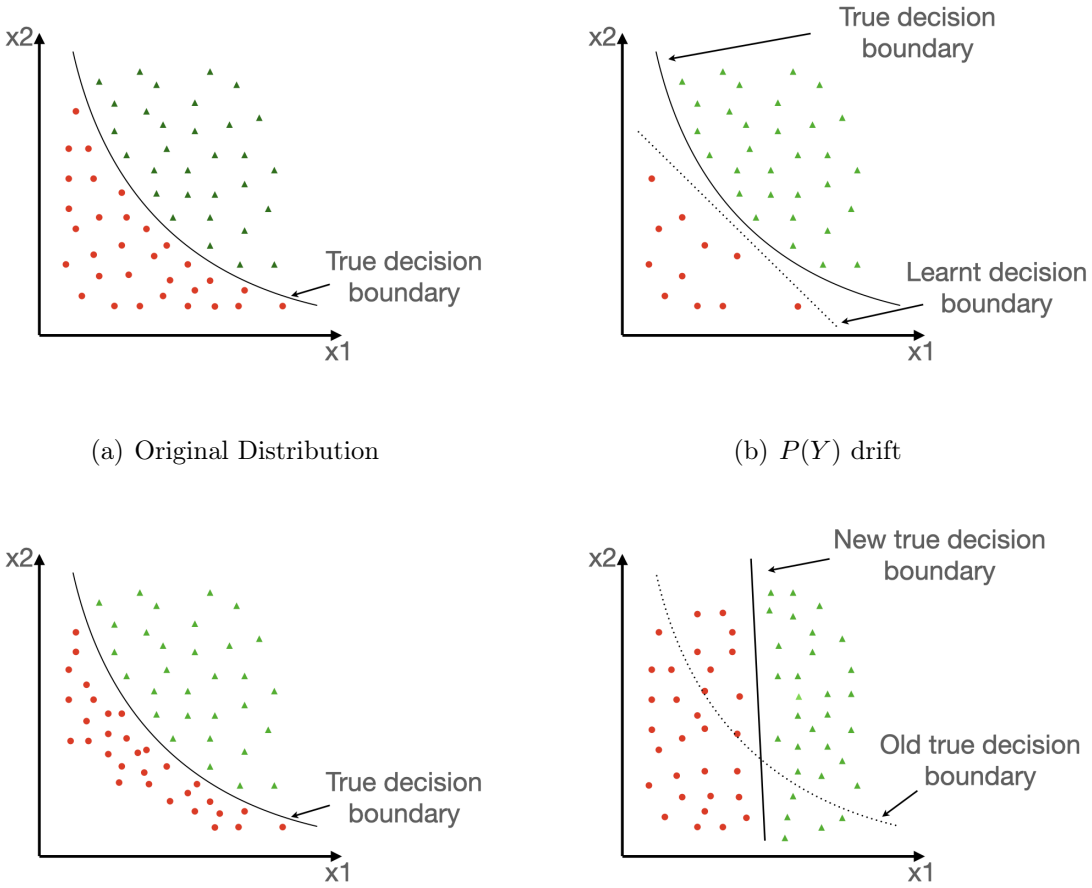
- the unconditional probability density function (pdf) $P(X)$, and
- the posterior probabilities of classes $P(y_i|X)$, where $y_i \in Y$.

Figure 1.2 illustrates concept drifts affecting different components if the joint probability distribution.

Depending on which of the above components are affected, concept drift can be further divided into two types: *real concept drift* and *virtual concept drift* (Ditzler, Roveri, et al., 2015).

Real concept drift refers to changes in the posterior probabilities of classes $P(Y|X)$, regardless changes in the unconditional pdf $P(X)$ (Hoens, Polikar, and N. V. Chawla, 2011; Gama, Žliobaitė, et al., 2014; Ditzler, Roveri, et al., 2015; Webb et al., 2016; S. Wang, Minku, and Yao, 2018). As illustrated in Figure 1.2(d), the true decision boundary is changed after the drift and the previously learnt discriminative function does not valid anymore (S. Wang, Minku, and Yao, 2018). It can be formally written as $P_{t_a}(Y|X) \neq P_{t_b}(Y|X)$, for any time steps $t_a \neq t_b$ (Webb et al., 2016). A real-world application example is that novel fraudulent strategies are developed in credit fraud detection.

Virtual concept drift is commonly referred to as a change in the unconditional pdf $P(X)$, without affecting the posterior probabilities of classes $P(Y|X)$ (Gama, Žliobaitė, et al., 2014; Ditzler, Roveri, et al., 2015; G. Oliveira, Minku, and A. L. I. Oliveira, 2021), i.e. $P_{t_a}(X) \neq P_{t_b}(X)$, for any time steps $t_a \neq t_b$ (Webb et al., 2016). Hoens, Polikar, and N. V. Chawla (2011), S. Wang, Minku, and Yao (2018), and Žliobaitė (2010) refer it to



(a) Original Distribution

(b) $P(Y)$ drift

(c) $P(X|Y)$ drift

(d) $P(Y|X)$ drift

Figure 1.2: Illustration of Concept Drift Types

as any changes that do not affect the posterior probabilities of classes $P(Y|X)$ (the true decision boundaries between classes). It can happen solely in the prior probability of the classes $P(Y)$ or in the class conditional pdfs $P(X|Y)$. Figure 1.2(b) illustrates virtual drift in the prior probability of the classes $P(Y)$ only. We can see that it does not affect the true decision boundaries between classes but it affects the class imbalanced ratio. A real-world application example could be the number of patients of a given disease increases or decreases. Figure 1.2(c) illustrates virtual drift affecting the class conditional pdfs $P(X|Y)$ only. In this case, the true decision boundary between the classes remains the same but the shape

of the decision region of circle class has changes. A real-world application example is that, given the number of patients of a given disease remains the same but the age group of the patients has shifted. In this thesis, we adopt the latter definition of virtual concept drift because it considered changes in the prior probability of the classes $P(Y)$ which is related to another important topic in this thesis – class imbalance. Further details regarding class imbalance is provided in Section 1.1.2.

In addition, concept drift can sometimes leads to previously seen concept, which is commonly referred to as reoccurring (recurring) concepts or recurrent drift (Minku, White, and Yao, 2010; Gama, Žliobaitė, et al., 2014). For example, in weather forecasting, different seasons may cause drifts lead to previously seen environments. Also, it is worth to note that recurrent drift may lead to a concept that is similar to a previous one, but not the same (Minku, White, and Yao, 2010). In both cases, exploiting previously acquired knowledge could be helpful to swiftly adapt to the drift (Ditzler, Roveri, et al., 2015).

Besides, Gama, Žliobaitė, et al. (2014) also reminds the challenge of distinguishing outlier or noise from concept drifts. They refer to outlier or noise as a once-off random deviation or anomaly where no adaptation is required (Gama, Žliobaitė, et al., 2014).

Speed is another criterion in the literature to describe the changing rate of the concept drift. It categorises concept drifts as sudden/abrupt or gradual. Typically, sudden or abrupt drift is referred to as changes occurring in a very short period, while gradual drift is referred to as the underlying concept changs slowly over time (Gama, Žliobaitė, et al., 2014; Brzezinski, Minku, et al., 2021; Hoens, Polikar, and N. V. Chawla, 2011). Minku, White, and Yao (2010) refer to abrupt drift as concept drifts resulting in a complete change in only one time step, while gradual drift is referred to as concept drifts that take more than one time step to complete. They further branched gradual drift to probabilistic and continuous gradual drift. Probabilistic gradual drift is referred to the new concept gradually takes over the old

one and some examples may still belong to the old concept, while continuous gradual drift is referred to as the concept itself gradually and continuously changes from the old to the new concept, by suffering changes between every consecutive time steps (Minku, White, and Yao, 2010). Gama, Žliobaitė, et al. (2014) distinguish incremental drift from gradual drift. They refer to incremental drift as the underlying concept slowly evolving and consisting of many intermediate concepts in between, which is similar to the definition of continuous gradual drift given by Minku, White, and Yao (2010). They then refer to gradual drift as two concepts existing at the same time with the new concept slowly taking over the old one, which is similar to the definition of probabilistic gradual drift given by Minku, White, and Yao (2010). Nevertheless, both types of gradual drift result in intermediate concepts during the transitional period of the drift. Thus, they may be harder to handle than abrupt drift. In this thesis, we solely concern the speed of change to distinguish abrupt and gradual drifts. We refer to a concept drift as abrupt when it completes the change in a single time step, or to as gradual in otherwise (Minku, White, and Yao, 2010).

Severity is another important criteria in categorising concept drifts. It divides concept drifts into *full concept* and *subconcept* drifts (Webb et al., 2016), which are also referred to as *severe* and *intersected* drifts (Minku, White, and Yao, 2010). *Full concept* drift is referred to as concept drifts affecting the whole attribute space, while *subconcept* drift is referred to as concept drifts affecting only a part of the attribute space (Minku, White, and Yao, 2010; Webb et al., 2016). Depending on the changes affecting the class prior/posterior probabilities or the unconditional/class-conditional pdfs, concept drift severity can be further divided into class and feature severity respectively (Minku, White, and Yao, 2010). In this thesis, we focus on the class severity and use the percentage difference between the concepts before and after the drift as a measure to indicate the level of severity (Minku, White, and Yao, 2010). The percentage difference between two concepts A and B is calculated as follows:

$$diff(A, B) = \frac{\sum_{i=1}^{n} |y_i^A - y_i^B|}{n} \tag{1.1}$$

where y_i^A and y_i^B are the class labels determined by the concepts A and B respectively, and n is the total number of examples generated uniformly at random to calculate the severity. We consider a concept drift to be severe when the concepts before and after the drift have at least around 50% difference, while around 25% difference is referred to as a mild severity drift.

Other criteria have also been used in the literature to further characterise concept drift, such as predictability, frequency and data difficulty factors (Minku, White, and Yao, 2010; Kosina, Gama, and Raquel Sebastião, 2010; Brzezinski, Minku, et al., 2021). Note that, the above criteria can be used at the same time to provide a rich description of a concept drift. For example, a real concept drift can be severe, gradual and recurrent at the same time. In this thesis, we focus on discussing real concept drift, drift severity, recurring concepts, and data difficulty factor drifts. The scope of investigating concept drift in Chapters 4 to 6 are specified in their corresponding introduction.

1.2 Research Questions

This section presents the research questions answered by this thesis and explains the motivation behind each of them. Detailed motivations behind the approaches to address each question/problem are presented in the corresponding sections of the proposed solutions.

1.2.1 Diversity-based Memory Strategies for Concept Drift Adaptation in Class Balanced Data Streams

Research Question 1 (RQ1): Can diversity be used to create a memory-based framework able to tackle different types of drifts, including recurrent, abrupt, gradual, severe and mild-severe drifts more efficiently and effectively? How well does this framework perform?

Most existing approaches in the context of concept drift adaptation focus on how to detect and adapt to concept drifts swiftly. They tend to gradually forget past knowledge and learn new models for new concepts. However, the past knowledge can sometimes be useful in adapting to concept drifts. This includes not only recurring drifts but also drifts leading to a concept that is similar to one of the past concepts. Also, the past knowledge could be particularly helpful in dealing with concept drift in class imbalanced data streams because there exists very few examples from the minority class concept after the drift. Existing approaches are likely to lose the advantage of exploiting past knowledge in these situations.

Several existing approaches attempted to explicitly handle concept drift by exploiting past knowledge (Gonçalves Jr and Barros, 2013; Minku and Yao, 2012; Mirza, Lin, and Liu, 2015). Evolving ensemble methods, such as OAUE (Brzeziński and Stefanowski, 2014), DWM (Kolter and Maloof, 2003) and DWMIL (Lu, Cheung, and Tang, 2017), can also be perceived as approaches exploiting past knowledge to handle concept drift, as they maintain a certain amount of past knowledge in the main ensemble while evolving to adapt to the concept drift. Yet, these existing approaches either forget the past knowledge by temporal order or based on their suitability to the current concept. In situations where a recurring drift or a severe drift leading a concept that is similar to one of the past concepts occurs after several drifts, these approaches are likely to have forgotten the relevant past knowledge by the time such drift occurs.

As we can see, strategies to decide which models to keep in the memory (memory management strategies) can be important to concept drift adaptation. Yet, no study of handling concept drift from a memory perspective has been done in the literature. To fill this research gap, we need to investigate the memory strategies that exploit the full potential of using relevant past knowledge to adapt concept drift. In particular, we need to first investigate them in class balanced data stream, such that we can evaluate them without being affected by class imbalance. Therefore, the work proposed to answer research question 1 are in the context of drifting class balanced data streams.

In Section 4.1 a novel approach that uses a diversity-based memory management strategy called Diversity Pool (DP) is proposed, partially answering research question 1. Section 4.1.3 shows that using diversity as the memory management strategy helps to achieve promising results in handling recurring drift and drifts leading a concept that is similar to one of the past concepts. This is because maintaining a diverse set of models in the memory optimises the amount of distinct knowledge in the memory. Thus, it increases the chance of having relevant knowledge to exploit when a concept drift occurs.

However, the relationship between a diverse set of models in the memory and the predictive performance of the system is not straightforward. Section 4.1.3 shows that, the higher predictive performance achieved by exploiting correct past model could sometimes lead to a subsequent poor predictive performance because it made the drift detection method sensitive to change in predictive performance, resulting in frequent false-positive drift detections. Also, despite the novel diversity-based memory management strategy helping the system to be ready for different concepts to reoccur, the system still requires an efficient and effective strategy to recover relevant past knowledge to handle concept drifts. We denote this kind of strategy to as memory recovery strategy or model selection strategy.

Further investigating the idea of using diversity as memory strategies, Section 4.2

completes the answer to research question 1 by proposing a novel diversity-based framework to handle multiple types of concept drift called Concept Drift Handling Based on Clustering in the Model Space (CDCMS). In particular, a novel clustering in the model space strategy has been proposed to recover relevant past knowledge and to build ensembles with different diversity levels. This strategy is based on the idea that models having similar knowledge are close to each other in the model space, thus, clustering them in the model space enables us to identify past models holding knowledge that is similar to the current underlying concept.

Section 4.2.3 shows that CDCMS achieved similar or better predictive performance than Diversity Pool (DP) (Chiu and Minku, 2018) and five existing approaches: Hoeffding Tree with Naïve Bayes at Leaves (HTNB) (Domingos and Hulten, 2000), Online Bagging (OB) (Oza, 2005), Online Accuracy Update Ensemble (OAUE) (Brzeziński and Stefanowski, 2014), Diversity for Dealing with Drifts (DDD) (Minku and Yao, 2012), and Recurrent Concept Drift (RCD) (Gonçalves Jr and Barros, 2013), in both artificial and real world data streams. It performed particularly well in handling abrupt and recurring drifts in terms of predictive performance recovery speed and the overall predictive performance. Such promising performance is mainly attributed to the novel clustering in the model space strategy not only able to efficiently and effectively identify and recover relevant past knowledge to handle concept drifts, but it also helped to prevent drastic drops in predictive performance after drifts by creating a highly diverse ensemble with distinct past knowledge. CDCMS was slightly disadvantageous in data streams with multiple extremely severe gradual drifts, due to intermediate concepts are likely to form during the transitional period of the severe gradual drifts. CDCMS considered them as standalone concepts, thus, recovered incorrect past knowledge.

1.2.2 Diversity-based Framework for Concept Drift Adaptation in Class Imbalanced Data Streams

Research Question 2 (RQ2): How can memory strategies be used to tackle class imbalanced data streams with concept drifts? When and why are they helpful?

The proposed DP and CDCMS approaches which answer the research question 1 have shown how diversity can be used as a memory strategies to tackle different types of drift in class balanced data streams. However, real world applications of data stream learning frequently suffer from class imbalance and concept drift at the same time. In such condition, the minority class is difficult to learn, especially after concept drifts.

To answer research question 2, this thesis first hypothesises that relevant past knowledge can be particularly helpful to effectively adapt to the minority class after concept drift. Therefore, Chapter 5 advances the diversity-based framework to learn drifting class imbalanced data streams and proposes a framework called Concept Drift Handling Based on Clustering in the Model Space for Class Imbalanced Learning (CDCMS.CIL). Specifically, the clustering in the model space strategy and diversity-based memory management strategy were redesigned and proposed for class imbalanced learning. The analysis in Section 5.3 shows that these strategies are effective in maintaining and recovering relevant past knowledge to deal with different types of concept drift, despite the class imbalance degree of the data stream. However, the effectiveness in recovering predictive performance from the drift depends whether the recovered past models have bias towards the majority class, indicating that either a resampling method is needed or the base learners are class imbalance insensitive.

The study shows that oversampling works well in CDCMS.CIL as it helps to reducing the learning bias of the base learners to the majority class while preserving the useful knowl-

edge. Undersampling could work well in CDCMS.CIL only when the concept after the drift is novel. The performance of CDCMS.CIL is also validated through experiments on artificial and real-world data streams. In most cases, it has been shown that CDCMS.CIL performed similar or better than five existing approaches: Gaussian Hellinger Very Fast Decision Tree (GH-VFDT) (Lyon et al., 2014), Hellinger Distance Very Fast Decision Tree (HD-VFDT) (Lyon et al., 2014), Online Bagging (OB) (Oza, 2005), Oversampling-based OnlineBagging (OOB) (S. Wang, Minku, and Yao, 2015), and Undersampling-based OnlineBagging (UOB) (S. Wang, Minku, and Yao, 2015). The analysis also shows that CDCMS.CIL performed better in data streams with abrupt and abrupt recurrent drifts because no intermediate concept could form, thus, CDCMS.CIL managed to recover relevant past knowledge to handle the drift.

1.2.3 Synthetic Minority Oversampling for Learning Drifting Class Imbalanced Data Streams

Research Question 3 (RQ3): How to produce synthetic examples that enable learning the decision boundaries better in class imbalanced stream learning with concept drifts? When and why are they helpful?

The answers to research questions 1 and 2 focus on handling concept drift from the memory perspective. Yet, there exists situations where past knowledge are not available to exploit, e.g., at the beginning of the data stream or concept drifts lead to a novel concept. Strategy is still required to handle these situations. Besides, in stable concept periods, strategy is still required to reduce the learning bias towards the majority class. Traditional random oversampling and undersampling are the simplest way to achieve this purpose but they do not provide additional information about the decision boundary. Synthetic Minority Oversampling Technique (SMOTE) (N. V. Chawla et al., 2002) is a very prominent and

promising approach in offline learning. However, it requires to store all the minority class examples in the memory, which is impossible in data stream learning as we have an infinite sequence of examples.

With these in mind, Section 6.1 proposes a novel synthetic minority oversampling method for online learning called Synthetic Minority Oversampling based on Stream Clustering (SMOClust), which exploits the power of stream clustering to compress class-wise information and uses these information to synthesise minority class examples for oversampling. Experiments with artificial and real world data streams show that SMOClust performed better than Oversampling-based OnlineBagging (OOB) (S. Wang, Minku, and Yao, 2015), OnlineUnderOverBagging (oUnderOverB) (B. Wang and Pineau, 2016), OnlineOversampling (oOS) and Synthetic Minority Oversampling based on Gaussian Noise (SMOGauNoise) (Song, Minku, and Yao, 2018) in extremely class imbalanced data streams presenting high proportions of safe and borderline minority class examples, and concept drifts involving different movements of minority class sub-clusters. The main weakness of SMOClust is to handle rare and outlier minority class examples. SMOClust also performed similar or marginally worse than the same set of existing approaches in data streams with less severe class imbalanced ratio.

1.3 Thesis Contribution and Organisation

In summary, the contributions of this thesis are:

- A study on the impact of different memory management strategies to predictive performance in class balanced data streams. This study is presented in Section 4.1, Chapter 3. 4.

- A novel approach using diversity-based memory management strategy (Chiu and Minku, 2018) is proposed to partially answer research question 1. It is presented in Section 4.1, Chapter 4.

- A novel diversity-based framework to deal with multiple types of concept drift in class balanced data streams (CDCMS) (Chiu and Minku, 2022). The main novelty of this framework is the clustering in the model space strategy to identify and recover relevant past knowledge to deal with multiple types of drifts, including recurrent, abrupt, gradual, severe and mild-severe drifts. This framework answers research question 1. It is presented in Section 4.2, Chapter 4.

- A set of novel memory strategies to maintain and to recover relevant past knowledge in dealing with concept drift in class imbalanced data stream. The effectiveness of these strategies are shown to be unaffected by the class imbalance degree of the data stream. From that, a novel framework called Concept Drift Handling Based on Clustering in the Model Space for Class Imbalanced Learning (CDCMS.CIL) is proposed. These strategies answer the research question 2. They are presented in Chapter 5.

- A new approach to deal with the joint challenge of concept drift and class imbalance using a novel synthetic minority class oversampling strategy based on stream clustering (SMOClust). This approach performed particularly better in extremely severe class imbalanced data streams presenting concept drift. This approach answers the research question 3. It is presented in Chapter 6.

This thesis first gives an introduction in Chapter 1 which includes the definitions to the terms and the problem formulation. It then provides a literature review in Chapter 3 to discuss the strengths and weaknesses of the existing approaches in data stream learning. It then presents the main contributions which are the answers to the research questions 1, 2,

and 3. These answers are presented in Chapters 4, 5, and 6 respectively. Lastly, Chapter 7 summarises this thesis and discusses the directions for future work.

Some materials presented in this thesis were published in the following papers, including the design of the proposed approaches, the discussion and analysis to the corresponding experiments:

1. **Chiu, Chun Wai and Leandro L. Minku (2018).** Diversity-Based Pool of Models for Dealing with Recurring Concepts, *IEEE International Joint Conference on Neural Networks (IJCNN18)*, pp. 1–8.

2. **Chiu, Chun Wai and Leandro L. Minku (2022).** A Diversity Framework for Dealing with Multiple Types of Concept Drift Based on Clustering in the Model Space, *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, vol. 33, no. 3, pp. 1299-1309.

Chapter Two

Background

This chapter provides the background knowledge that is important to this thesis and helpful in understanding the proposed approaches. First of all, Section 2.1 introduce a classic data stream learning approach – Hoeffding Tree (HT), which is frequently used as the base learner of the proposed approaches in this thesis because of its computational speed, stability and popularity in the literature.

As discussed in Chapter 1, diversity can be particularly related to deal with concept drift and class imbalance. Chapter 4 and 5 provides studies that follow this direction and proposed three novel approaches to deal with concept drift and class imbalance. Hence, Section 2.2 introduces diversity with a set of popular measures in the literature.

Whilst diversity and exploiting relevant past knowledge can address numbers of situation in dealing with concept drift and class imbalance, past knowledge is not always available to exploit. In such a situation, synthesising minority class examples for oversampling could be an intuitive way to deal with concept drift and, particularly, class imbalance in data stream learning. Synthetic Minority Over-sampling Technique (SMOTE) (N. V. Chawla et al., 2002) is a well-known method to synthesise minority class examples for oversampling, thus addressing the class imbalance issue in finite data set learning. It is also a technique

that inspires the proposed approach in Chapter 6. Hence, Section 2.3 briefly introduces the working mechanism of SMOTE.

Lastly, Section 2.4 gives an introduction and reviews several existing stream clustering methods. Stream clustering methods are crucial to the proposed approaches in Chapters 5 and 6. They provide an memory-efficient way to compress the information of past examples.

2.1 Hoeffding Tree

Hoeffding Tree (HT) or Very Fast Decision Tree (VFDT) (Domingos and Hulten, 2000) is one of the classic data stream learning approaches. It is a decision tree algorithm that can learn from data stream incrementally and capable of making predictions at anytime, based on the assumption that the underlying concept of the data stream does not change over time. Due to its discriminative nature and ability in handling nominal input attributes, it has been widely used and studied as base learner in literature of dealing with concept drift and class imbalance in the last decade.

HT is based on the idea that a small subset of training examples may often be enough to choose the optimal splitting node (Domingos and Hulten, 2000). To determine how many examples in this small subset is enough, Domingos and Hulten (2000) introduced to use *Hoeffding bound*. Consider a random variable r with range R, where r is an attribute selection measure (For a probability, R is one, and for an information gain, R is $logc$, where c is the number of classes of the problem). For n observation of r, we compute their average $\bar{r}$. The Hoeffding bound states that, with the probability $(1 - \delta)$, the true mean of r is at least $(\bar{r} - \epsilon)$, where δ is a hyper-parameter and

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}} \tag{2.1}$$

HT uses Hoeffding bound to determine the minimum number of examples, n, needed to select a splitting attribute at a node. The advantage of using Hoeffding bound is that it is independent of the probability distribution, which is a desirable property for data stream learning because any underlying probability distribution of the data stream is unknown in advance.

In practice, an attribute selection measure $r = G(X_i)$ is employed to choose test attributes, where X_i is the i-th input attribute of the problem. This attribute selection measure could be information gain, or Gini index, etc. In this thesis, we adhered to MOA's default value to use information gain. The goal at each node is to ensure with high probability that the chosen attribute to split based a minimum number of n examples is the same as it would be chosen using infinite amount of examples, Assuming G is to be maximise (e.g information gain), let X_a and X_b be the attributes with highest and the second highest observed $\bar{G}$ after observing n examples at current node. If $\bar{G}(X_a) - \bar{G}(X_b) > \epsilon$, Hoeffding bound guarantees that, with a probability of $(1 - \delta)$, X_a is indeed the best attribute for splitting the current node.

In addition, HT allows to use different types of predictor in the leaves, which include Majority Class (MC), Naïve Bayes (NB) and Naïve Bayes Adaptive (NBAdaptive)[1]. In this thesis, we adopted Naïve Bayes (NB) as the predictor at the HT's leaves because preliminary experiments show that it has better stability in predictive performance than the other two types of predictors. Hence, the HT is often called as Hoeffding Tree with Naïve Bayes at Leaves (HTNB) in this thesis.

[1]Naïve Bayes Adaptive combine Majority Class and Naïve Bayes to make predictions based on the predictor with highest weight. The weight of the predictor is based on their prediction correctness

2.2 Diversity

Diversity has been investigated and adopted to build ensemble learners explicitly and implicitly in the literature (Kuncheva and Whitaker, 2003; Minku and Yao, 2012; Brzeziński and Stefanowski, 2014; Kolter and Maloof, 2003). Yet, there is no strict nor formal definition to what is diversity. Intuitively, it can be perceived as the level of disagreement among a set of learners. For instance, when two classifiers always make different predictions (e.g. one always predicts the data to be class 0 while the other one always predicts the data to be class 1), the diversity level among them is very high. On the other hand, when two learners always make the same prediction, there is no diversity among them.

There are different ways to build a diverse ensemble. For examples, we can use the Poisson distribution to train the base learners on different subsets of data from the same data stream (Oza, 2005). We can also vary the λ parameter of the Poisson distribution to control the diversity of the ensemble (Minku and Yao, 2012). Alternatively, we can also build a diverse ensemble by letting the base learners to train on different periods of the data stream or by taking base learners which trained on different concepts.

To measure the diversity between learners, Kuncheva and Whitaker (2003) surveyed a list of popular calculations in the literature. These calculations are mainly divided into two categories: pairwise diversity measures and non-pairwise diversity measures. As the names state, measures in the former category calculate the diversity between two models while measures in the latter category calculate on measuring the diversity of an ensemble as a whole. The followings are a few examples of pairwise and non-pairwise diversity measures.

A popular pairwise diversity measure is Yule's Q statistic (Yule, 1900; Kuncheva and Whitaker, 2003). Considering two models c_i and c_k, it can be calculated as Equation 2.2:

$$Q_{i,k} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}} \tag{2.2}$$

where N^{ab} is the number of examples where the classification by c_i is a and the classification by c_j is b, a value of 1 represents a correct classification while a value of 0 represents a misclassification. Q varies between 1 and -1. Models that tend to classify the same examples correctly will have positive values of Q, whereas models that tend to classify different examples incorrectly will have negative values of Q. The expected Q value for two statistically independent models is 0 (Yule, 1900; Kuncheva and Whitaker, 2003).

For an ensemble of L models, Equation 2.3 can be used to compute the averaged Q statistics over all pairs of models (Kuncheva and Whitaker, 2003).

$$Q_{average} = \frac{2}{L(L-1)} \sum_{i=1}^{L-1} \sum_{k=i+1}^{L} Q_{i,k} \tag{2.3}$$

The Double-Fault measure is another pairwise diversity measure used by Giacinto and Roli (2001). It was used to produce a pairwise diversity matrix for a set of models then select the models that are least related to the other models. The idea of double-fault measure is to calculate the proportion of examples that are misclassified by both models c_i and c_k. Equation 2.4 presents the calculation of double-fault measure (Giacinto and Roli, 2001; Kuncheva and Whitaker, 2003).

$$DF_{i,k} = \frac{N^{00}}{N^{11} + N^{10} + N^{01} + N00} \tag{2.4}$$

where the notation of $N^{a,b}$ follows the Yule's Q-Statistics. It is the number of examples where the classification by c_i is a and the classification by c_j is b, a value of 1 represents a correct classification while a value of 0 represents a misclassification.

As for non-pairwise diversity measure, one popular measure is the Entropy Measure E (Kuncheva and Whitaker, 2003). Its idea is to compute the diversity level among a set of L models based on a batch of examples $Z = \{z_j\}_{j=1}^{N}$, where N is the number of examples in the batch. Equation 2.5 presents the calculation of Entropy Measure E.

$$E = \frac{1}{N} \sum_{j=1}^{N} \frac{1}{(L - \lceil L/2 \rceil)} min\{l(z_j), L - l(z_j)\} \tag{2.5}$$

where $l(z_j)$ refers to the number of models that correctly predicts the j-th example in Z, where $0 \leq l(z_j) \leq L$. E varies between 0 and 1, where 0 refers to no disagreement while 1 refers to the highest possible diversity. Kuncheva and Whitaker (2003) explain, if $\lceil L/2 \rceil$ of the votes in Z are with the same value (0 or 1), while the other $L - \lceil L/2 \rceil$ of votes in Z are with the opposite value. These L models have the highest diversity, i.e., $E = 1$. On the other hand, if all L models give the same vote (all 0s or 1s), they have the no diverse or no disagreement, i.e., $E = 0$.

In this thesis, we follow Kuncheva and Whitaker (2003) to quantify diversity using a diversity measure. For example, if Entropy Measure E (Equation 2.5) is chosen to evaluate the diversity of a set of models, we say that this set of models is very diverse when $E = 1$. Similarly, we say that this set of models has no disagreement when $E = 0$.

2.3 Synthetic Minority Oversampling Technique (SMOTE)

Synthetic Minority Over-sampling Technique (SMOTE) (N. V. Chawla et al., 2002) is one of the pioneer data-level approaches in creating synthetic examples to deal with class imbalanced data sets (offline learning). N. V. Chawla et al. (2002) argued that simple random oversampling, which simply replicates existing minority class examples, likely leads to the

machine learning algorithm to overfitting. Thus, introducing synthetic examples to the minority class could be a better solution. In general, the idea of SMOTE is to generate new minority class examples along the line between two existing minority class examples in the feature space.

In practice, SMOTE first randomly chooses an existing minority class example from the data set, denoted as x_i. It then randomly chooses one of the k-nearest neighbours of x_i in the minority class, denoted as x_i'. After that, a difference vector between x_i and x_i' is calculated. To create a point along the line between x_i and x_i', each dimension of the difference vector is multiplied by a random number θ ($0 < \theta < 1$), then the resulting difference vector is added to x_i dimensionwisely. SMOTE performs this procedure until the target oversampling rate M is reached. This oversampling rate M and the k for k-nearest neighbours are the hyper-parameters of the algorithm.

The advantage of SMOTE is that it systematically creates synthetic minority class examples can help to explore the true decision areas of the minority class. Thus, the machine learning algorithm can have a better generalisation than using simple random oversampling. However, SMOTE ignored the fact that some existing minority class examples could locate deep inside the decision area of the majority class. The strategy of creating synthetic examples along the line between existing minority class examples could potentially create noisy examples within the majority class region, exacerbating the overlapping between two classes.

Several other approaches were then proposed to further improve the idea of SMOTE. For example, Borderline-SMOTE (Han, W.-Y. Wang, and B.-H. Mao, 2005) considers that examples close to the decision boundary are more difficult to learn. Therefore, Borderline-SMOTE only applies SMOTE to generate synthetic examples near the "border" (decision boundary). Gaussian-based SMOTE (Lee, J. Kim, and S. Kim, 2017) is a more recent approach which also creates synthetic examples along the line between x_i and x_i' but with

higher probability to be closer to x_i. Such probability is controlled by Gaussian distribution.

2.4 Data Stream Clustering

Data stream clustering is another research sub-field of data stream learning with unsupervised methods, i.e., analysing data stream regardless the label of the examples. Although Unsupervised data stream learning is not part of the main scope of this thesis but it plays a key role in the proposed approaches in Chapters 5 and 6. Specifically, the stream clustering methods proposed in this field of research provide an efficient way to compress and to keep tracking the locations of decision areas. Whilst data stream clustering is a large field of research which further branched into lots of sub-field, this section only introduces the key components that are used in the proposed approaches of Chapters 5 and 6. When explaining these components, two existing stream clustering approaches: CluStream (Aggarwal et al., 2003) and ClusTree (Kranen et al., 2011) are used as examples. These two approaches are considered in Chapters 5 and 6 as part of the hyper-parameters of the proposed approaches because they have a property that is very desirable to the idea of the proposed approaches – *micro-cluster*, which will be discussed in details in the latter paragraphs.

Data stream clustering is a research field which extends clustering analysis on finite data sets to potentially infinite long data streams. Whilst finite data set clustering has the availability of all data which allows the algorithm to process the data set several times, data stream clustering algorithms are often required to process the incoming examples upon their arrival. Meanwhile, data streams usually have a non-stationary nature, where the underlying data generating process may change over time (Moulton et al., 2019). This makes data stream clustering even more challenging. Silva et al. (2013) presented a list of requirements in developing data stream clustering methods:

- Provide timely results by performing fast and incremental processing of data objects.
- Rapidly adapt to changing dynamics of the data, which means algorithms should detect when new clusters may appear, or other disappears.
- Scale to the number of objects that are continuously arriving.
- Provide a model representation that is not only compact, but that also does not grow with the number of objects processed (notice that even a linear growth should not be tolerated).
- Rapidly detect the presence of outliers and act accordingly.
- Deal with different data types, for example, XML trees, DNA sequences, GPS temporal and spatial information.

With the aforementioned requirements, data stream clustering methods often have an online component and an offline component (Moulton et al., 2019).

The online component allows the stream clustering method to quickly and incrementally process new examples, usually without storing the examples in the memory explicitly. This part of the stream clustering algorithm often uses a data structure to compress and represent the unbounded examples compactly. In the literature, there are four types of data structure which are commonly used: 1) feature vector, 2) prototype array, 3) coreset trees, and 4) grids (Silva et al., 2013). Among these four data structures, this section focuses on *feature vector* as it can be considered as a compact and tiny spherical cluster. This property is very helpful for synthesising examples which is an idea heavily used in the proposed approaches of Chapters 5 and 6.

In the literature, *feature vector* (T. Zhang, Ramakrishnan, and Livny, 1997) is often names as *Clustering Feature vector (CF)* (Silva et al., 2013). It usually consists of three

components: N, the number of examples it represents, LS, the linear sum of the examples it represents, SS, the squared sum of the examples it represents (Silva et al., 2013; Moulton et al., 2019). LS and SS are n-dimensional arrays, where n is the number of dimensions (attributes) of the data stream (Silva et al., 2013; Moulton et al., 2019). Based on these information, we can compute the centre, radius and diameter of the micro-cluster using Equations 2.6, 2.7 and 2.8 respectively (Silva et al., 2013).

$$centre = \frac{LS}{N} \tag{2.6}$$

$$radius = \sqrt{(\frac{SS}{N} - (\frac{LS}{N})^2)} \tag{2.7}$$

$$diameter = \sqrt{(\frac{2N \times SS - 2 \times LS^2}{N(N-1)})} \tag{2.8}$$

Feature vector also has important incrementality and additivity properties (Silva et al., 2013; Moulton et al., 2019).

Incrementality: Given a CF and a new example x^j with j-dimensions, x^j can be easily inserted into CF (Silva et al., 2013; Moulton et al., 2019):

$$N \leftarrow N + 1$$

$$LS \leftarrow LS + x^j$$

$$SS \leftarrow SS + (x^j)^2$$

additivity: Given two disjoint CFs, CF_1 and CF_2, they can be easily merged to become CF_3 (Silva et al., 2013; Moulton et al., 2019):

$$N_3 \leftarrow N_1 + N_2$$

$$LS_3 \leftarrow LS_1 + LS_2$$

$$SS_3 \leftarrow SS_2 + SS_2$$

CluStream (Aggarwal et al., 2003) and ClusTree (Kranen et al., 2011) extended the idea of Clustering Feature vector to as *micro-cluster*. They augmented it with components to record the temporal information, such as the average timestamp of examples the micro-cluster represents . With these additional components, the algorithm knows which micro-clusters are relevant to the current underlying data distribution. This property is crucial, particularly, to the proposed approach in Chapter 6 which focuses on synthesising examples in the most recent decision region of minority class for oversampling.

Depending on the design of the stream clustering method, there are several ways to manage a set of micro-clusters or Clustering Feature vectors. For examples, CluStream (Aggarwal et al., 2003) uses a window-based method. It removes the oldest micro-cluster with the average timestamp smaller than a predefined threshold δ or merges two micro-clusters that are deemed to be close. ClusTree (Kranen et al., 2011) does not use a window but a hierarchical (tree) structure to store the micro-clusters. It has aggregating and splitting strategies to adjust the size of itself automatically, though it has a hyper-parameter to specify the maximum depth of the tree.

The offline component allows the stream clustering method to present the "marco" clustering result at the current time step, based on the information processed by the online component, i.e., the micro-clusters or Clustering Feature vectors. This is usually by taking the centre of the micro-clusters or Clustering Feature vectors (Equation 2.6) as representatives, then apply an offline clustering algorithm. For examples, K-Means clustering (Hartigan and Wong, 1979) and DBSCAN (Ester et al., 1996).

Also, it is important to note that not all the micro-clusters or Clustering Feature vectors in the system are used for this offline clustering phase but only those were created or updated within a predefined time horizon are used. For example, if the time horizon is set to 500, it usually means that the micro-clusters or Clustering Feature vectors which were created or updated within 500 time steps before the current time step are used for offline clustering.

Chapter Three

Related Works in Dealing with Concept Drift and Class Imbalance in Data Stream Classification

This chapter reviews the literature in the areas of concept drift adaptation, class imbalanced learning, and the joint challenge posed by the coexistence of concept drift and class imbalance in data stream learning. As it was discussed in Chapter 1, exploiting past knowledge is an intuitive strategy to deal with concept drift in class imbalanced data streams. For that, the memory strategy to decide which past model to preserve in and to recover from the memory becomes important. In this chapter, further comments are given to the memory strategy when reviewing existing approaches. Meanwhile, creating synthetic minority class examples could be another strategy to deal with concept drift when the data stream is class imbalanced, especially in situations where relevant past knowledge does not exist in the system's memory. Therefore, more emphasis is given to this kind of strategy when reviewing existing approaches for class imbalance learning.

This chapter is organised as follows. Section 3.1 reviews existing concept drift adap-

tation approaches for learning class balanced data streams. Section 3.2 reviews literature of class imbalanced learning in both offline and data stream learning. Although this thesis concentrates on online learning, reviewing offline class imbalanced learning approaches enables us to have a general idea of the potential issues when developing class imbalanced learning methods. Section 3.3 reviews existing approaches for learning drifting class imbalanced data streams. Also, Section 3.4 reviews the current evaluation methods in both class balanced and class imbalanced data stream learning. Lastly, this chapter is summarised in Section 3.5

3.1 Approaches for Learning Drifting Class Balanced Data Streams

The literature of concept drift adaptation for class balanced data stream learning is well-established in the past years. They mainly can be divided into two groups explicit and implicit approaches (Žliobaitė, 2010; Hoens, Polikar, and N. V. Chawla, 2011; S. Wang, Minku, and Yao, 2018; Gama, Žliobaitė, et al., 2014; Ditzler, Roveri, et al., 2015).

Explicit approaches are referred to as approaches that employ a concept drift detector to estimate when does concept drift happen, usually by monitoring the predictive performance of the base learner / main ensemble (Ditzler, Roveri, et al., 2015; Žliobaitė, 2010). As they "actively" detect concept drift, they are also frequently referred to as active approaches in the literature (Ditzler, Roveri, et al., 2015; Žliobaitė, 2010; Gama, Žliobaitė, et al., 2014; S. Wang, Minku, and Yao, 2018). Explicit approaches are typically more effective in dealing with abrupt drifts (Ditzler, Roveri, et al., 2015; S. Wang, Minku, and Yao, 2018), even though their effectiveness is usually determined by the hyper-parameters that control the trade-off between the drift detection delay and the amount of false-positive drift detection. Some explicit approaches reset their base learner / main ensemble upon concept drift detec-

tion, which means the knowledge they learnt previously is forgotten (Gama, Medas, et al., 2004; Baena-Garc et al., 2005; Nishida and Yamauchi, 2007). Modern approaches usually were proposed as a whole frameworks which include strategies to detect concept drift and strategies to respond it. For example, some approaches exploit past knowledge to help in learning the new concept after the drift and are robust to false-positive drift detection (Minku and Yao, 2012; Gonçalves Jr and Barros, 2013; Gama and Kosina, 2009; Abad, J. B. Gomes, and Ruiz, 2016).

In contrast, implicit approaches do not employ a concept drift detector. They consider that concept drift can happen at any time with any speed and severity (Ditzler, Roveri, et al., 2015; Žliobaitė, 2010). Thus, they continuously update their parameters, usually based on their predictive performance. As they do not actively attempt to detect concept drifts, they are also frequently referred to as passive approaches in the literature (Ditzler, Roveri, et al., 2015; Žliobaitė, 2010; Gama, Žliobaitė, et al., 2014; S. Wang, Minku, and Yao, 2018).

Implicit approaches can be further categorised as single learner and ensemble learner approaches. The former category continuously updates the single learner to reflect the current underlying concept. Existing single learner approaches include Concept-adaptive Very-Fast Decision Tree (CVFDT) (Hulten, Spencer, and Domingos, 2001), Online Information Network (OLIN) (Cohen et al., 2008), etc. The latter category usually associates weights to the ensemble members based on their predictive performance and they allow to add and remove base learners in the ensemble (Polikar et al., 2001; Kolter and Maloof, 2003; Brzeziński and Stefanowski, 2014). These actions can be considered as memory strategies where the ensemble is the memory. As implicit approaches do not explicitly address concept drift, they are likely to have delay in reacting to concept drift. However, they are more effective in dealing with gradual drifts as they adjust the weights of the ensemble members continuously (Ditzler, Roveri, et al., 2015; S. Wang, Minku, and Yao, 2018).

As it was discussed in the introduction of this chapter, memory strategies are important as they could help the data stream learning approach in adapting to different concepts. However, it is difficult to find any literature that investigates the impact of using different memory strategies to the accuracy of a data stream learning approach. Such a study is present in Chapter 4 of this thesis and reveals that diversity can be a solution to maximise the chances of maintaining various past concepts in the memory, preparing the approach for different concepts to (re)occur in the future.

In the next sections, several existing approaches are explained, giving special attention to those that are closely related to this work. The approaches that are closely related to this work are also considered in the experiments in later chapters. Section 3.1.1 reviews explicit approaches while Section 3.1.2 reviews implicit approaches.

3.1.1 Explicit Concept Drift Adaptation Approaches for Class Balanced Data Stream Learning

Explicit concept drift adaptation approaches are intuitive because they detect the concept drift and then adapt to it. The simplest way to build an explicit approach is to associate a concept drift detector to a base learner. Upon concept drift detection, the current base learner is replaced by a new one. Note that the base learner can be a single learner or an ensemble of data stream learning approaches that was not designed to adapt to concept drift. The concept drift detector can be any from the literature. In Section 3.1.1.1, several concept drift detectors are reviewed and they are considered in the experiments of this thesis.

However, the simple strategy of replacing the old base learner by a new one upon drift detection may not be always ideal. One major disadvantage is that the predictive performance usually decreases drastically when compared against approaches that do not

address concept drift, upon concept drift detection, because a new base learner has to be learnt from scratch. Besides, a concept drift could lead to a concept that is difficult to learn but similar to a previous concept. In this case, this kind of approach would not maintain learnt previously knowledge. If the data stream learning approach has relevant past knowledge in the memory, it has the advantage to exploit it and swiftly recover the predictive performance from concept drift. Also, the drift detection could sometimes be false-positive. Thus, the strategy of always resetting the base learner could result in poor predictive performance and being sensitive to subsequent false-positive drift detection.

In Section 3.1.1.2 and 3.1.1.3, two explicit concept drift adaptation approaches (frameworks) are discussed in detail. They are Diversity for Dealing with Drifts (DDD) (Minku and Yao, 2012) and Recurrent Concept Drift (RCD) (Gonçalves Jr and Barros, 2013). These approaches proposed strategies to exploit past knowledge in adapting to concept drift. Such strategies have the potential to prevent a drastic drop in predictive performance upon drift detection.

3.1.1.1 Concept Drift Detection Methods for Class Balanced Data Streams

Lots of concept drift detectors were proposed in the context of learning drifting class balanced data stream. The following are some examples of methods that were well-studied in the literature: Page-Hinckley Test (PHT) change detector (Page, 1954), Drift Detection Method (DDM) (Gama, Medas, et al., 2004), Early Drift Detection Method (EDDM) (Baena-Garc et al., 2005), and Adaptive Windowing (ADWIN) (Bifet and Gavaldà, 2007).

The Page-Hinckley Test (PHT) (Page, 1954) is one of the pioneer methods in the literature. It is a sequential analysis technique that can be used to detect concept drift. The general idea is to test if the cumulative difference between the classification correctness and their average upon the current time step T has a large difference with the minimum value

of such cumulative difference in previous time steps. Formally speaking, such cumulative difference at the current time step T is $m_T = \sum_{t=1}^{T}(x_t - \bar{x_T} - \delta)$, where $\bar{x_T} = \frac{1}{T}\sum_{t=1}^{T} x_t$ is the mean value of the classification correctness up until the current time step T and δ is a hyper-parameter controlling the tolerance of change magnitude. The minimum value of m_T can easily be obtained by $m_T^{min} = min(\{m_t\}_{t=1}^{T})$. PHT confirms a concept drift when $m_T - m_T^{min} > \lambda$ and the variables concerning the previous cumulative differences are reinitialised. λ is a hyper-parameter to control the trade-off between drift detection delay and the number of false-positive drift detections.

PHT only requires a few variables to store the statistics of the cumulative differences. It does not require to store any example for drift detection. Therefore, it is an online approach. Experiments by Gonçalves Jr et al. (2014) on eight artificial data streams showed that PHT was one of the fastest approaches in terms of the computational time taken to evaluate if concept drift occurs at a given time step. However, it was also one of the approaches that have a high miss detection rate (Gonçalves Jr et al., 2014). As previously described, PHT has two hyper-parameters to control the trade-off between drift detection delay and the number of false-positive drift detections. Possibly, these hyper-parameters can be finely tuned to reduce the miss detection rate. However, finding the best hyper-parameters is difficult for data stream learning approaches because it is impossible to know the characteristics of the concept drift in advance. Nevertheless, we can see that PHT is sensitive to the hyper-parameter settings.

Drift Detection Method (DDM) (Gama, Medas, et al., 2004) considers that the error of the base learner to a set of examples is a random variable from Bernoulli trials. When the class distribution of the examples is stationary, the Binomial distribution gives the general form of the probability of such random variable in a sample of n examples. With a sufficiently large number of examples, the Binomial distribution can be closely approximated by the normal distribution. For time step i in the sequence of the examples, the error rate of the

base learner is the probability of observing an incorrect prediction, p_i, with the standard deviation $s_i = sqrt(p_i(1 - p_i)/i)$. DDM detects concept drift based on the idea that the error rate of the base learner (p_i) should decrease as the number of examples (i) increases when the class distribution of the examples is stationary. When the error of the base learner increases significantly compared to the expected value based on a normal distribution, it suggests that the class distribution of the examples has changed.

To achieve this idea, DDM records the minimum error rate (p_{min}) and the corresponding standard deviation (s_{min}) of the base learner obtained until the current time step i. These values are updated when $p_i + s_i$ is lower than $p_{min} + s_{min}$. Warning alarm is issued when $p_i + s_i \leq p_{min} + 2 * s_{min}$ which corresponds to 95% confidence interval of the normal distribution. Since the warning alarm is given, the incoming examples are stored, anticipating a potential concept drift. DDM confirms a concept drift occurs when $p_i + s_i \leq p_{min} + 3 * s_{min}$, which corresponds to 99% confidence interval of the normal distribution. The variables of p_{min} and s_{min} are then reset, while a new base learner is created to replace the old one and trains on the examples stored since warning level.

Experiments with eight artificial data streams showed that, DDM usually recovered the predictive performance much better than the base learner that does not use DDM. Experiments with a real-world data stream of electricity market showed that DDM presents similar behaviours as in artificial data streams. Such results were obtained by repeating the experiments with three different base learners: a perceptron, a neural network and a decision tree. Thus, the effectiveness of DDM was independent of the base learner choice. The disadvantage of DDM is that it is likely to take a long time to confirm a concept drift if the drift is very slow. Thus, depending on the design of the implementation, the number of examples stored in the memory since the warning level could be large, potentially overflowing the system's memory. Possibly, it can create a new base learner to learn alongside with the existing one since the warning level but it still does not resolve the issue that it could not

detect very slow gradual drift in a timely manner.

To overcome the disadvantages of DDM, Baena-Garc et al. (2005) proposed an improved method called Early Drift Detection Method (EDDM). They considered that, while learning a stable concept, not only the number of classification errors decreases but also the distance between two classification errors should increase. For time step i in the sequence of the examples, EDDM calculates the average distance between two classification errors (p'_i) and the corresponding standard deviation (s'_i). It records p'_i and s'_i when $p'_i + 2 * s'_i$ reaches its maximum value, which are denoted as p'_{max} and s'_{max} respectively. A warning alarm is issued when $(p'_i + 2 * s'_i)/(p'_{max} + 2 * s'max) < \alpha$. From then, the incoming examples are stored, anticipating a potential concept drift. EDDM confirms a concept drift when $(p'_i + 2 * s'_i)/(p'_{max} + 2 * s'max) < \beta$, where $\alpha > \beta$ are the hyper-parameters to control the thresholds of warning and drift level respectively. Also, variables of p'_{min} and s'_{min} are reset upon concept drift detection (confirm level), while a new base learner is created to replace the old one and trains on the examples stored since warning level.

Experiments with an artificial data streams presenting slow gradual drift showed that EDDM reacts faster than DDM. Meanwhile, experiments with four other artificial data streams showed that EDDM reacts similarly to DDM in both abrupt and gradual drift. However, when noise is present, EDDM is more sensitive than DDM, especially when the base learner is not prepared to learn with noise. The experiments done in this thesis also show that EDDM tends to give a lot of false-positive drift detections, being consistent with the experiment results obtained by Baena-Garc et al. (2005).

Apart from EDDM, several recent approaches were also proposed based on the inspiration of DDM. Two examples are Hoeffding's inequality Drift Detection Method (HDDM) (Frías-Blanco et al., 2015) and Exponentially Weighted Moving Average Charts for Concept Drift Detection (ECDD) (Ross et al., 2012). HDDM uses Hoeffding's bounds to identify

concept drift, instead of using the confident interval of normal distribution in DDM. ECDD monitors the predictive performance of the base learner using an EWMA chart. Particularly, it modifies the EWMA chart from using a static mean to a dynamic mean which is estimated on the fly. ECDD reports a drift detection when the current EWMA estimator is larger than the dynamic mean with a proportion of the corresponding standard deviation, where such proportion is controlled by a hyper-parameter.

Adaptive Windowing (ADWIN) (Bifet and Gavaldà, 2007) maintains a sliding window W with the most recent classification correctness of the base learner. ADWIN detects concept drift by checking if there exists two (sub)windows of W exhibiting distinct averages. If there is, ADWIN concludes that the expected values of those windows are different, thus, confirming the occurrence of concept drift. Meanwhile, the older (sub)window is dropped because it is deemed to belong to the old concept. In other cases where the (sub)windows of W do not exhibit distinct averages, ADWIN dynamically grows the sliding window W.

The main advantage of ADWIN is that it could potentially find the precise location of the concept drift point, despite the potential delay in detecting the drift if the number of processed examples is less than the length of W. In addition, ADWIN has rigorous guarantees of performance bounds on rates of false-positive and false-negative drift detection. Empirical results obtained from the experiments performed by Gonçalves Jr et al. (2014) also showed that ADWIN is one of the drift detectors that achieved the lowest false positive drift detection rates, being similar to STEPD (Nishida and Yamauchi, 2007) and DDM (Gama, Medas, et al., 2004) but better than PHT (Page, 1954), ECDD (Ross et al., 2012), EDDM (Baena-Garc et al., 2005), PL (Bach and Maloof, 2008) and DoF (Sobhani and Beigy, 2011).

The disadvantages of ADWIN include computationally expansive search of distinct averages exhibited by two (sub)windows of W, and the contents in the window W are explicitly stored which grows linearly if there is no concept drift presented in the data

stream. To overcome these issues, Bifet and Gavaldà (2007) uses buckets to group the contents within W, thus reducing the computational time in finding the best cut point of W and used a variation of exponential histogram (Datar et al., 2002) to approximate the number of classification errors in W, thus the classification correctness do not need to store explicitly. Besides, experiments performed by Gonçalves Jr et al. (2014) showed that ADWIN could struggle in detecting concept drift when the concepts before and after the drift are not mutually exclusive.

In addition to the aforementioned advantages and disadvantages, the concept drift detectors analysed in this section cannot be applied to detect concept drift in class imbalanced data streams. First of all, PHT and DDM detect drift based on the average error rate of the base learner. This metric is not adequate in evaluating base learners when the data stream is class imbalanced because it biases towards the predictive performance in the majority class. EDDM detects drifts based on the distance between two consecutive classification errors. When the data stream is class imbalanced, the errors made by the base learners are likely the errors in the majority class. If a concept drift only happens in the minority class, EDDM likely fails to identify it. Similar reasons apply to ADWIN in detecting concept drift in class imbalanced data streams. Therefore, other work have been proposed to detect concept drift in class imbalanced data streams, which are discussed in Section 3.3.1.1.

3.1.1.2 Diversity for Dealing with Drifts (DDD)

Diversity for Dealing with Drifts (DDD) (Minku and Yao, 2012) is an explicit ensemble approach which applies diversity to deal with concept drifts. Its general idea is to use the knowledge learnt with high diversity from the concept before the drift to aid in learning the concept after the drift. This is because high diversity ensembles tend to be able to preserve some potentially useful knowledge of the past concept while being more easily adaptable to

the new concept than ensembles trained with low diversity.

DDD adopts Online Bagging (OB) (Oza, 2005) as the base ensemble. To achieve training the OBs with different diversity levels, it varies the λ of the Poisson distribution, which decides the number of times that an ensemble member is trained on the current example. This method was first studied by Minku, White, and Yao (2010). During stable concept periods, DDD is in the state of "before drift" and it maintains two ensembles. One learns from the data stream with low diversity while the other one learns with high diversity. A concept drift detector is employed to monitor the error rate of the low diversity ensemble. This drift detector can be any from the literature, where some examples are given in Section 3.1.1.1.

When a concept drift is detected, DDD changes its state to "after drift" and creates new low and new high diversity ensembles. From drift detection onwards, four ensembles learn from the data stream together and the drift detector is changed to monitor the new low diversity ensemble. The old high diversity ensemble starts to learn with low diversity, attempting to converge from the old concept to the new concept. During the state of "after drift", DDD makes predictions by weighted majority of all the ensembles where the weights are proportional to the ensembles' prequential accuracy since last drift detection. Except the weight of the old low diversity ensemble is multiplied by a constant W, which is a hyper-parameter to control the trade-off between the robustness to false-positive drift detection and the accuracy to the current concept.

When either the new low or the old high diversity ensemble performs better than other ensembles, DDD reverts back to the state of "before drift" and the ensemble which performs better will be the low diversity ensemble afterwards. Predictions are then made by this low diversity ensemble only. If the new low diversity remains in the same role, the old low and old high diversity ensembles are discarded. Otherwise, the new low and the old

diversity ensembles are discarded, which means that the old high diversity ensemble becomes the low diversity ensemble. Besides, DDD is robust to false-positive drift detection because it has a mechanism to revert back to use the old low and old high ensembles.

Experiments with six artificial data streams showed that DDD successfully exploits the knowledge from the old concept before the drift to aid in learning the new concept after the drift. It also showed that an ensemble trained on the old concept with high diversity can converge to the new concept when training with low diversity. Such ensemble can outperform new ensembles created from scratch upon drift detection, especially when the drift has low severity and high speed. Also, DDD is robust to false-positive drift detections.

However, the memory strategy of DDD is a weakness. DDD only retains the past knowledge that belongs to the concept prior to the most recent drift. If a concept recurs after many concept drifts, DDD loses the advantage of exploiting the corresponding past knowledge. The same issue happens when a concept drift is highly severe but the concept after the drift is similar to a previous one. Besides, DDD could not be applied to class imbalanced data streams because it weights and selects the main ensemble based on prequential accuracy, which is likely biased towards the majority class.

3.1.1.3 Recurrent Concept Drift (RCD)

Recurrent Concept Drift (RCD) (Gonçalves Jr and Barros, 2013) is a concept drift adaptation framework that exploits relevant past knowledge to handle recurrent drift. The idea of RCD to identify recurrent drift is simple. It pairwisely compares the most recent window of examples with all the stored windows of examples in the memory, using a multivariate non-parametric statistical test. Recurrent drift is identified when the statistical test shows that there is no significant difference between the most recent window and a stored window of examples.

RCD is a framework employing a drift detector to handle concept drift explicitly. The base learner of RCD can be a single learner or an ensemble of learners while the concept drift detector can be any from the literature. Section 3.1.1.1 presents a few examples. RCD associates a sliding window (sw_a) to the base learner (c_a). The sliding window sw_a stores the incoming examples which are used to train c_a, thus representing the knowledge learnt by c_a. The drift detector continuously monitors the predictive performance of c_a. When the error rate of c_a increases, the drift detector will give the warning alarm, indicating that a potential concept drift is going to occur. Meanwhile, a new base learner (c_n) is created with a new sliding window (sw_n). Since the warning alarm is given, c_n trains on the incoming examples alongside with c_a but the incoming examples are only stored in sw_n. Note that the sw_a is retained because the concept drift is not confirmed and such warning alarm could be a false-positive (i.e. the error rate of c_a decreases afterwards). In this case, c_n and sw_n are discarded and sw_a is used to store the incoming examples.

When the error rate of c_a further increases, the drift detector confirms that a concept drift is likely occurred. RCD then performs a statistical test to compare sw_n with all stored windows in the memory. If the statistical test shows that there is no significant difference between sw_n and one of the stored windows, RCD considers that an old concept is recurring. Thus, the corresponding past base learner and sliding window are restored to be the new c_a and sw_a, respectively. The model c_n and sliding window sw_n are discarded. If the statistical test shows that sw_n is significantly different from all stored windows in the memory, this means that the concept drift leads to a new concept. In this case, the current c_a and sw_a are stored into the memory by the policy of First in First Out (FIFO) while the c_n and sw_n are considered to be the new c_a and sw_a.

The main advantage of RCD is using a statistical test to confirm the recurrence of a past concept. This strategy can prevent the past base learner from learning a concept that is not the same as it learnt before. Thus, RCD can identify recurrent concept with high

confidence. However, this strategy can be considered as conservative because it recovers past base learners to deal with concept drift only when the new concept after the drift matches a previous one. In the situation where the concept drift leads to a concept that is similar but not exactly the same as a previous one, RCD loses the advantage of exploiting similar past knowledge in dealing with drifts. Meanwhile, RCD adopts a FIFO strategy to manage a memory of past base learners. Depending on the memory size, this strategy may be disadvantages when a concept recurs after lots of concept drifts. Lastly, RCD cannot be applied to learn drifting class imbalanced data streams directly. This is because the statistical test relies on the sliding windows of examples which are likely class imbalanced when the class distribution of the data stream is skewed.

3.1.2 Implicit Concept Drift Adaptation Approaches for Class Balanced Data Stream Learning

Implicit concept drift adaptation is another category of approaches in the literature of concept drift adaptation. Unlike explicit approaches, implicit approaches do not employ any concept drift detector but continuously evolve themselves to better reflect the current underlying concept. In this section, Dynamic Weighted Majority (DWM) (Kolter and Maloof, 2003) and Online Accuracy Update Ensemble (OAUE) (Brzeziński and Stefanowski, 2014) were chosen to review because they present the major characteristics that most ensemble learners usually have to implicitly deal with concept drift and they have been studied by many works in literature. In particular, we focus on how they manage the ensemble members in dealing with concept drift, which can be considered as memory management strategy. Note that single learners were not chosen to reviewed here because they do not have memory of past learners, thus no memory strategy is needed.

3.1.2.1 Dynamic Weighted Majority (DWM)

Dynamic Weighted Majority (DWM) was proposed by Kolter and Maloof (2003). It extends the idea of Weighted Majority algorithm (Littlestone and Warmuth, 1994) to learn drifting data streams. It maintains an ensemble of learners, where the size of the ensemble is flexible. Each ensemble member starts with a weight of one and this weight is reduced by constant β ($0 \leq \beta < 1$) when it makes a mistake, where β is a hyper-parameter. Despite being one of the early approaches, it tends to achieve very competitive results, and thus continue to be used as a baseline in data stream learning studies (Ditzler, Roveri, et al., 2015; Gama, Žliobaitė, et al., 2014).

DWM makes predictions by a weighted majority over the whole ensemble. If such global prediction makes a mistake, DWM first normalises the weights of the existing ensemble members and then it expands the ensemble by introducing a new base learner. The normalisation of the existing weights is to prevent the new base learner from dominating the the weighted majority votes. Besides, DWM also prunes ensemble members whose weights are less than a threshold θ, which is a hyper-parameter. Note that, DWM only allows to perform the aforementioned actions at every p time steps, where p is also a hyper-parameter. This is to prevent DWM from frequently introducing and pruning ensemble members because of noisy examples.

The training mechanism of DWM allows it to adapt to any potential concept drift by adding more new base learners to learn the new concept, while allowing existing ensemble members to take time in converging to the new concept (Though, this depends on the choice of hyper-parameters). Thus, such strategy can also be considered as exploiting past knowledge to aid the learning of new concepts. However, the strategy of pruning ensemble members which perform continually poorly on the current concept may not always be ideal. For example, when the data stream severely drifts to a new concept and then drifts back to

the previous one after a long while, DWM is likely to struggle in handling such recurrent drift. This happens because the relevant knowledge may have been pruned from the ensemble, or may be represented by models that have now been trained on new concepts and thus forgotten knowledge about the past concept. Besides, DWM also likely to have delay in adapting to abrupt drifts, depending on the chosen value of hyper-parameter p.

In addition, DWM cannot be applied to learn class imbalanced data stream because its weighting mechanism is based on the error on the most recent examples. Thus, it is likely biased towards the majority class when the data stream is class imbalanced. With this in mind, Lu, Cheung, and Tang (2017) proposed an improved version of DWM, called Dynamic Weighted Majority for Imbalanced Learning (DWMIL), which is discussed in Section 3.3.2.2.

3.1.2.2 Online Accuracy Update Ensemble (OAUE)

Brzeziński and Stefanowski (2014) proposed the Online Accuracy Update Ensemble (OAUE) by converting the batch-based Accuracy Update Ensemble (AUE) (Brzeziński and Stefanowski, 2011) to be an online ensemble learner. OAUE maintains a weighted majority vote ensemble. The weight of each ensemble member is based of their estimated Mean Squared prediction Error (MSE) over the last d time steps. Whenever a new example is received, it is used to weight and to incrementally train each ensemble member. At every d time steps, OAUE creates a new base learner and trains it alongside with the main ensemble. This new base learner replaces the least performing ensemble member after the next d time steps.

OAUE successfully brought the advantages of a batch-based ensemble learner to an online ensemble learner. In particular, its weighting function is memory efficient as it does not require to store a set of past examples. Experiments with eleven artificial data streams and five real-world data streams showed that OAUE achieved better predictive performance

than four other existing ensemble approaches including Online Bagging (OB) and Dynamic Weighted Majority (DWM) with the least time and memory consumption. However, OAUE also faces a similar issue as DWM, which is Likely to have delay in adapting to abrupt drifts, depending on the hyper-parameter value of d. Also, OAUE prunes ensemble members that perform poorly to the current concept. Thus, it also loses the advantage of exploiting past knowledge to handle the recurrence of a past concept that is distinct from the current one.

Besides, OAUE also cannot be applied to learn class imbalanced data stream directly because its weighting function is based on the MSE of the ensemble members. However, it could potentially be modified to give a higher penalty when an ensemble member misclassifies a minority class example than misclassifying majority class example.

3.2 Approaches for Learning Stable Concept Class Imbalanced Data Streams

Class imbalanced learning is another important topic in data stream learning. It was first investigated in offline learning. Generally speaking, class imbalance tends to bias machine learning approaches towards predicting the majority class. To address this challenge, class imbalance handling techniques can mainly be divided into three major categories: algorithm-level and data-level strategy and ensemble algorithm.

Algorithm-level strategies usually modify the cost function of the learning algorithm to put more emphasis in learning the minority class than the majority class. Thus, they are specific to the learning algorithm. For example, GH-VFDT, HD-VFDT, RLSACP (Ghazikhani, Monsefi, and yazdi, 2013), ONN (Ghazikhani, Monsefi, and yazdi, 2014) are in this category. Section 3.2.1 discusses Gaussian Gaussian Hellinger Very Fast Decision Tree (GH-VFDT)

and Hellinger Distance Very Fast Decision Tree (HD-VFDT) (Lyon et al., 2014) in detail, as representative examples of this category.

Data-level strategies balance the number of training examples from both classes, which can be further divided into oversampling and undersampling strategies (Japkowicz and Stephen, 2002; He and Garcia, 2009; S. Wang, Minku, and Yao, 2018). They are applicable to any learning algorithm but they are likely to increase the training time (oversampling) or to lose potentially useful information about the majority class (undersampling). Existing techniques to deal with class imbalance in offline learning include ADASYN (He, Bai, et al., 2008), SMOTE (N. V. Chawla et al., 2002), Gaussian-based SMOTE (Lee, J. Kim, and S. Kim, 2017), Tomek links (Tomek, 1976) and One-sided selection (Kubat and Matwin, 1997). While some recent work attempted to bring the successful strategies in offline learning to data stream learning, they usually have some limitations (B. Wang and Pineau, 2016; W. Mao, J. Wang, and L. Wang, 2015). For example, some of them require additional minority class examples for initialisation before training starts (W. Mao, J. Wang, and L. Wang, 2015) which is impractical because the minority class examples are usually expansive to collect. Some other work require to store all the minority class examples that have been seen so far into the system memory (B. Wang and Pineau, 2016), which is infeasible in data stream learning.

Ensemble algorithms combine multiple base learners and train the base learners with different subsets of training examples, thus reduce the risk of overfitting (Krawczyk et al., 2017; Hansen and Salamon, 1990). To handle class imbalance, existing work usually focuses on how to make different class balanced training subsets to train the base learner and / or emphasise the cost in misclassify minority class examples. The main advantage of ensemble algorithms is that they usually do not need to modify the base learner to be cost-sensitive nor create any synthetic examples, but still able to address the challenge of class imbalance. However, they may suffer when the number of minority class examples in the data set

(stream) is extremely low. In that case, a synthetic data generation strategy, such as SMOTE (N. V. Chawla et al., 2002), could be incorporated to resolve the issue (N. Chawla et al., 2003; S. Wang and Yao, 2009). Existing ensemble algorithms for offline class imbalanced learning include, SMOTEBoost (N. Chawla et al., 2003), Bagging Ensemble Variation (BEV) (Li, 2007), and AdaBoost.NC (S. Wang, H. Chen, and Yao, 2010) etc. Some existing work brought the such idea to class imbalanced data stream learning, but they usually require to store all the minority class examples or assume that relevant past minority class examples exist (Gao et al., 2007; S. Chen and He, 2009; S. Chen and He, 2011).

In this thesis, we are particularly interested in a category of data-level strategies, which create synthetic minority class examples for oversampling. This kind of strategy could potentially be helpful in situations where exploiting past knowledge to deal with drift is impossible. Therefore, we focus on discussing approaches from this category of data-level strategies.

This section is further organised as follows. Section 3.2.1 discusses Gaussian Hellinger Very Fast Decision Tree (GH-VFDT) and Hellinger Distance Very Fast Decision Tree (HD-VFDT) (Lyon et al., 2014) in detail, as examples of algorithm-level strategies for class imbalanced data stream learning. Section 3.2.2 discusses OnlineUnderOverBagging (B. Wang and Pineau, 2016) as an example of existing approaches that adopt the simplest version data-level strategies for class imbalanced data stream learning. Section 3.2.2 and 3.2.3 discuss OnlineSMOTEBagging (B. Wang and Pineau, 2016) and Online Sequential Extreme Learning Machine (OS-ELM) with Improved SMOTE (PCI-OSELM) (W. Mao, J. Wang, and L. Wang, 2015) respectively, as two examples of existing work attempting to bring the idea of synthetic minority class oversampling to data stream learning. As OnlineSMOTE-Bagging and PCI-OSELM both adopt a notable approach in offline learning called Synthetic Minority Over-sampling Technique (SMOTE) (N. V. Chawla et al., 2002), Section 2.3 first introduce the idea of SMOTE before the discussion in Sections 3.2.2 and 3.2.3.

3.2.1 Gaussian Hellinger Very Fast Decision Tree (GH-VFDT) and Hellinger Distance Very Fast Decision Tree (HD-VFDT)

Gaussian Hellinger Very Fast Decision Tree (GH-VFDT) and Hellinger Distance Very Fast Decision Tree (HD-VFDT) were proposed by Lyon et al. (2014) to handle class imbalanced data streams. The general idea is to use Hellinger distance as split criterion in the Very Fast Decision Tree (VFDT) (Domingos and Hulten, 2000). It splits on the feature that best discriminates the examples seen so far, by comparing the class distributions of both classes using Hellinger distance. Therefore, this method is class imbalance insensitive as more examples in one class will only bring the sample distribution closer to the real one. In contrast, the split criterion used in the original VFDT was based on information gain by Hoeffding bounds, which splits on the feature that describes the largest possible number of examples seen so far. Thus, it is likely to be affected when the data stream is class imbalanced.

The aforementioned idea was first proposed by Cieslak and N. V. Chawla (2008), then an offline decision tree for learning class imbalanced data set called Hellinger Distance Decision Tree (HDDT) was proposed (Cieslak and N. V. Chawla, 2008). Lyon et al. (2014) further improved it by assuming that the distributions of the two classes in a single continuous feature follows the Gaussian distribution. Thus, the Hellinger distance can be computed using the mean and the standard deviation of two distributions. This improves the memory consumption over the original method by Cieslak and N. V. Chawla (2008), which discretises continuous features. Thus, GH-VFDT was proposed. In order to show advantages of the improved method, Cieslak and N. V. Chawla (2008) applied the original method to VFDT, thus created HD-VFDT and compared it with GH-VFDT in data stream learning environment.

Experiments with eight data sets showed that GH-VFDT effectively improves minor-

ity class recall, being similar to HD-VFDT's performance but returning fewer false positive prediction (positive refers to the minority class). The major drawback of GH-VFDT is the parametric assumption to class distributions, which real-world applications may not follow. Another drawback is that the challenge of concept drift was not considered in this work.

3.2.2 OnlineUnderOverBagging and OnlineSMOTEBagging

B. Wang and Pineau (2016) proposed OnlineUnderOverBagging and OnlineSMOTEBagging by applying the idea of UnderOverBagging and SMOTEBagging (S. Wang and Yao, 2009) to Online Bagging (OB) (Oza, 2005). S. Wang and Yao (2009) first investigated the idea of UnderOverBagging by combining Undersampling-based Bagging and Oversampling-based Bagging, which aims to further boost the diversity within a offline ensemble learning to deal with multi-class imbalanced data sets. Though, multi-class classification is not within the scope of this thesis.

The strategy of OnlineUnderOverBagging is to undersample the majority class and oversample the minority class at the same time, by varying the sampling rate over the bagging iteration. OnlineSMOTEBagging has the same training mechanism as OnlineUnderOverBagging but changed to use SMOTE instead of simple oversampling. As it needs to apply SMOTE to all the minority class examples seen until the current time step, it requires to store them into system's memory, which is a big disadvantage because the data steam is potentially infinite but the memory is always limited. Another disadvantage is that they both adopt a constant resampling rate, following their offline counterpart. Thus, they cannot adapt to concept drift of changing class imbalance ratio, i.e., drifts affecting $P(Y)$.

3.2.3 Online Sequential Extreme Learning Machine (OS-ELM) with Improved SMOTE

Online Sequential Extreme Learning Machine (OS-ELM) with Improved SMOTE (PCI-OSELM) was proposed by W. Mao, J. Wang, and L. Wang (2015), which combines an Online Sequential Extreme Learning Machine (OS-ELM) (Liang et al., 2006) with SMOTE to learn class imbalanced data streams. It has additional strategies to select the synthetic examples produced by SMOTE, which aims to reduce the randomness of SMOTE in online learning. Such strategies is also used to select majority class examples to train the OS-ELM.

In practice, principle curve (Hastie and Stuetzle, 1989) is adopted to achieve this purpose. This may be the reason why W. Mao, J. Wang, and L. Wang (2015) abbreviated this approach to as PCI-OSELM, but they did not provide the full name in their article. Principle curve (Hastie and Stuetzle, 1989) is an extension to principle component analysis (PCA) (Pearson, n.d.), which is a dimensionality reduction technique. It finds a continuous one dimensional manifold to approximate the real examples. PCI-OSELM calculates the mean projection distance of all real examples to the principle curve by class. Such mean value is the threshold (δ) which decide which (synthetic) examples to train the Online Sequential Extreme Learning Machine (OS-ELM).

Regarding the training procedure, PCI-OSELM first uses the aforementioned procedure to produce a class balanced training set, thus uses it to train the OS-ELM in an offline setting. Such offline initialisation is required by OS-ELM. Once the OS-ELM is initialised, PCI-OSELM then starts learning the class imbalanced data stream in an online way.

Whenever a new majority class example arrives, PCI-OSELM calculates its distance to the current principle curve in the majority class. If the distance exceeds the current threshold (δ_{maj}), this majority class example is discarded. Otherwise, it is used to update

OS-ELM and the principle curve in the majority class, thus update the threshold (δ_{maj}). Likewise, if the most recent example belongs to the minority class, PCI-OSELM first applies SMOTE based on all minority class examples seen so far to create synthetic examples. It then calculates their distance to the current principle curve in the minority class and only uses those with distance smaller than or equals to the current threshold (δ_{min}) to update OS-ELM and the principle curve in the minority class, thus update the threshold (δ_{min}).

The benefits of using principle curve as a filter for strategic undersampling and synthetic minority oversampling include training the base learner with valuable examples, reducing the risk of losing useful information in the majority class and increasing generalisation in the minority class. The main drawback of PCI-OSELM is the offline initialisation of the OS-ELM. Such a method may be impractical, depending on the cost of collecting minority class examples of the target problem. Also, PCI-OSELM applies SMOTE based on all the minority class examples seen so far, meaning that it needs to store them in the system's memory explicitly. This method not only leads to the memory problem, but also ignores the presence of concept drift in the data stream.

Complementing the aforementioned reviews, a concept drift detector can potentially be combined with a class imbalanced data stream learning approach to address the challenges of concept drift and class imbalance at the same time. However, as it was discussed in Section 3.1.1, resetting the base learner upon drift detection is a simple method but not always ideal.

3.3 Approaches for Learning Drifting Class Imbalanced Data Streams

From the discussions and reviews in Sections 3.1 and 3.2, we can see that concept drift and class imbalance are both challenging topics in data stream learning. Yet, the joint challenge of them can escalate the difficulty to a higher level (S. Wang, Minku, and Yao, 2015). This is because concept drift and class imbalance could affect each other. For example, class imbalance could increase the difficulty in detecting concept drift, especially when the drift only affects the minority class. Meanwhile, concept drift could also lead to very difficult learning of the minority class because there exists particularly few information about the minority class concept after drifts.

Similar to learning class balanced data streams, existing approaches can also be divided into two main categories: explicit and implicit approaches. The difference is that these approaches also need to take class imbalance into the consideration when dealing with concept drift. Section 3.3.1 and 3.3.2 review explicit and implicit approaches to deal with concept drift in class imbalanced data streams respectively.

3.3.1 Explicit Concept Drift Adaptation Approaches

Similar to dealing with concept drift in class balanced data streams, explicit approaches are also intuitive in learning drifting class imbalanced data streams. Yet, it is more challenging because class imbalance could hinder the approach in detecting concept drifts. For example, concept drifts could only affect the minority class and the data stream is severely class imbalanced. This could be particularly difficult in detecting the drift because we have very few number of minority class.

Section 3.3.1.1 reviews some existing concept drift detectors that were proposed to detect concept drift under class imbalance condition. Section 3.3.1.2 - 3.3.1.4 reviews several explicit approaches for learning drifting class imbalanced data streams. These approaches are Oversampling-based and Undersampling-based Online Bagging (OOB and UOB) (S. Wang, Minku, and Yao, 2015), Continuous Synthetic Minority Oversampling Technique (C-SMOTE) (Bernardo, H. M. Gomes, et al., 2020), and Ensemble of Subset Online Sequential Extreme Learning Machine (ESOS-ELM) (Mirza, Lin, and Liu, 2015).

3.3.1.1 Concept Drift Detection for Class Imbalanced Data Stream Learning

In this section, the following concept drift detectors were chosen as the representative approaches to review: Drift Detection Method for Online Class Imbalance (DDM-OCI) (S. Wang, Minku, Ghezzi, et al., 2013), Linear Four Rates (LFR) (H. Wang and Abraham, 2015) and Prequential Area Under Curve in Page Hinkley Test (PAUC-PH) (Brzezinski and Stefanowski, 2014).

Drift Detection Method for Online Class Imbalance (DDM-OCI) proposed by S. Wang, Minku, Ghezzi, et al. (2013) is one of those methods. The design of DDM-OCI is similar to that of DDM. The major difference is that DDM-OCI monitors the recall of the base learner in the minority class rather than the accuracy as DDM does. This idea is based on the fact that accuracy could not truly reflect the predictive performance of the base learner when the data stream has skewed class distribution. Also, the minority class usually is the main interest of the problem, thus, tracking the recall in the minority class is more reasonable than the overall accuracy. Besides, DDM-OCI is class imbalance insensitive because it only monitors the recall in the minority class.

Experiments with artificial data streams of SEA (Street and Y. Kim, 2001) and STAGGER (Schlimmer and Granger Jr., 1986) concepts showed that DDM-OCI reacted to concept

drift faster than DDM, especially in low severity drift. However, it also presented a higher false-positive drift detection rate than DDM. The main drawback of DDM-OCI is that it only considers the concept drift affecting the minority class (H. Wang and Abraham, 2015). However, concept drifts could only affect the majority class but not the minority class. Thus, DDM-OCI is unlikely to detect this kind of drift. Also, DDM-OCI assumes that the minority class is known in advance. Therefore, it cannot handle concept drift which only affects the prior probability $P(Y)$, as this type of concept drift may change the role of majority and minority among two classes.

H. Wang and Abraham (2015) proposed Linear Four Rates (LFR) which aims to overcome the disadvantages of DDM-OCI. LFR monitors the performance of the base learner in four different metrics concerning the confusion matrix. These metrics are true positives rate (TP), true negatives rate (TN), false positives rate (FP) and false negatives rate (FN). LFR confirms a concept drift when there is a significant change in at least one of these rates, based on an improved method to apply statistical test for time decaying rates. In addition, LFR applies a time decaying factor η to each of the four rates to make their past statistics less important to the current time step.

Experiments with ten artificial data streams (including class balanced and class imbalanced data streams) showed that LFR achieved the best true detection rate and the least false-positive drift detection rate when compared against DDM and DDM-OCI. However, the survey work done by S. Wang, Minku, and Yao (2018) showed that LFR could sometimes have a higher false-positive drift detection rate and larger delay of detection than DDM-OCI, depending on the characteristics of the data stream. To further reduce the false-positive drift detection rate, Yu et al. (2019) recently proposed Hierarchical Linear Four Rates (HLFR) which puts an additional layer of permutation test to verify the drift detection given by LFR.

Prequential Area Under Curve in Page Hinkley Test (PAUC-PH) (Brzezinski and Ste-

fanowski, 2014) is another concept drift detection method for class imbalanced data stream learning. Brzezinski and Stefanowski (2014) first proposed a window-based method to estimate the Area Under the Receiver Operating Characteristic Curve (AUC) for data stream learning. In the same work, they then proposed to modify the Page-Hinckley Test (Page, 1954) from monitoring the error rate of the base learner to the prequential AUC performance. As AUC considers both the true-positive and false-positive rates, it is class imbalance insensitive, which makes it suitable for detecting concept drift in class imbalanced data streams. However, it is a window-based method and its effectiveness depends on the window size, which is a hyper-parameter. The survey work done by S. Wang, Minku, and Yao (2018) showed that PAUC-PH is more conservative in terms of drift detection, thus very low true detection rate. That said, S. Wang, Minku, and Yao (2018) also pointed out that the effectiveness of PAUC-PH depends on the type of concept drift and whether resampling method is employed. In particular, PAUC-PH could be more effective in detecting fast and severe drifts. Also, their experiments shows that combining PAUC-PH and OOB performed the best, in terms of minority class recall and G-Mean.

The drift detection methods presented in this section can be used to reset any learning approach to deal with concept drift in a similar way to the methods described in Section 3.1.1.1.

3.3.1.2 Oversampling-based and Undersampling-based Online Bagging (OOB and UOB)

Oversampling-based and Undersampling-based Online Bagging (OOB and UOB) were proposed by S. Wang, Minku, and Yao (2015). Their idea is to combine random oversampling or random undersampling to OB (Oza, 2005). OOB and UOB estimate the current class size based on exponential smoothing with a fading factor θ. Whenever a new example s_t with

a class label y_t arrives, it is first used to calculate the class imbalance ratio of class y_t to the majority class (OOB) or the minority class (UOB). Such ratio is used as the parameter λ of Poisson distribution in OB, thus deciding the number of times to train each ensemble member on s_t.

The idea of OOB and UOB is simple but effective in addressing class imbalance and concept drift affecting $P(Y)$. The method of estimating the current class size with fading factor allows OOB and UOB to be able to adapt to concept drifts affecting the prior probability $P(Y)$ while incorporating random oversampling or random undersampling allows them to handle class imbalance. However, OOB and UOB do not employ any drift detector nor modify the main ensemble (which is OB) explicitly. Thus, they can only handle concept drifts affecting $P(Y)$ only. Besides, only random oversampling or random undersampling is used to reduce the learning bias towards the majority class, thus they likely inherit the disadvantages of these simple methods.

Another major issue is about their training procedure. OOB only oversamples the minority class when the most recent example belongs to the minority class. When the data stream is extremely class imbalanced, the next minority class example is likely to take a long period to arrive, meaning that OOB can only learn from the majority class during this period. Then, the learning bias towards the majority class is likely to build up. Thus, OOB is likely to misclassify the next minority class examples. Meanwhile, with the same situation of the data stream, UOB will have a very small λ during most of time, meaning that it tends to discard lots of majority class from training and wait for the next minority class example.

Even though this approach is unable to cope with drifts other than drifts affecting $P(Y)$, S. Wang, Minku, and Yao (2018) coupled them with with drift detection methods, such as the ones presented in Section 3.3.1.1, to become explicit drift handling approaches.

Regarding the memory management strategy, OOB and UOB both do not store any

past models in their memory but continuously train the base learner with incoming examples. Even though S. Wang, Minku, and Yao (2018) coupled them with with drift detection methods to address concept drifts other than drifts affecting $P(Y)$, their base learner is reset upon drift detection. Hence, they both do not have the advantage of exploiting past knowledge to help learning the post-drift concept. Nevertheless, they can effectively handle drifts affecting $P(Y)$ without the help of past knowledge because of their adaptive resampling strategy based on the estimated class imbalanced ratio.

3.3.1.3 Continuous Synthetic Minority Oversampling Technique (C-SMOTE)

Continuous Synthetic Minority Oversampling Technique (C-SMOTE) (Bernardo, H. M. Gomes, et al., 2020) is a very recent approach that brought the idea of SMOTE to data stream learning. The idea of C-SMOTE adopts Adaptive Windowing (ADWIN) (Bifet and Gavaldà, 2007) to store the most recent examples. It then applies SMOTE to the minority class examples in the ADWIN to create synthetic minority class examples for oversampling. Meanwhile, the ADWIN is also used to monitor the error rate of the base learner, as a strategy to detect concept drift. If a concept drift is detected, the older sub-window of the ADWIN is discarded. However, such strategy is not responsible to adapt the base learner to concept drift. It only makes sure that the examples in the ADWIN belongs to the current concept. This means that there is another concept drift detector incorporated if the base learner is an explicit learner.

Using ADWIN to make sure that the examples in the memory belong to the current concept is one advantage of C-SMOTE. However, memory consumption is still an issue if the data stream does not present any concept drift, triggering the ADWIN to drop the older sub-window. To overcome this issue, Bernardo and Della Valle (2021) proposed a successor to C-SMOTE, called Very Fast Continuous Synthetic Minority Oversampling Technique

(VFC-SMOTE), which uses "sketches" to summarise the feature distribution of the examples without storing the examples themselves explicitly.

C-SMOTE controls the oversampling rate based on the class imbalance ratio over the whole ADWIN, which may not truly reflect the class imbalance ratio at the current time step t. Besides, if the base learner is an explicit approach, its concept drift detector may not always agree with the drift detection by the ADWIN. Thus, conflicts could happen and result in unstable predictive performance. Moreover, C-SMOTE is a single learner approach and it does not store any past models into its memory. So, it lost the advantage of exploiting relevant past knowledge to deal with concept drifts.

3.3.1.4 Ensemble of Subset Online Sequential Extreme Learning Machine (ESOS-ELM)

Ensemble of Subset Online Sequential Extreme Learning Machine (ESOS-ELM) (Mirza, Lin, and Liu, 2015) is an ensemble approach that was proposed to deal with the joint challenge of concept drift and class imbalance. In addition, it also has a memory to store past knowledge, as a strategy to handle recurring concept. The general idea of ESOS-ELM is as follows.

ESOS-ELM maintains an ensemble of Online Sequential Extreme Learning Machine (OS-ELM) (Liang et al., 2006) and a pool of Weighted Extreme Learning Machines (WELM) (Mirza, Lin, and Toh, 2013) to store past knowledge. ESOS-ELM makes predictions by weighted majority vote over the main ensemble. It deals with class imbalance by training each OS-ELM with approximately equal number of majority and minority class examples. Concretely, each minority class example is processed by m OS-ELMs, where m is the class imbalance ratio, while the majority class examples are processed in a round robin fashion. During training, the OS-ELMs in the main ensemble use the aforementioned procedure to process new examples. At the same time, an individual module, called ELM-store, contin-

uously updates the statistics to summarise the incoming examples of the current concept. These statistics are used to build the WELM.

As for dealing with concept drift, ESOS-ELM has both implicit and explicit adaptation strategies. For implicit strategies, it has similar strategy as DWM to add and remove OS-ELMs in the main ensemble at every p time steps. The voting weights of the OS-ELMs are proportional to their G-Mean on a separate validation set that is collected from the same environment as the training stream. For explicit strategy, ESOS-ELM uses a threshold-based strategy together with a hypothesis testing to detect any significant change in the predictive performance of the main ensemble, thus reports concept drift. Therefore, ESOS-ELM is not entirely an implicit approach but a hybrid approach. Upon concept drift detection, a WELM is built using the information summarised by the ELM-store. After that, the ELM-store is reset to prepare for the new concept. Meanwhile, if any WELM performs better than the main ensemble, indicating a possible recurring concept, it will be introduced to the main ensemble with a voting weight of one.

The advantage of ESOS-ELM is that it considers to use relevant past knowledge to handle concept drift in class imbalanced data stream. Also, the method of using ELM-store to summarise the incoming examples of the current concept is memory efficient. However, the authors did not specify what to do when the WELMs fill up the memory. They seem to just allow ESOS-ELM to keep storing new WELMs, which will cause memory issues. Besides, ESOS-ELM is restrictive in terms of base learner type. It only allows to use ELMs.

3.3.2 Implicit Concept Drift Adaptation Approaches

Implicit approaches to deal with concept drift in class imbalanced data streams usually update the voting weights of the base learner continuously to reflect the any potential changes

in the data stream. However, in class imbalanced data stream learning, such weighting strategy also needs to consider that the base learner may bias towards the majority class. To address this issue, one can use a cost-sensitive approach which penalise more in the weight of the base learners performing poorly in the minority class. Another method is the data-level approach which employs a resampling method to reduce the learning bias.

In this section, the following approaches from the literature were chosen to review in details: Learn++ for Concept Drift with SMOTE (Learn++.CDS) (Ditzler and Polikar, 2013), Learn++ for Non-stationary and Imbalanced Environment (Learn++.NIE) (Ditzler and Polikar, 2013), Dynamic Weighted Majority for Imbalanced Learning (DWMIL) (Lu, Cheung, and Tang, 2017), and Heuristic Updatable Weighted Random Subspaces with Instance Propagtion (HUWRS.IP) (Hoens and N. V. Chawla, 2012).

3.3.2.1 Learn++ for Concept Drift with SMOTE (Learn++.CDS) and Learn++ for Non-stationary and Imbalanced Environments (Learn++.NIE)

Learn++ for Concept Drift with SMOTE (Learn++.CDS) and Learn++ for Non-stationary and Imbalanced Environment (Learn++.NIE) were proposed by Ditzler and Polikar (2013) in the same work. They were both built on idea of Learn++ for Non-stationary Environment (Learn++.NSE) (Polikar et al., 2001) which is a batch-based data stream learning approach for learning drift class balanced data streams.

The idea of Learn++.CDS is simply combining the Learn++.NSE with SMOTE. Whenever a batch D of training examples arrives, Learn++.CDS first updates a distribution of example weights by evaluating the ensemble members on D. The weight of a given example increases when an ensemble member misclassifies it. After that, SMOTE is applied to oversample the minority class in D to obtain a set of synthetic minority class examples S. A new ensemble member is then created and built on the batch $D \cup S$. Lastly, all the

ensemble members including the new one are evaluated on D to obtain their voting weights. Learn++.CDS makes predictions by a weighted majority vote over the whole ensemble. Thus, Learn++.CDS is a batch-based data stream learning approach which attached a data-level strategy to handle class imbalance.

Learn++.NIE does not use any form of oversampling strategy to handle class imbalance. Instead, it creates sub-ensembles, where all base learners train on the same set of minority class examples from D but each of them trains on a different subset of the majority class examples from D. Note that the size of such subset should equal to the number of minority class examples in D. Also, Learn++.NIE has a different weighting function from Learn++.CDS. It puts higher voting weight to ensemble members that perform well in both classes. This weighting mechanism can be fine-tuned to put more emphasis to the majority or minority class, depending on the requirements of the problem.

The idea of Learn++.CDS is intuitive and successfully applied SMOTE to data stream learning, while Learn++.NIE's method of creating different class balanced training sets to build sub-ensemble does not require to replicate or to augment the minority class examples nor to discard any majority class examples. The major drawback of these approaches is the fact that they are both batch-based approaches. They assume that concept drift does not happen within a batch of examples. Also, when the data stream is severely class imbalanced, both Learn++.CDS and Learn++.NIE could struggle because there could be some batches which has no minority class example. This means that, Learn++.CDS's method of applying SMOTE to the most recent batch will not work, while Learn++.NIE's method to create different class balanced training set will not work neither.

In terms of exploiting relevant past knowledge, both Learn++.CDS and Learn++.NIE retain past models in their ensemble. If a given model is deemed to be irrelevant to the current concept, they are assigned with a zero weight, thus temporarily excluded from making

prediction. When the model is deemed to be relevant to the current concept again, they will then be assigned with a non-zero weight to participate making prediction. However, Ditzler and Polikar (2013) did not specific whether Learn++.CDS and Learn++.NIE have a limited memory. They seem to allow these approaches to keep growing in number of base learners. This could be problematic if we allow these two approaches to learn from an infinite long data stream indefinitely because their memory are likely overflown.

3.3.2.2 Dynamic Weighted Majority for Imbalance Learning (DWMIL)

Dynamic Weighted Majority for Imbalanced Learning (DWMIL) (Lu, Cheung, and Tang, 2017) is a batch-based ensemble learning approach to deal with drifting class imbalanced data streams. DWMIL is generally inspired by DWM (Kolter and Maloof, 2003). Its idea is simple and can be described as follows.

Whenever a new batch of examples arrives, DWMIL uses it to evaluate the ensemble members and update their voting weights proportionally to their F1 measure or G-Mean, based on the evaluation metric for class imbalance learning used. Then, DWMIL prunes base learners with weight less than a predefined threshold θ. After that, a new base learner is built using this most recent batch of examples and is introduced to the main ensemble with an initial voting weight of one. This strategy is similar to DWM. It allows the gradual adaptation to any potential concept drift in the data stream. As for dealing with class imbalance, DWMIL simply adopts UnderBagging (S. Wang and Yao, 2009), which is an offline ensemble learner for learning class imbalanced data set, as the base learner. In other words, DWMIL did not proposed any specific strategy to deal with class imbalance, except using a class imbalance insensitive metric to evaluate and weight the base learners.

As the design of DWMIL is similar to DWM, it inherits the drawback of DWM's memory strategy which always prunes base learners performing poorly on the current concept.

Thus, DWMIL also loses the advantage of exploiting relevant past knowledge to handle recurring concepts or concepts that are similar to a previous one but different from the current one.

3.3.2.3 Heuristic Updatable Weighted Random Subspaces with Instance Propagtion (HUWRS.IP)

Heuristic Updatable Weighted Random Subspaces with Instance Propagtion (HUWRS.IP) (Hoens and N. V. Chawla, 2012) is a batch-based learning approach to deal with drifting class imbalanced data streams. It is based on the approach of Heuristic Updatable Weighted Random Subspaces (HUWRS) (Hoens, N. V. Chawla, and Polikar, 2011) which was proposed to learn drifting class balanced data streams. In particular, the main novelty of HUWRS.IP is the example selection mechanism, called Instance Propagation, which selects relevant past minority class examples for oversampling the most recent train batch.

Instance Propagation selects relevant past minority class examples by first building a Naïve Bayes classifier on the most recent training batch. This method attempts to approximate the class conditional probabilities $P(X|Y)$ of the most recent train batch. After that, the Naïve Bayes classifier predicts the probability of each past minority class example belong to the current minority class. As for dealing with concept drift, HUWRS.IP relies on HUWRS to use Hellinger distance to weight the base learners.

The advantage of using Naïve Bayes classifier to select relevant past minority class examples for oversampling is that we do not need to define the similarly boundary between two probability distributions. However, such Naïve Bayes classifier is built upon the most recent training batch only. It could also suffer from class imbalance. Thus, it could be unstable because there are not enough minority class examples, allowing it to model the class conditional probabilities $P(X|Y)$ correctly. Besides, the method of Instance Propagation

relies on finding examples that are relevant (similar) to the current minority class concept. However, these examples may not exist in the system memory. Also, this method requires the system to store all minority class examples, which likely leads to memory problems. Though, the authors argued that the minority class will only represent a manageable number of instances because of class imbalance. Yet, the memory will still overflown eventually if we consider to let HUWRS.IP to learn from an infinite long data stream.

3.4 Evaluation Methods for Data Stream Learning

To evaluate the performance of a data stream learning approach, traditional evaluation methods for offline learning could not be applied. In offline learning, the data set has finite number of examples. The whole data set is assumed to be static, meaning that the examples are drawn from the same joint probability distribution. It is also assumed that there is no temporal order among the examples within the data set. In contrast, a data stream could potentially have an infinite number of examples where the examples arrive in sequence. Concept drift could happen at any time, meaning that the examples in the data stream are not necessarily drawn from the same joint probability distribution and respecting chronology is important.

One intuitive evaluation method for data stream learning is to train the data stream learning approach with the training stream while to test it periodically using a separate holdout test set. Such test set should be drawn from the same joint probability distribution as the training stream. In the case of concept drift, a new test set, which is drawn from the new concept after the drift, has to be created.

This method allows us to obtain an idea of how well the data stream learning approach does in generalising to unseen examples. This is particularly helpful in evaluating class im-

balanced data stream learning approaches with class imbalanced holdout test set. However, the main limitation is that it is impractical in most real-world applications because it is difficult to know when concept drift happens in advance. Therefore, it is typically impractical to ensure that an appropriate holdout test set can be collected. Also, collecting additional examples to create the test set is usually expensive. Evaluation based on a holdout set is thus usually only feasible for synthetically generated data streams.

Gama, Rqauel Sebastião, and Rodrigues (2013) proposed an efficient method to evaluate the data stream learning approach sequentially. This method is called prequential evaluation. The term "prequential" refers to predictive sequential. The general idea is to use the most recent example to first evaluate the data stream learning approach then to train it. The predictive performance is incrementally computed over time based on exponential decay. Equation 3.1 presents the performance estimation of a data stream learning approach D at each time step.

$$preq_perform_t(D) = \begin{cases} preq_perform_t^{ex}(D), & \text{if } t = f \\ \frac{preq_perform_t^{ex}(D) + \alpha \times preq_perform_{t-1}(D) \times (t-f)}{t-f+1}, & \text{otherwise} \end{cases} \tag{3.1}$$

where t is the current time step, ex is the most recent example, f is the first time step used in the calculation and $0 < \alpha \leq 1$ is a time decay factor. It calculates the cumulative sum of the loss function $L(y_i, \hat{y}_i)$ (i.e. $preq_perform_t^{ex}(D) = L(y_i, \hat{y}_i)$) with a time decaying factor α to make the previous performance less important to the current time step, where y_i is the true label of ex and $\hat{y}_i$ is the predicted label by D. Larger value of α makes the past performance fades gradual, while smaller value of α put more emphasis to the most recent performance. Note that, this idea of estimating performance prequentially is not limited to estimate the error or the accuracy of the data stream learning approach. It also can be applied to estimate the time decay recalls in each class, or the time decay G-Mean for class

imbalanced data stream learning, depending on the error function $L(y_i, \hat{y}_i)$ used.

Prequential evaluation has been widely used in the literature of data stream learning. It overcomes the issue of periodic holdout test evaluation which requires additional examples to create test sets. Also, it is almost memory-less as it does not require to store any examples. When applying it to experiments with artificial drifting data stream, we can potentially reset the prequential performance upon known concept drift point to eliminate the impact of the performance on the past concept to the performance on the current concept.

Bifet, Morales, et al. (2015) brought the idea of k-fold validation from offline learning to prequential evaluation. They proposed three variations of prequential evaluation: k-fold distributed cross-validation, k-fold distributed split-validation, and k-fold distributed bootstrap validation.

The general idea of these validation methods is to make k copies of the target data stream learning model and evaluate them by testing and training them with different subset of examples from the data stream in parallel. Whenever a new example s_t arrives, the aforementioned validation methods have the following strategies:

- **k-fold distributed cross-validation**: a random copy of the target data stream learning approach is tested on s_t while the other copies are trained on s_t.
- **k-fold distributed split-validation**: $(k-1)$ copies of the target data stream learning approach are randomly chosen without replacement and are tested on s_t while the last copy is trained on s_t.
- **k-fold distributed bootstrap validation**: each copy of the target data stream learning approach is trained on s_t for n $Poisson(1)$ times. This means that s_t is used for training in approximately two-thirds of the k copies of the target data stream learning approach and for testing the rest. This idea is also used by Online Bagging

(Oza, 2005).

The main advantage of these validation methods is that they can take into account slightly different samples of the underlying distributions, because they basically compute the k runs average performance of the data stream learning approach on k different versions of the data stream. However, these validation methods have a serious problem when evaluating data streams with concept drift and/or class imbalance, because they change the characteristics of the data stream, potentially modifying the type of concept drift and the class imbalance ratio of the data stream. This means that they cannot be used to evaluate data stream learning approaches on real-world data streams. Hence, the original prequential evaluation is adopted in this thesis.

Apart from evaluation methods, evaluation metrics are also important, especially in evaluating approaches on class imbalanced data stream.

Traditionally, accuracy or error rate are commonly used as the performance metrics. However, they are strongly biased towards the majority when the data stream is class imbalanced. For example, consider a finite binary classification data stream consisting of ninety majority class and ten minority class examples, and a naïve approach predicting all the examples to belong to the majority class. Such approach will achieve 90% accuracy (without applying any fading factor), which is a very high. Yet, it performed poorly on the minority class. Therefore, other metrics are adopted when evaluating class imbalanced learning.

As this thesis concentrates on binary classification tasks, thus there exists a majority and a minority class when the data stream is class imbalanced. Table 3.1 presents the confusion matrix for binary classification problem.

Based on the Table 3.1, accuracy, error rate, recall, specificity and precision can be formally defined as follows:

Table 3.1: Confusion Matrix for Binary Classification Problem

	Predicted Positive	Predicted Negative
Actual Positive	True positive (TP)	False negative (FN)
Actual Negative	False positive (FP)	True negative (TN)

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.2}$$

$$ErrorRare = 1 - Accuracy \tag{3.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.4}$$

$$Specificity = \frac{TN}{TN + FP} \tag{3.5}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.6}$$

Recall is also known as positive class recall, true positive rate (TPR), hit rate or sensitivity. It measures the rate of the true positive class examples being correctly classified. Specificity can also be called negative class recall, true negative rate (TNR), or selectivity. It measures the rate of the true negative class examples being correctly classified. Precision is also known as positive predictive value. It measures the rate of the amount of true positive class examples from all the examples that were predicted as positive class. F-measure or F-score (Van Rijsbergen, 1979) is a harmonic mean between recall and precision. It is computed as follows:

$$F_\beta = \frac{(1+\beta^2) \cdot precision \cdot recall}{\beta^2 \cdot precision + recall} \tag{3.7}$$

where β is a hyper-parameter to shift the importance between precision and recall. A higher value of β puts more emphasis to recall (Van Rijsbergen, 1979). β is usually set to a value of one.

The Geometric Mean (G-Mean) was first proposed by Kubat and Matwin (1997) as an evaluation metric. Their idea is that a good classifier should have good accuracy in both positive and negative classes, i.e., high TPR and TNR. With this idea, they proposed to take the geometric mean between TPR and TNR. Therefore, G-Mean is computed as follows:

$$Gm = \sqrt{\frac{TP}{TP+FN} \times \frac{TN}{FP+TN}} \tag{3.8}$$

Among the aforementioned evaluation metrics, however, not all of them are suitable for the evaluation in class imbalanced learning. In particular, only recall, specificity and G-Mean are class imbalance insensitive (He and Garcia, 2009; S. Wang, Minku, and Yao, 2018), meaning that their ability to evaluate the machine learning approach will not be affected by the skewed class distribution. On the other hand, accuracy, error rate, precision and F-measure are class imbalance sensitive because these measures involve values from both rows of the confusion matrix for addition or subtraction. (He and Garcia, 2009; S. Wang, Minku, and Yao, 2018).

Another well-known evaluation metric for class imbalanced learning is the Area Under the Receiver Operating Characteristic (AUROC or simply AUC) curve (Fawcett, 2004; Fawcett, 2006). The ROC curve shows the trade-off between true positive rate (TPR) and false positive rate at different classification thresholds. Figure 3.1 shows an example of the ROC curve.

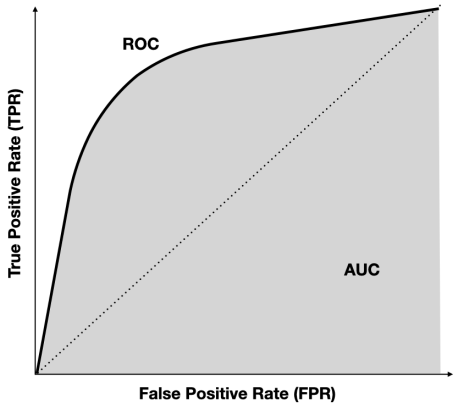


Figure 3.1: Example of ROC Curve and the Area Under ROC Curve (AUC)

A perfect classifier has a 100% TPR and 0% FPR, regardless the classification thresholds, i.e., the ROC curve has a right angel at the top left corner of the plot. Thus, any good classifier should also has a ROC curve with an elbow close to the top left corner. This stems the idea of calculating AUC to aggregate the performance of the machine learning approach at different classification thresholds.

The calculation of AUC, however, requires to store all the prediction made by the machine learning approach, which is infeasible in data stream learning. To overcome this issue, Brzezinski and Stefanowski (2014) and Brzezinski and Stefanowski (2017) proposed the prequential AUC (PAUC) which uses a sorted tree structure with a sliding window to compute AUC with forgetting. Experiments with finite data sets showed that PAUC is statistically consistent with the traditional AUC. However, in the work by S. Wang and Minku (2020), it has been found that AUC-based evaluation metrics can be optimistic whereas G-Mean reflects the performance in minority class better. In addition, when considering different prediction thresholds for computing the ROC curve, AUC also takes thresholds that are unrealistic for use in practice into the consideration, potentially leading to misleading performance evaluations that are not usable for real-world problems.

3.5 Chapter Summary

Lots of data stream learning approaches have been proposed in the past years. However, most of them concentrate on either the challenge of concept drift or class imbalance. The area of the joint challenge of concept drift and class imbalance still has a lot to explore. Hence, this chapter first reviewed the existing work of concept drift adaption in class balanced data streams, class imbalanced learning, and recent work that address the joint challenge of them, outlining their strengths and limitations.

Section 3.1 reviewed the existing work that deal with concept drift in class balanced data stream explicitly and implicitly. Explicit approaches can provide a swift adaptation to drift, as they estimate the exact point the drift occurs. Yet, they could suffer from false-positive drift detections. Implicit approaches accept that the underlying concept could change anytime. They continuously evolve themselves to adapt to any potential concept drift. However, they usually take longer time than explicit approaches in reacting to and recovering from concept drift. Nevertheless, existing approaches in both categories usually discard past knowledge that is irrelevant to the current concept or by chronological order. These past knowledge could potentially be useful in adapting to future concept drifts. Therefore, new memory management strategies are needed to maintain a variety set of past knowledge in the system's memory. Also, these approaches could not be applied to learning drifting class imbalanced data stream directly because they usually deal with concept drift by monitoring their prequential accuracy or error rate.

Section 3.2 reviewed the existing work in class imbalanced learning, including algorithm-level and data-level strategies. Although this thesis concentrates on data stream learning, we also reviewed some existing techniques that deal with class imbalance in offline learning to see what are limitations in bringing these ideas to data stream learning. The main drawback of the algorithm-level strategies are the fact that they were designed for a particular learning

approach. When such learning approach does not perform well on a given data stream, it is difficult to transfer the same algorithm-level strategies to another learning approach and evaluate the effectiveness of the strategy. This may be the reason why most existing work focuses on data-level strategy as they are not specific to any learning approach. However, except the simple oversampling and undersample strategy, most existing data-level strategy requires the availability of all the training examples, which is infeasible in data stream learning. Therefore, new strategies are needed to develop to handle class imbalanced data stream learning effectively and efficiently.

Section 3.3 reviewed the existing work in dealing with the joint challenge of concept drift and class imbalance in data stream learning. Similarly, these approaches can also be divided into explicit and implicit drift adaptation approaches. In the former category, we reviewed the existing concept drift detector for class imbalance data stream learning as well as approaches with adaptation strategies in response to the drift detections. However, these approaches are either batch-based approaches or only able to handle a subset of concept drifts in class imbalanced data streams. More recent work attempts to use a sliding window based method to bring the successful offline data-level strategy to handle class imbalance data streams. Yet, they ignore the fact that concept drift could happen within the sliding window or concept drift may not always be captured by the drift detector. In the latter category, most of the existing implicit approaches to deal with concept drift in class imbalanced data streams are batch-based approach. They assume that concept drift does not occur within each batch of examples, which is not always true in real-world applications. Meanwhile, some approaches consider to exploit relevant past knowledge to help in adapting the concept after drift. However, they require to store all the past knowledge in the system's memory, which is infeasible in data stream learning.

Table 3.2 summarises the main strengths and weaknesses of existing approaches reviewed in this chapter..

As we can see from the review in this chapter, strategies to preserve which past models in the memory (memory management strategies) are crucial in adapting concept drift in both class balanced and imbalanced data streams. However, existing work is limited to naïve strategies to remove models from the memory, such as deleting the oldest model or the model that currently performs the worse. These strategies may easily lose relevant past knowledge. No existing work has investigated how well these strategies perform and whether other better alternatives could increase the chances of maintaining relevant knowledge in the memory. Such study is presented in Chapter 4, where a new memory-based framework is proposed to handle multiple types of drifts, including recurrent, abrupt, gradual, severe and mild-severe drifts, in class balanced data streams. Meanwhile, Chapter 5 investigates this idea further on class imbalanced data streams and proposed a new set of memory strategies to enable the memory-based framework to handle multiple types of drifts, including recurrent, abrupt, gradual, severe and mild-severe drifts, in class imbalanced data streams. Besides, existing strategies for learning drifting class imbalanced data streams usually have memory issues or depend on the effectiveness of the drift detector. Chapter 6 provides new insights in solving these issues and proposed a stream clustering based strategy to produce synthetic examples for oversampling.

Table 3.2: Summary of Strengths and Weaknesses of Reviewed Approaches

Approach	Explicit[a]/ Implicit[a]	Strengths	Weaknesses	Issues of Memory Management Strategy
Approaches for Learning Drifting Class Balanced Data Streams				
Base Learner[b]+ Drift Detector[c]	Explicit	• Intuitive design. • Easy to implement.	• Unlikely to handle gradual or low severity drifts well. • Unlikely to handle drift properly when the data stream is class imbalanced.	• Base learner is always reset upon drift detection. • Does not store any past base learner. • Lost the advantage of exploiting past knowledge.
DDD	Explicit	• Able to deal with drifts of different speed and severity. • Robust to false-positive drift detections. • Exploit pre-drift knowledge to learn the post-drift concept.	• Unlikely to handle recurrent drift well. • Weighting metric is prequential accuracy, which is class imbalance sensitive.	• Only retain past knowledge prior to the current drift. • Lost the advantage of various past knowledge to exploit.
RCD	Explicit	• Using a statistical test to identify recurring concepts.	• Strategy to exploit past knowledge is conservative. • Past knowledge can only be exploit when it is very similar to the post-drift concept. • Not designed for class imbalanced data stream learning.	• Uses First in First Out (FIFO) to manage a memory of past models. • Not always ideal, as a very old concept could reappear at any time.

DWM	Implicit	• Continuously improve the ensemble to reflect current underlying concept, based on base learners' error rate. • Can handle gradual drifts well.	• Likely to have delay in adapting abrupt drift, depending on hyper-parameter p. • Struggle in dealing with recurrent drift. • Weighting metric is prequential accuracy, which is class imbalance sensitive.	• Pruning ensemble members which perform poorly to current concept. • Not always ideal, as concepts that are very different from current concept could reappear at any time.
OAUE	Implicit	• Memory efficient weighting function to achieve predictive performance as good as a batch-based ensemble learner. • Continuously improve the ensemble to reflect current underlying concept, based on base learners' error rate. • Can handle gradual drifts well. • Achieved promising results in most class balanced data stream learning studies.	• Likely to have delay in adapting abrupt drift. • Struggle in dealing with recurrent drift. • Weighting metric is MSE, which is class imbalance sensitive.	• Pruning ensemble members which perform poorly to current concept. • Not always ideal, as concepts that are very different from current concept could reappear at any time.
Approaches for Learning Stable Concept Class Imbalanced Data Streams				

GH-VFDT & HD-VFDT	N/A	• Strong theoretical algorithm-level strategy to address class imbalance. • Low computational resource consumption.	• Strategy to address class imbalance is tailor-made for themselves. • Strong parametric assumption to class distributions. • Does not explore the true decision boundaries between classes. • Not designed to deal with concept drift.	N/A
OnlineUnder-OverBagging	N/A	• Brought the idea of combing Undersampling-based Bagging and Oversampling-based Bagging into online learning. • Such an idea further boosts the diversity in the main ensemble to improve predictive performance (S. Wang and Yao, 2009). • Low memory consumption.	• Constant resampling rate. • Does not explore the true decision boundaries between classes. • Not designed to deal with concept drift.	N/A

Online-SMOTEBagging	N/A	• Replaced simple oversampling in OnlineUnderOverBagging with SMOTE (N. V. Chawla et al., 2002). • Likely to explore the true decision boundaries between classes because of SMOTE (N. V. Chawla et al., 2002).	• Constant resampling rate. • Exceptionally high memory consumption because of SMOTE (N. V. Chawla et al., 2002). • Not designed to deal with concept drift.	N/A
PCI-OSELM	N/A	• Using principle curve (Hastie and Stuetzle, 1989) as a filter to reduce the risk of losing useful information in the majority class and over generalisation in the minority class. • Likely to explore the true decision boundaries between classes because of SMOTE (N. V. Chawla et al., 2002).	• Offline initialisation of OS-ELM may be impractical as minority class examples are likely expansive to collect. • Exceptionally high memory consumption because of SMOTE (N. V. Chawla et al., 2002). • Not designed to deal with concept drift.	N/A
Approaches for Learning Drifting Class Imbalanced Data Streams				

Base Learner[b]+ Drift Detector[d]	Explicit	• Intuitive design. • Easy to implement.	• Unlikely to handle gradual or low severity drifts well. • Only able to detect drifts in class imbalanced data stream. • Additional strategy is needed to address learn bias due to class imbalance.	• Base learner is always reset upon drift detection. • Does not store any past base learner. • Lost the advantage of exploiting past knowledge.
OOB	Explicit	• Simple yet efficient strategy to deal with class imbalance and drift affecting $P(Y)$.	• Only able to deal with drifts affecting $P(Y)$ • Likely to cause overfitting in the minority class because of simple oversampling. • Perform oversampling only when the current examples belongs to the minority class.	• Does not store any past base learner. • Lost the advantage of exploiting past knowledge.
UOB	Explicit	• Simple yet efficient strategy to deal with class imbalance and drift affecting $P(Y)$.	• Only able to deal with drifts affecting $P(Y)$ • Likely to lose useful information in the majority class because of simple undersampling.	• Does not store any past base learner. • Lost the advantage of exploiting past knowledge.

C-SMOTE	Explicit	• Partially solved the memory issue of OnlineSMOTEBagging (B. Wang and Pineau, 2016) using ADWIN (Bifet and Gavaldà, 2007). • Likely to explore the true decision boundaries between classes because of SMOTE (N. V. Chawla et al., 2002).	• ADWIN (Bifet and Gavaldà, 2007) may not always agree with the drift detector in reporting concept drift. Thus, the current window may contain examples from other concepts. • ADWIN (Bifet and Gavaldà, 2007) is likely to be overflown if there is no concept drift.	• Does not store any past base learner. • Lost the advantage of exploiting past knowledge.
ESOS-ELM	Explicit	• Has a memory of past models to exploit when dealing with drifts. • Weighting metric is G-Mean, which is class imbalance insensitive.	• Restrictive in terms of base learner type – OS-ELM (Liang et al., 2006).	• Does not specific any strategy to manage the memory of past model, which likely causes memory issues.

Learn++.CDS	Implicit	• Simple batch-based approach which combine Learn++.NSE (Polikar et al., 2001) with SMOTE (N. V. Chawla et al., 2002). • Strong foundation as Learn++.NSE (Polikar et al., 2001) and SMOTE (N. V. Chawla et al., 2002) have been well-studied in the literature.	• Likely to have delay in adapting abrupt drift, depending the batch size which is a hyper-parameter. • May struggle when the data stream is severely class imbalanced, as there may exist some batch having no minority class example. Thus, SMOTE (N. V. Chawla et al., 2002) would not work. • Assuming concept drift does not happen within any batch of examples.	• Does not specific any strategy to manage the memory of past model, which likely causes memory issues.
Learn++.NIE	Implicit	• Batch-based approach which creates sub-ensembles to balance the training batch to each base learner. • Strong foundation as Learn++.NSE (Polikar et al., 2001) has been well-studied in the literature.	• Likely to have delay in adapting abrupt drift, depending the batch size which is a hyper-parameter. • May struggle when the data stream is severely class imbalanced, as there may exist some batch having no minority class example. Thus, sub-ensemble method would not work. • Assuming concept drift does not happen within any batch of examples.	• Does not specific any strategy to manage the memory of past model, which likely causes memory issues.

DWMIL	Implicit	• Strong foundation as DWM and UnderBagging (S. Wang and Yao, 2009) have been well-studied in the literature. • Weighting metric is F1 measure or G-Mean, which are class imbalance insensitive.	• UnderBagging (S. Wang and Yao, 2009) as base learner may not ideal for data stream learning. • Likely to have delay in adapting abrupt drift, depending on hyper-parameter p. • Struggle in dealing with recurrent drift.	• Same as DWM (Kolter and Maloof, 2003), pruning ensemble members which perform poorly to current concept. • Not always ideal, as concepts that are very different from current concept could reappear at any time.
HUWRS.IP	Implicit	• Proposed strategy using Naïve Bayes classifier to select relevant past minority class examples for oversampling, which is an intuitive method to exploit relevant past knowledge. • As Naïve Bayes classifier used, no similarity measure is needed. • Weighting metric is Hellinger distance, which is class imbalance insensitive.	• Naïve Bayes classifier could also suffer from class imbalance • Relevant past minority class examples may not always exist.	• Does not store any past base learner but a memory of past minority class examples. • Memory issues arises when HUWRS.IP learns from an infinite long data stream.

[a] In terms of drift adaptation.

[b] Base Learner refers to as any data stream learning approach which does not adapt to concept drift and class imbalance. For example, Hoeffding Tree with Naïve Bayes at Leaves (HTNB).

[c] Any existing drift detector for class balanced data stream learning, such as those reviewed in Section 3.1.1.1.

[d] Any existing drift detector for class imbalanced data stream learning, such as those reviewed in Section 3.3.1.1

Chapter Four

Diversity-based Memory Strategies for Concept Drift Adaptation in Class Balanced Data Streams

RQ1: Can diversity be used to create a memory-based framework able to tackle different types of drifts, including recurrent, abrupt, gradual, severe and mild-severe drifts more efficiently and effectively? How well does this framework perform?

The discussion in Section 3.1 has shown that most existing approaches proposed to deal with concept drift in class balanced data streams focus on how to detect drift timely and how to update the main ensemble/learner such that it can adapt to changes in the underlying concept. We can rarely see any existing work attempting to exploit past knowledge to deal with concept drift. The approaches DDD and RCD discussed in Section 3.1 are two representatives of the very few approaches that attempt to use past knowledge. They have shown that past knowledge is indeed helpful in adapting to concept drift.

RCD has shown that recovering a past model to continue learn the recurring concept

is effective. DDD has shown that an ensemble trained on the pre-drift concept with high diversity can converge to the post-drift concept when training with low diversity, being helpful in drift adaptation and robust to false-positive drift detection. However, the fact that they either adopt a First in First Out (FIFO) memory management strategy where the oldest models are removed (RCD) or only preserve the knowledge of the concept prior to the most recent drift (DDD) could be disadvantageous when recurrent drift happens after lots of concept drift leading to other concepts. Other approaches that exploit past knowledge have similar disadvantages (Gama and Kosina, 2009; Abad, J. B. Gomes, and Ruiz, 2016; Ramamurthy and Bhatnagar, 2007; J. B. Gomes et al., 2014; Abdallah et al., 2016). Meanwhile, approaches that continuously evolve their main ensemble, such as OAUE and DWM, can also be considered as exploiting recent past knowledge to deal with concept drift implicitly. Yet, the elitist strategy adopted by these approaches, where models perform poorly on the current concept are removed, is not always ideal neither. As these currently poor performing models could perform very well on other concepts, thus, they could be helpful when drifts leading to those concepts. Section 3.1 discusses the strengths and weaknesses of the existing approaches in more details, particularly to their memory management strategies.

As we can see, there is a need for a new strategy to manage a memory of past models, allowing the learning system to handle recurrent drifts in different situations. This chapter first presents a preliminary study which investigates the impact of different memory management strategies in handling recurrent concept. Besides, a novel approach with a diversity-based memory management strategy called Diversity Pool (DP) is proposed to answer research question 1 posed in Section 1.2.1: *"Can diversity be used to create a memory-based framework able to tackle different types of drifts, including recurrent, abrupt, gradual, severe and mild-severe drifts more efficiently and effectively? How well does this framework perform?"*

The idea is that, if we can maintain a diverse set of models representing different concepts / knowledge in the memory, we can have a higher chance to be ready for different concepts recur. Although there are existing work studying different diversity measures and how they impact the predictive performance of ensembles (Minku and Yao, 2012; Kuncheva and Whitaker, 2003; Minku, White, and Yao, 2010), no existing work provides a detailed analysis on whether, when and to what extent diversity can be used to manage a memory of past knowledge to deal with recurrent drifts.

In addition, DDD has shown that diversity can be applied to handle concept drifts other than recurrent drift. Thus, we posit that the benefit of diversity could go beyond that and be used as an overall framework to handle multiple types of drifts, including recurrent, abrupt, gradual, severe and mild-severe drifts. Thus, a diversity-based framework called Concept Drift Handling Based on Clustering in the Model Space (CDCMS) is proposed to continue answer research question 1 posed in Section 1.2.1. In particular, a novel clustering in the model space strategy is proposed to identify and recover relevant knowledge and to create ensembles with different diversity levels, handling recurrent, abrupt, gradual, severe and mild-severe drifts. An improved diversity-based memory management strategy is also incorporated into this framework. As the literature on data stream learning has a gap in the understanding of how diversity could be used to deal with recurrent drifts in addition to other types of drift, this chapter focuses on class balanced data streams. The potential benefit of memories for class imbalanced data streams will be investigated in Chapter 5.

Experiments with artificial and real-world data streams were performed to evaluate and to validate DP and CDCMS. The analysis shows that both DP and CDCMS are likely to perform similar or better than existing approaches in most cases, with CDCMS performed better than DP as it was designed to handle recurrent, abrupt, gradual, severe and mild-severe drifts. This thanks to the improved diversity-based memory managed strategy and the clustering in the model space strategy in CDCMS.

This chapter focuses on concept drifts affecting the posterior probabilities of classes $P(Y|X)$, i.e., *Real drifts*, in the context of class balanced data streams. Other types of drift could be considered as future work. Chapter 5 brings the idea proposed in this chapter to deal with real concept drift in class imbalanced data streams. This chapter is further organised as follows. Section 4.1 provides the description of DP, the experiment setup and the analysis done to compare different memory management strategies and validate DP. They have been published in Chiu and Minku (2018). Section 4.2 provides the description of CDCMS, the experiment setup and the analysis done to evaluate CDCMS. They have been published in Chiu and Minku (2022). Section 4.3 presents a summary and discussion.

4.1 Diversity-based Memory Management

This section presents the preliminary study of dealing with recurrent drifts from the memory perspective, partly answering research question 1 posed in Section 1.2.1. The following sub-research questions are derived from research question 1 are used to guide this preliminary study:

- RQ1.1.1: Can a memory management strategy based on diversity help to handle recurring concepts in comparison with other memory strategies? When and why?
- RQ1.1.2: How does our diversity-based proposed approach perform in comparison with other known approaches designed for dealing with concept drift?

A novel approach using diversity-based memory management strategy, called Diversity Pool (DP) (DP), is proposed. It aims to maintain a set of models representing distinct knowledge in the memory, maximising the chance of having relevant past knowledge to exploit when recurrent drifts happen. We performed experiments to compare the proposed

approach with its variations using other four types of memory management strategies, in order to investigate when and why a diversity-based memory management strategy is advantageous and detrimental. We then compared the proposed approach against five existing approaches to validate its performance.

In an overview, the analysis of experiment results shows that, the proposed diversity-based memory management strategy helped to perform similar or better than using other memory management strategies in most cases, especially in dealing with recurrent drifts. The analysis also shows that the proposed approach usually performed similar or better that compared approaches in dealing with recurrent drifts. Yet, the proposed approach could be detrimental when the data stream presents severe gradual drifts where intermediate concepts could form.

This section is further organised as follows. Sections 4.1.1 presents the details of the proposed approach while Sections 4.1.2.2 and 4.1.3 presents the experiment setup and the analysis of the experiment results respectively. At last, a short summary of this preliminary study is presented in Section 4.1.4.

4.1.1 Description of Diversity Pool (DP)

This section details the design of the proposed diversity-based memory management approach – DP, answering RQ1.1.1. The pseudo-code of DP is presented in Algorithm 1.

DP is a single learner which maintains a memory of past models to handle recurring concepts. Predictions are made by selecting a single representative model from the memory. Alternatively, if the approach is deemed not confident enough to select a representative model, the memory is used as a weighted majority vote ensemble to make predictions. DP also employs a drift detection method, i.e. DP is an explicit drift adaptation or active

Algorithm 1 Diversity Pool (DP)

Hyper-parameters: Buffer Size (b), Pool Size (p), Data Stream (S)

Variables: Representative Model (c_r), Buffer (B), New Model (c_n), Model Pool (P), Current Example (s_t), Drift Level ($drift_level$)

$c_r \leftarrow Create(c_n)$; $P \leftarrow P \cup c_r$; $warningZoneReset \leftarrow False$
for each $s_t \in S$ **do**
    $drift_level \leftarrow DriftDetection(c_r, s_t)$
    switch ($drift_level$)
        case $WARNING$**:**
            if $warningZoneReset$ **then**
                $c_n.resetLearning()$; $warningZoneReset \leftarrow False$
                $B.clear()$; $P.resetAllModelsWeight()$; $c_n.resetWeight()$
            end if
            $B.saveFIFO(s_t)$
            $P.updateAllModelsWeightwithExample(s_t)$
            $c_n.updateWeightwithExample(s_t)$
            $c_n.trainOnInstance(s_t)$
        end case
        case $DRIFT$**:**
            if $|P| < p$ **then**
                $P \leftarrow P \cup c_n$; $c_r \leftarrow c_n$; $Create(c_n)$
            else
                if $|B| \geq b$ **then**
                    $c_{best} \leftarrow getBestByEvaluation(P, B)$
                    $c_r \leftarrow decideRepresentative(c_{best}, c_n, B)$
                    if $c_r == c_n$ **then**
                        $P.removeByDiversity()$
                        $P \leftarrow P \cup c_n$; $Create(c_n)$
                    end if
                else
                    $B.saveFIFO(s_t)$

28: $P.updateAllModelsWeightwithExample(s_t)$
29: $c_n.updateWeightwithExample(s_t)$
30: $c_n.trainOnInstance(s_t)$
31: **end if**
32: **end if**
33: **end case**
34: **case** $NORMAL$**:**
35: $warningZoneReset \leftarrow True$
36: **end case**
37: **end switch**
38: $c_r.trainOnInstance(s_t)$
39: **end for**

(Ditzler, Roveri, et al., 2015) approach. New models are added to the memory. Therefore, each model should ideally hold the knowledge of a single concept. One of the key aspects of DP is that the strategy of deciding which models to keep is based on diversity (Kuncheva and Whitaker, 2003). This prevents the pool from storing very similar models when it reaches the maximum size, thus, it increases the chance of holding a diverse set of knowledge to be ready for any concept recurs. In general, the algorithm consists of three components: 1) *Online learning with concept drift detection*, 2) *Model selection strategy*, and 3) *Diversity-based memory management strategy*. The details of these three components are detailed in Sections 4.1.1.1, 4.1.1.2 and 4.1.1.3 respectively. Section 4.1.1.4 presents the details of how DP makes predictions.

4.1.1.1 Online Learning with Concept Drift Detection

DP trains on new examples sequentially. When a new training example s_t is received, it is first fed to a concept drift detector, such as DDM (Gama, Medas, et al., 2004) or EDDM (Baena-Garc et al., 2005), for concept drift detection purpose (line 3, Algorithm 1). When

the drift detector reports a "WARNING" alarm (line 5, Algorithm 1), DP uses a sliding window B to store incoming examples (line 10, Algorithm 1). Meanwhile, a new model c_n is created and starts training on new examples alongside with the current representative model c_r (line 13, Algorithm 1). Note that the drift level could go back to "NORMAL" without triggering the "DRIFT" alarm (line 35, Algorithm 1). In this case, the new model is reset upon the drift level becomes "WARNING" again (line 6-9, Algorithm 1).

Upon the drift detector reports a "DRIFT" alarm (line 15, Algorithm 1), DP will decide whether to use the new model c_n or to retrieve a model from the memory as representative for learning and making predictions (line 21, Algorithm 1). The new model will always be used as the representative when the maximum size of the pool is not reached. One may question that there could be a more appropriate model in the pool than the new model to be the representative even though the pool has not reached the predefined maximum size, thus DP should check the pool before adopting the new model as representative. Indeed, this is likely to happen in real-world application. However, this work focuses on evaluating the proposed diversity memory management strategy. The current design of always using a new model to be the representative when the pool is not full allows the proposed approach to fill up the pool quickly, thus triggering the memory management strategy. Hence, we can evaluate it. While in application, DP should be modified to always check the pool even though the pool is not full yet. The current design is only for experiment and evaluation purposes.

The details of the model selection strategy and the diversity-based memory management strategy are explained in Sections 4.1.1.2 and 4.1.1.3 respectively. Lastly, the representative model c_r trains on the example s_t (line 38, Algorithm 1).

4.1.1.2 Model Selection Strategy

Upon the "DRIFT" alarm, DP uses the new model as representative and stores it into the memory if the maximum size of the pool is not reached (line 16-17, Algorithm 1). Otherwise, it evaluates all models in the memory and retrieve the best one (c_{best}) based on their accuracy on sliding window B (line 20, Algorithm 1). DP then compares the predictive performance of c_{best} and the model c_n (line 21, Algorithm 1). If the accuracy of the new model c_n is within a 95% confidence interval of accuracy of c_{best}, this means that c_n performed statistically the same as c_{best}. DP deems that a recurrent drift is identified, thus recovering c_{best} as the new representative model. Otherwise, the new model is used as the representative model and stored into the memory. At the same time, a model is discarded from the memory by diversity (line 22-24, Algorithm 1). This strategy is detailed in Section 4.1.1.3

This model selection strategy is conservative[1] but it enables us to focus on analysing the effectiveness of the diversity-based memory management strategy, answering RQ1.1.1. In particular, if the diversity-based memory management strategy leads to a worse predictive performance than other strategies, we can then be confident to deduce that it is caused by the poor behaviour of the proposed strategy itself, rather than the poor behaviour of the model selection strategy.

Besides, it is worth to note that concept drifts could complete in fewer time steps than the size of the sliding window B (line 19, Algorithm 1). In this case, DP uses the whole memory including the new model c_n as a weighted majority vote ensemble, where the weight of each model is calculated based on their average accuracy to the examples

[1]We performed additional experiments with a less conservative strategy which compare the time decay testing accuracy of c_{best} on the examples during the warning period against the prequential accuracy of the new model c_n. We used the result of these experiments to complement out analysis and will explicitly mention it when we do so.

during the warning period (line 11-12, 28-29, Algorithm 1). This is a moderate strategy to avoid drastic drop in predictive performance right after concept drift as there is not enough examples from the post-drift concept for DP to decide which past model to recover or to use the new model as the representative.

DP keeps storing the incoming examples into the sliding window B until B reaches the maximum size (line 27, Algorithm 1), uses them to weight the models in the memory and the new model c_n (line 11-12, 28-29, Algorithm 1) and train the new model (line 30, Algorithm 1). When the buffer B has reached the predefined maximum size b, DP is assumed to be confident enough to select a single model from the memory to be the representative mode, using the strategy described previously.

4.1.1.3 Diversity-Based Memory Management Strategy

DP is restricted to have a finite memory, which is a practical consideration for real-world application with time and memory constraints and / or data come in high speed. Thus, a strategy is required to decide which model to remove when the memory reaches the maximum size as a new model needs to be added. We refer to this mechanism as "memory management strategy". The memory management strategy used in DP simulates all possible diversity levels of the memory after removing each model. Afterwards, DP removes the model that leads to the memory state with the highest level of diversity. If there are two states with the same level of diversity, DP deletes the older model. To measure the diversity level of a set of models, any existing diversity measure from the literature can be used, as long as it is designed to measure the diversity level of more than two models. For instance, averaged Q-Statistics (Yule, 1900) for a set of models and Entropy Measure (Kuncheva and Whitaker, 2003) were considered in this work as they are well-studied in the literature. The time complexity of this simulation depends on the memory size and the diversity measure.

Therefore, applications with strict time and memory constraints should limit the memory with a small number of models and/or use a diversity measure that computes quickly. In order to the reduce the overall computational time, multi-threading techniques can also be used to perform multiple simulations concurrently. The level of diversity is calculated based on the sliding window B explained in Section 4.1.1.2. This strategy could potentially be further improved by updating the diversity level whenever a new example arrives, thus, saving memory to store examples in the sliding windows B.

4.1.1.4 Make Prediction on New Examples

This section presents the mechanism of DP making prediction on a new example. Algorithm 2 presents the psuedocode of this mechanism.

DP is a single learner approach. Hence, most of time, it makes predictions by delegating to the current representative model c_r (line 12, Algorithm 2). However, Section 4.1.1.2 also discusses that the drift could complete in fewer time steps than the size of the sliding window B, which results in B having few examples than the predefined size b. In this case, DP is deemed not confident in deciding to recover a past model from the pool P or to use the new model c_n as the representative. DP, thus, uses the whole pool including the new model c_n as a weighted majority ensemble to make prediction until B receive b examples, which is a moderate strategy to avoid drastic drop in predictive performance right after the drift (line 2-10, Algorithm 2). It is worth to note that the sliding window B is not always full as it is emptied upon the first "WARNING" alarm is given by the drift detector (line 8, Algorithm 1).

When DP uses the whole pool P including the new model c_n as an weighted majority ensemble, it first requires each model in P and the new model c_n to make a prediction to s_t. It then scales the prediction of the model by its weight and sums up the votes (line 6, 9,

Algorithm 2 DP Make Predictions

Hyper-parameters: Buffer Size (b), Pool Size (p), Data Stream (S)

Variables: Representative Model (c_r), Buffer (B), New Model (c_n), Model Pool (P), past model in P (c_p) current example (s_t)

1: **function** MAKEPREDICTION(s_t)
2: **if** $B.numberOfExamples() < b$ **then** ▷ "DRIFT" alarm is given before B receives enough examples
3: $combinedVote \leftarrow double[]\{0.0, 0.0\}$ ▷ Only two values because of binary classification
4: **for** each $c_p \in P$ **do**
5: $vote \leftarrow c_p.makePredictionOn(s_t)$ ▷ *vote* in either 0 or 1 for binary classification
6: $combinedVote[vote] += vote \times c_p.getWeight()$ ▷ Weight from line 11, 28, Alg. 1
7: **end for**
8: $vote \leftarrow c_n.makePredictionOn(s_t)$
9: $combinedVote[vote] += vote \times c_n.getWeight()$ ▷ Weight from line 12, 29, Alg. 1
10: **Return** $combinedVote.indexWithMaxValue()$
11: **else**
12: **Return** $c_r.makePrediction(s_t)$
13: **end if**
14: **end function**

Algorithm 2). The weight of the models is their average accuracy to the examples since the first "WARNING" alarm was given (line 11-12, 28-29, Algorithm 1). Finally, based on the votes scaled and accumulated, DP returns the class label that with the highest vote as its final prediction (line 10, Algorithm 2).

4.1.2 Design of the Experiment

This section presents the design of the experiment to evaluate the novel diversity-based memory management strategy and the proposed approach – DP, as a whole, partly answering research question 1 posed in Section 1.2.1. The proposed approach was first evaluated using four different memory management strategies, including the novel diversity-based memory

management strategy. This is to investigate when and why the novel diversity-based memory management strategy can help to deal with concept drift. The proposed approach using the novel diversity-based memory management strategy was then compared against five existing approaches. Massive Online Analysis (MOA) (MOA) (Bifet, Holmes, et al., 2010) was chosen to be the experiment platform. Section 4.1.2.1 presents the data streams used in the experiments. Section 4.1.2.2 presents the setup, which includes the procedure of hyper-parameter tuning and the experiments.

4.1.2.1 Data Streams

Table 4.1: Summary of Data Streams

Data Stream	Concept Drift Sequence	Width of Drift	#Examples	Max. #Models in Pool
Agrawal1	f2→f5→f3→f6→f1→f4→f1→f4→f1→f4	20,000	2M	4
Agrawal2	f1→f2→f1→f3→f1	20,000	1M	3
Agrawal3	f2→f6→f3→f6→f2	20,000	1M	3
Agrawal4	f4→f2→f1→f3→f4	20,000	1M	3
Agrawal5	f9→f8→f10→f7→f8→f7→f8→f7	1	1M	3
Agrawal6	f1→f3→f6→f5→f4	20,000	1.6M	3
Agrawal7	f7→f8→f9→f10	1	800k	3
Agrawal8	f2→f5→f4→f6	20,000	800k	3
Agrawal9	f1→f8→f3→f10→f9	1	1M	3
SEA1	f3→f1→f2→f4→f3	20,000	1M	4
SEA2	f5→f3→f1→f2→f4	20,000	1M	3
SEA3	f5→f1→f4→f3→f2	1	1M	3

- Coloured rows (lime / light grey) represent that the stream contains recurring concepts.
- *fn* represents the n-th function of the generator, i.e., f1 in Agrawal1 refers to the first function of Agrawal generator.
- Central position of the all drift is at every 200k time steps.

Table 4.1 presents a summery of artificial data streams used in this study. As this is

a preliminary study investigating the effectiveness of the proposed diversity-based memory management strategy, we only performed experiments with artificial data streams, which allow us to gain an in-depth understanding of when and why the proposed strategy is advantageous or is detrimental.

Concept drifts in each streams happen at every 200k examples. Abrupt drifts only happen in one time step while gradual drifts have a width (transitional period) of 20k time steps. i.e. there are 10k time steps before and after the central drift point respectively. As for the type of data streams, we have chosen Agrawal (Agrawal, Imielinski, and Swami, 1993) and SEA (Street and Y. Kim, 2001) generators. The Agrawal streams are binary classification problems with nine numerical input attributes that simulate the status of a stream of applications. The classification task is to distinguish which group does a given pseudo application belongs to. i.e. it simulates a real-world application. The concept function used to compose the data streams in Table 4.1 were proposed by Agrawal, Imielinski, and Swami (1993), which is also available in MOA (Bifet, Holmes, et al., 2010). The SEA streams are binary classification problems with three numerical input attributes, where the last of them is irrelevant. The SEA concepts are hyper-plane concept and the classification task is to distinguish a given data point is below or above the hyper-plane. The first to fourth concept functions (f1-f4) used to compose the data streams in Table 4.1 were proposed by Street and Y. Kim (2001), which use thresholds of 8, 9, 7, and 9.5 respectively. We have added an additional SEA concept function with threshold of 4, as the fifth function (f5), to simulate a stricter hyper-plane concept. Concept drifts were simulated by connecting two data streams with different concepts using a sigmoid function to decide the probability of examples coming from the pre and post drift concepts during the transitional period (Bifet, Holmes, et al., 2010).

4.1.2.2 Experiment Setup

This section presents the experiment setup for the comparative studies, which includes the procedure of hyper-parameter tuning and the experiments with artificial data streams. To answer RQ1.1.1, we compared four variations of the predictive performance of the proposed approach using four different memory management strategies (including the proposed diversity-based strategy). We denote these variations as Diversity Pool (DP), First in First Out (FIFO) Pool, Least Recently Used (LRU) Pool, and Elitism Pool. These memory strategies and their memory management idea are as follows.

- *Diversity*: maintain a diverse set of models in the memory, as explained in Section 4.1.1.
- *First in First Out (FIFO)*: remove oldest model.
- *Least Recently Used (LRU)*: remove the least recently used model.
- *Elitism*: remove the least performing model over the sliding window B.

To answer RQ1.1.2, we compared DP using the proposed diversity-based memory management strategy against five existing approaches. The compared approaches and their reasons of choice are as follows.

- *HTNB* (Domingos and Hulten, 2000): to analyse how the proposed approach compares to its base learner.
- *DDM* (Gama, Medas, et al., 2004): to see if the proposed approach improves over an approach that resets the system upon drift detection.
- *RCD* (Gonçalves Jr and Barros, 2013): to see how the proposed approach compares to another online learning approach designed to handle recurring concepts.

- *OAUE* (Brzeziński and Stefanowski, 2014) and *DWM* (Kolter and Maloof, 2003): to see how the proposed approach compares to ensemble approaches for non-stationary environments. Although these approaches were not explicitly intended to handle recurring concepts, ensembles can be seen as pools of models.

Table 4.2: Options for Hyper-parameters

Hyper-parameter	DP	RCD	OAUE	DWM
Base Learner	Hoeffding Tree with Naïve Bayes at leaves (HTNB)			
Memory / Ensemble Size	{3, 4} (depends on data streams, see Table 4.1)			
Drift Detector (*ddm*)	{DDM, EDDM}			
Warning Period Buffer size (b)	{200, 400, 600, 800, 1000}	-		
Size of Buffer associated to model (b_{model})	-	{200, 400, 600, 800, 1000}	-	
Evaluation Frequency (e)	-	{200, 400, 600, 800, 1000}	-	
Window Size (w)[#]	-		{200, 400, 600, 800, 1000}	-
Period between Ensemble Changes (c)	-			{200, 400, 600, 800, 1000}
Diversity Measure (d)	{Entropy Measure, Q-Statistics}	-		
k-Nearest Neighbour (k)	-	{1, 3, 5, 7}	-	
p-value Threshold (p)	-	{0.01, 0.05}	-	

[-] Grey rows denote pre-chosen hyper-parameter values.

To tune the hyper-parameters of the approaches, we created two additional artificial

Table 4.3: Chosen Hyper-parameters

Approach	Chosen Hyper-parameters
Agrawal	
DP	ddm=DDM, b=600, d=Entropy Measure
RCD	ddm=EDDM, b_{model}=800, e=1000, k=7, p=0.01
OAUE	w=1000
DWM	c=800
SEA	
DP	ddm=DDM, b=400, d=Entropy Measure
RCD	ddm=DDM, b_{model}=400, e=600, k=1, p=0.01
OAUE	w=1000
DWM	c=200

data streams (one for Agrawal and one for SEA), consisting of a random sequence of concepts, drift locations and widths. We denote these two data streams as "hyper-parameter tuning streams". The set of values that leads to the highest average prequential accuracy across the hyper-parameter tuning streams were then used in the experiments with the corresponding data stream type. The options for the hyper-parameters are presented in Table 4.2 while the chosen hyper-parameters are presented in Table 4.3. Other hyper-parameters that are not shown in Tables 4.2 and 4.3 were kept in MOA's default values. All approaches used Hoeffding Tree with Naïve Bayes at Leaves (HTNB) (Domingos and Hulten, 2000) as the base learner, with MOA default of hyper-parameter values. Also, we have limited the maximum number of models in the memory to increase the chance of triggering the memory strategies, enabling us to evaluate their effectiveness. Note that, in the comparison of different memory management strategies, we applied the same set of hyper-parameter values to all variations, ensuring that the only difference was the memory management strategy.

As for the experiments, the accuracy was measured prequentially, i.e., each example is first used to test the approach before training it (Bifet, Holmes, et al., 2010). A fading

factor of 0.999 was used to make past examples less important to the current accuracy (Gama, Rqauel Sebastião, and Rodrigues, 2013). The prequential performance of a given approach D is calculated as follow.

$$preq_perform_t(D) = \begin{cases} preq_perform_t^{ex}(D), & \text{if } t = f \\ \frac{preq_perform_t^{ex}(D) + \alpha \times preq_perform_{t-1}(D) \times (t-f)}{t-f+1}, & \text{otherwise} \end{cases} \tag{4.1}$$

where t is the current time step, $preq_perform_t^{ex}$ is 0 if the prediction to the current training example ex before learning is wrong and 1 if it is correct, f is the first time step used in the calculation, and α is the time decaying factor. Hence, $\alpha = 0.999$.

To compare the prequential accuracy of the approaches on each data stream, Friedman test with a level of significance of 0.05 was performed to identify which specific data streams that DP was advantageous or detrimental. In the Friedman test, each group corresponds to an approach while each observation within a group corresponds to the prequential accuracy sampled at every 1k time steps. Nemenyi post-hoc tests were also performed to identify which specific pairs of groups (approaches) have statistical significant difference, using the same level of significance of 0.05.

Based on the comparison, a few representative cases were chosen to analyse the approaches' behaviour in details. This provides an in-depth understanding to the experiment results, answering both RQ1.1.1 and RQ1.1.2. Other cases can be explained similarly.

4.1.3 Analysis

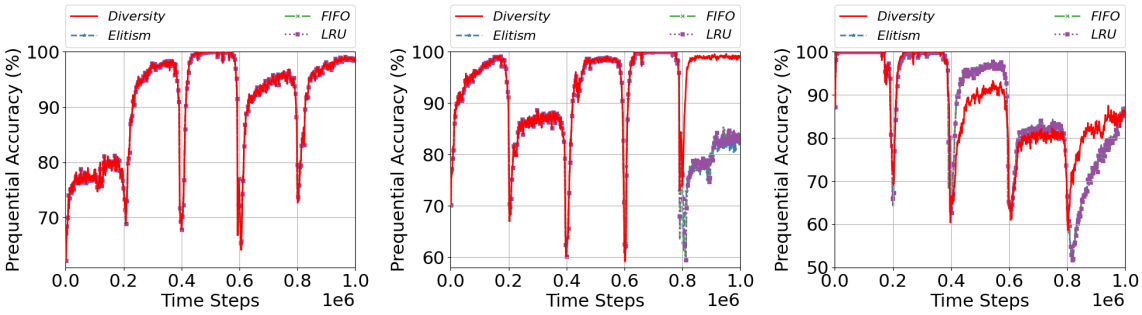
This section presents the analysis done to evaluate the proposed approach – DP. Section 4.1.3.1 compares the effectiveness of different memory management strategies. Section 4.1.3.2

compares DP against five existing approaches in twelve artificial data streams.

4.1.3.1 Comparison of Memory Strategies

Table 4.4 presents the means, standard deviations, and results of Friedman and Nemenyi tests to compare the prequential accuracy of the proposed approach, using four different memory strategies, on each data stream. It shows that the diversity memory strategy obtained similar or better prequential accuracy in most cases, except for Agrawal5 stream (in comparison to LRU). The elitism strategy was frequently worse than the diversity strategy, being the least recommended strategy.

Figure 4.1 presents the prequential accuracy of the proposed approach, using four different memory strategies, throughout Agrawal3, Agrawal4, and Agrawal6 streams. These streams were chosen to be representative cases where the diversity memory strategy performed similarly most of the time, better and worse for some different periods of time with respect to other memory management strategies. Figures of other cases are available in the Appendix B.



(a) Agrawal3 (b) Agrawal4 (c) Agrawal6

Figure 4.1: Prequential Accuracy Using Different Memory Strategies over Artificial Data Streams (Agrawal3-4,6)

Table 4.4: Memory Strategy Comparison – Mean, Standard Deviations, Friedman Test and Nememyi Test Results

Data Stream	Mean of Prequential Accuracy with Fading Factor 0.999 Across the Data Stream (Standard Deviation)				Friedman Test p-value	Nememyi Post Hoc Test p-value		
	Diversity	FIFO	LRU	Elitism		Diversity vs FIFO	Diversity vs LRU	Diversity vs Elitism
Agrawal1	84.359% (11.302%)	84.094% (11.322%)	84.094% (11.322%)	**84.362% (11.300%)**	7.522E-07	0.541	0.541	0.303
Agrawal2	**97.233% (6.548%)**	97.017% (7.555%)	97.017% (7.555%)	94.994% (8.367%)	<2.2E-16	0.44	0.44	5.5E-14
Agrawal3	90.581% (9.494%)	90.581% (9.494%)	90.581% (9.494%)	90.581% (9.494%)	1.00	-	-	-
Agrawal4	**93.468% (8.021%)**	89.637% (9.569%)	89.637% (9.569%)	89.689% (9.436%)	<2.2E-16	3.0E-14	3.5E-14	1.6E-13
Agrawal5	98.335% (1.768%)	98.501% (1.612%)	98.500% (1.609%)	**98.504% (1.613%)**	<2.2E-16	1.00	1.6E-09	0.82
Agrawal6	**87.912% (10.475%)**	87.792% (12.569%)	87.792% (12.569%)	87.794% (12.566%)	4.982E-06	0.0474	0.0474	0.0028
Agrawal7	98.320% (1.820%)	98.256% (1.780%)	**98.320% (1.813%)**	98.268% (1.783%)	<2.2E-16	1.5E-06	0.9788	0.0010
Agrawal8	**80.143% (8.694%)**	**80.143% (8.693%)**	**80.143% (8.693%)**	79.827% (8.539%)	<2.2E-16	1.00	1.00	4.4E-0.7
Agrawal9	98.951% (2.213%)	98.593% (2.217%)	**98.953% (2.214%)**	98.716% (2.443%)	<2.2E-16	0.731	0.077	2.2E-10
SEA1	98.297% (2.194%)	98.297% (2.194%)	98.297% (2.194%)	98.297% (2.194%)	<1.00	-	-	-
SEA2	**98.961% (1.129%)**	**98.961% (1.129%)**	**98.961% (1.129%)**	98.908% (1.185%)	3.683E-11	0.8861	0.8861	3.4E-05
SEA3	98.945% (1.204%)	99.027% (1.040%)	99.028% (1.056%)	**99.053% (1.003%)**	0.4422	-	-	-

Cells with bold text represent the best performance in the data stream.

Highlighted data streams refer to have recurrent drifts.

Coloured Nemenyi p-value cells represent that there is statistical significant difference (p-value $\leq$ 0.05) and the diversity-based memory management strategy performed better (lime or light grey) / worse (orange or dark grey) based on the mean and standard deviation of its prequential accuracy. Nemenyi p-value cells in white represent no statistical significant difference.

In general, all variations performed similarly at the early stage of all streams because the their memory had not reached the predefined maximum size, which means the memory strategies were not yet triggered. Once the maximum memory size was reached, they started to perform differently, in Agrawal4 and Agrawal6 streams. Even though the diversity strategy did not typically present large accuracy improvements with respect to other strategies, it was the most consistent one among the best strategies in all data streams. This means that, in practice, diversity may be a safer strategy to adopt when the underlying characteristics of the data stream are unknown in advance.

In order to provide an in-depth understanding to these results, we discuss the representative cases in details, where the diversity memory strategy performed better, worse and similarly to the best memory management strategy.

4.1.3.1.1 Cases where the diversity strategy performed better: Figure 4.1(b) shows that the diversity memory strategy performed the best in Agrawal4 stream. In particular, it recovered the best in prequential accuracy at the last drift, which is a recurrent drift leading back to the first concept of the stream – Agrawal-f4.

According to Table A.1, Agrawal-f1 and Agrawal-f4 are very similar but quite distinct from Agrawal-f1 and Agrawal-f3. Based on the concept sequence of Agrawal4 stream (Table 4.1), the knowledge of Agrawal-f4 appeared at the beginning of the stream should have been forgotten by the memory strategies of FIFO, LRU and Elitism when it recurred at 800k time steps. The knowledge of Agrawal-f2 was also forgotten due to false-positive drift detection. In contrast, the diversity strategy still held the knowledge of the Agrawal-f4 despite the false-positive drift detection (see Table 4.5). Hence, it can exploited the previous knowledge to adapt to this recurrent drift. Therefore, it helped to recover the predictive performance better than other memory management strategies, confirming that the diversity memory management strategy can, indeed, maximise the number of concept held in the memory to

handle recurrent drifts.

Table 4.5: State of Diversity Pool P in Agrawal4 stream

Time Steps	P[0]	P[1]	P[2]	Actual Concept
0 - 199,810	0(f4)	-	-	f4
199,810 - 201,919	0(f4)	1(f2)	-	f2
201,919 - 394,594	0(f4)	1(f2)	2(f2)	f2
394,594 - 594,128	0(f4)	1(f2)	3(f1)	f1
594,128 - 594,386	0(f4)	3(f1)	4(f1)	f1
594,386 - 594,925	0(f4)	3(f1)	4(f1)	f1
594,925 - 600,859	0(f4)	3(f1)	5(f1)	f1
600,859 - 601,431	0(f4)	3(f1)	6(f3)	f3
601,431 - 601,823	0(f4)	3(f1)	6(f3)	f3
601,823 - 788,649	0(f4)	6(f3)	7(f3)	f3
788,649 - 799,364	0(f4)	6(f3)	8(f3)	f3
799,364 - 1,000,000	0(f4)	6(f3)	8(f3)	f4

- Format: $\{number_i\}$(f$\{number_j\}$), where i refers to the index of the models created and j refers to the concept that was active when they were created. The coloured cells (lime or light grey) indicate the representative model in the time period analysed. When all models from the pool P are highlighted, this means that a weighted majority ensemble was used, rather than a single representative model.

4.1.3.1.2 Cases where the diversity strategy performed worse: Figure 4.1(c) shows that diversity strategy has a significant under-performance for a long period (around 400k-600k time steps) in Agrawal6 stream. According to Table 4.1, Agrawal6 stream has a concept drift occurs at 400k time steps, with a transitional period of 10k time steps before and after it. Table 4.1 and Table A.1 show that the concepts before and after this drift are Agrawal-f3 and Agrawal-f6 which are quite different. A closer look into the behaviour reveals that a concept drift detection was given at 396,264 time steps when using the diversity strategy while the concept drift detection for the other strategies was given at 400,928 time steps.

All strategies created a new model as representative after this drift detection. The early drift detection of the diversity strategy means that its new model was trained with around 3.7k examples which likely belong to the pre-drift concept. Thus, the subsequent predictive performance was detrimental affected (400k examples to 600k examples). In contrast, the new model of other strategies were created after the drift point, meaning that they learnt from data more likely to belong to the post-drift concept. Although we cannot see the direct cause of the under-performance was the diversity memory strategy, we have such deduction because the memory management strategy is the only difference between these variations. we further investigate why the diversity memory strategy indirectly led to poor subsequent predictive performance.

Through a closer look at Table 4.6, which presents the memory states of the diversity memory strategy throughout the Agrawal6 stream, and Figure 4.1(c), we can see that the diversity memory strategy successfully exploited past models learnt during the first drift to achieve high and stable predictive performance for a short period during the second concept. This stable accuracy made the concept drift detector being sensitive to any change to the predictive performance. Thus, several false-positive drift detection were given during the second concept (200k-400k time steps). It also led to an early drift detection to the third drift (400k-600k time steps). This early drift detection, eventually, resulted in poor predictive performance during the third concept, as explained in the previous paragraph. This shows that, even though diversity can help as expected and demonstrated in Agrawal4 stream, the stable predictive performance it achieved can, sometimes, lead to a subsequent poor predictive performance, as it could lead to the drift detector being sensitive to change in predictive performance of the representative model.

4.1.3.1.3 Cases where the diversity strategy performed similarly: Table 4.4 shows that the predictive performance of using the diversity memory management strategy did not

Table 4.6: State of Diversity Pool P in Agrawal6 stream

Time Steps	P[0]	P[1]	P[2]	Actual Concept
0 - 171,261	0(f1)	-	-	f1
171,261 - 171,682	0(f1)	1(f1)	-	f1
171,682 - 175,383	0(f1)	1(f1)	2(f1)	f1
175,383 - 175,786	0(f1)	1(f1)	2(f1)	f1
175,786 - 190,502	0(f1)	2(f1)	3(f1)	f1
190,502 - 190,615	0(f1)	2(f1)	3(f1)	f1
190,615 - 193,250	2(f1)	3(f1)	4(f1)	f1
193,250 - 193,453	2(f1)	3(f1)	4(f1)	f1
193,453 - 197,003	2(f1)	4(f1)	5(f1)	f1
197,003 - 197,347	2(f1)	4(f1)	5(f1)	f1
197,347 - 201,681	2(f1)	4(f1)	5(f1)	f1
201,681 - 387,518	2(f1)	4(f3)	5(f1)	f3
387,518 - 394,373	4(f3)	5(f1)	6(f3)	f3
394,373 - 394,485	5(f1)	6(f3)	7(f3)	f3
394,485 - 395,046	5(f1)	6(f3)	7(f3)	f3
395,046 - 395,771	5(f1)	6(f3)	8(f3)	f3
395,771 - 396,264	5(f1)	6(f3)	8(f3)	f3
396,264 - 597,296	5(f1)	8(f3)	9(f6)	f6
597,296 - 798,344	5(f1)	9(f6)	10(f5)	f5
798,344 - 1,000,000	5(f1)	9(f6)	11(f4)	f4

- Format: $\{number_i\}$(f$\{number_j\}$), where i refers to the index of the models created and j refers to the concept that was active when they were created. The coloured cells (lime or light grey) indicate the representative model in the time period analysed. When all models from the pool P are highlighted, this means that a weighted majority ensemble was used, rather than a single representative model.

have any statistical significant difference with respect to FIFO and LRU in most cases.

Agrawal3 stream is one of the cases. We can see from Figure 4.1(a) that the difference

in prequential accuracy between diversity and elitism strategies was not noticeable. This is an expected result for cases which do not present any recurrent drift. As for cases presenting recurrent drift, such phenomenon of similar predictive performance was because they all used the newly created model to be the representative model in adapting to recurrent drifts, rather than recovering a past model from the memory. Thus, all the variations performed similarly despite the knowledge held in their memory were different. This is mainly caused by the model selection strategy used by the proposed approach is conservative. Yet, it allows us to focus on analysing the behaviour of the memory management strategies themselves, rather than affected by the effectiveness of the model selection strategy.

We also performed additional experiments with a less conservative model selection strategy. However, it did not make any noticeable improvements to the prequential accuracy of the proposed approach. A potential reason is that the less conservative strategy recovered models that are not similar enough to the post-drift concept, hindering the approach's subsequent prequential accuracy. Thus, the model selection strategy is influential in adapting to concept drift but the memory management strategy is also important as it determines which past model to keep for future use. In Section 4.2, we devised a novel model selection strategy and incorporated it with the diversity-based memory management strategy to become a diversity-based framework, aiming to handle multiple types of concept drift.

4.1.3.2 Comparison against Existing Approaches

Table 4.7 presents the means, standard deviations, and results of the Friedman and Nemenyi tests to compare the prequential accuracy of DP against existing approaches. Agrawal1, Agrawal5, and SEA1 streams were chosen as the representative cases. Their performance over the streams are presented in Figure 4.2.

Table 4.7 shows that DP performed better than its base learner HTNB, which is not

Table 4.7: Comparison against Existing Approaches – Mean, Standard Deviations, Friedman Test and Nemenyi Test Results

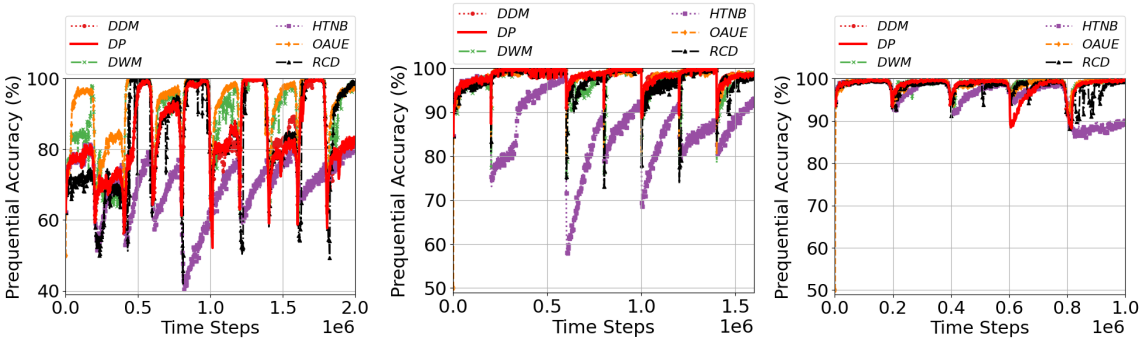
Data Stream	Mean of Prequential Accuracy with Fading Factor 0.999 Across the Data Stream (Standard Deviation)						Friedman Test p-value	Nememyi Post Hoc Test p-value				
	DP	HTNB	DDM	RCD	OAUE	DWM		DP vs HTNB	DP vs DDM	DP vs RCD	DP vs OAUE	DP vs DWM
Agrawal1	84.359% (11.301%)	69.170% (8.612%)	**84.581% (11.211%)**	83.906% (14.391%)	92.988% (8.926%)	89.871% (11.374%)	<2.2E-16	<2.2E-16	0.126	9.3E-08	<2.2E-16	<2.2E-16
Agrawal2	**97.233% (6.548%)**	82.134% (13.602%)	96.383% (7.611%)	91.629% (13.248%)	95.418% (9.243%)	92.302% (12.123%)	<2.2E-16	<2.2E-16	6.6E-09	<2.2E-16	0.653	5.2E-14
Agrawal3	90.581% (9.494%)	75.255% (6.792%)	88.331% (10.008%)	82.172% (14.324%)	**93.895% (7.939%)**	88.200% (9.192%)	<2.2E-16	<2.2E-16	0.0098	3.1E-14	<2.2E-16	0.9847
Agrawal4	**93.468% (8.021%)**	79.019% (12.432%)	91.423% (8.605%)	76.172% (10.693%)	93.700% (8.069%)	87.204% (10.802%)	<2.2E-16	<2.2E-16	6.1E-14	<2.2E-16	4.0E-13	<2.2E-16
Agrawal5	98.335% (1.768%)	86.611% (8.165%)	**98.484% (1.620%)**	97.210% (3.438%)	98.087% (2.421%)	97.165% (3.526%)	<2.2E-16	<2.2E-16	3.7E-08	<2.2E-16	<2.2E-16	<2.2E-16
Agrawal6	87.912% (10.475%)	72.865% (19.424%)	85.498% (12.290%)	89.844% (10.831%)	**93.283% (9.083%)**	87.358% (13.210%)	<2.2E-16	<2.2E-16	8.2E-06	7.0E-14	<2.2E-16	7.4E-11
Agrawal7	**98.320% (1.820%)**	88.097% (8.254%)	98.273% (1.795%)	96.180% (3.669%)	97.971% (2.888%)	97.130% (4.737%)	<2.2E-16	<2.2E-16	0.52	5.1E-14	5.2E-14	6.0E-14
Agrawal8	80.143% (8.694%)	72.137% (6.883%)	80.143% (8.694%)	75.968% (11.808%)	**87.771% (10.676%)**	83.560% (12.151%)	<2.2E-16	<2.2E-16	1.00	<2.2E-16	<2.2E-16	5.9E-14
Agrawal9	**98.951% (2.213%)**	91.103% (10.224%)	98.971% (2.095%)	98.114% (4.091%)	98.704% (2.848%)	97.156% (6.811%)	<2.2E-16	<2.2E-16	1.00	<2.2E-16	6.6E-12	4.8E-14
SEA1	98.297% (2.194%)	95.782% (4.120%)	98.819% (1.338%)	98.067% (2.133%)	98.705% (2.103%)	**98.823% (1.275%)**	<2.2E-16	<2.2E-16	1.3E-11	7.3E-14	2.9E-14	1.1E-13
SEA2	**98.961% (1.129%)**	97.353% (2.952%)	98.931% (1.188%)	98.579% (1.327%)	98.529% (2.050%)	98.651% (1.411%)	<2.2E-16	<2.2E-16	0.30756	<2.2E-16	5.5E-14	<2.2E-16
SEA3	**98.945% (1.204%)**	95.826% (4.390%)	98.942% (1.166%)	98.593% (1.347%)	98.248% (2.317%)	98.615% (1.622%)	<2.2E-16	<2.2E-16	1.00	5.4E-14	<2.2E-16	5.0E-11

- Highlighted data streams refer to have recurrent drifts.; Cells with bold text represent the best performance in data stream; Highlighted Nemenyi p-value cells denote better (lime or light grey) / worse (orange or dark grey) prequential accuracy by the proposed approach. Nemenyi p-value cells in white represent no statistical significant difference.

prepared for learning data streams with concept drift. Therefore, it is not surprise that DP performed better than HTNB, as DP employs a drift detector to deal with concept drift explicitly.

Comparing the predictive performance of DP against DDM, Table 4.7 shows that they have no statistical significant difference in most data streams presenting no recurrent drift. This is an expected result because, when there is no recurrent drift, the diversity memory strategy should never be triggered. Thus, it performed similarly to DDM. As for data streams with recurrent drifts, DP performed better than DDM, especially in cases where there were several distinct concepts before the recurrent drift (Agrawal2-4). This is because DP maintained a diverse memory of models to be ready for handling recurrent drifts, while DDM discard past model upon drift detection. On the other hand, Table 4.7 also shows that DP performed worse than DDM in two data streams – Agrawal5 and SEA1. These are two extreme cases where concept drifts of swapping two distinct concepts near the end of the streams. The potential reason behind this is that DP may have recovered an inappropriate model in response to drift detections, which resulted in slow recovery in predictive performance. Yet, Figures 4.2(a) and 4.2(b) show that the difference between the predictive performance of DP and DDM were usually small.

In the comparison of DP against an existing approach that aimed to handle recurrent drifts – RCD, Table 4.7 shows that DP performed better in most cases. DP only performed worse than RCD in only one case that presents no recurrent drift – Agrawal6. In general, we can see that DP usually had better average prequential accuracy with smaller standard deviations than RCD, especially in data streams presenting recurrent drifts. This is an encouraging result for DP. The potential reason is the difference in memory strategy, where RCD uses FIFO, while DP uses diversity. The benefits of using diversity over FIFO have been discussed in Section 4.1.3.1. Another potential reason is that the model selection strategy used by RCD may have a stricter requirement to identify relevant past knowledge, leading



(a) Agrawal1 (b) Agrawal5 (c) SEA1

Figure 4.2: Prequential Accuracy of DP and Existing Approaches over Artificial Data Streams (Agrawal1,5 and SEA1)

RCD frequently failed to recognise recurrent drifts. It is also worth to mention that RCD employed EDDM as the drift detector while DP employed DDM. Thus, their reaction to the data stream may also affected by their drift detector. However, we have performed several additional experiments on RCD using DDM. The results show that using DDM in RCD would leads to even worse predictive performance. Therefore, we can deduce that the drift detector was not the main reason for the poor performance of RCD.

When comparing DP against ensemble methods that implicitly handle concept drifts – OAUE, DDM, Table 4.7 shows that DP performed worse than OAUE and DWM in Agrawal1 and Agrawal8 streams. We deduce that this is likely caused by the conservative model selection strategy of DP because Table 4.7 shows the predictive performance of DP and DDM had no statistical significant difference in these data streams, suggesting that the model selection strategy may fail to recover relevant past models from the memory. The diversity memory strategy, thus, could not help to deal with recurrent drifts. Besides, ensemble approaches usually perform better than single base learner approaches.

4.1.4 Summary of Diversity-based Memory Management Study

This preliminary study of dealing with recurrent drifts from the memory perspective showes that maintaining a memory of past models by a diversity-based strategy can improve prequential accuracy of a data stream learner, especially when the data stream presents recurrent drifts (RQ1.1.1). However, the relationship between the diversity-based memory management strategy and the overall predictive performance of the approach is more complex than one may initially have thought, because it does not directly improve the predictive performance of the approach. It helps to maximise the number of different concepts in the memory, such that when recurrent drift occurs and is identified, the approach can have a variety of concepts to exploit.

In addition, the analysis also shows that a stable predictive performance achieved through the diversity-based memory strategy can, sometimes, hinder the subsequent predictive performance, as it can lead to the drift detector being sensitive to small change in subsequent predictive performance.

Moreover, this preliminary study shows that the proposed approach of using diversity as memory management strategy performed particularly better than existing approaches in data streams with recurrent drifts (RQ1.1.2). In general, it performed better than RCD, which was prepared for recurrent drifts. It also performed similar to or better than DDM, which discards past knowledge upon drift detection. It performed sometimes better and sometimes worse than OAUE and DWM.

4.2 Diversity-based Memory Recovery

The preliminary study presented in Section 4.1 shows that the idea of using diversity to manage a set of past models in the memory can, indeed, be advantageous in dealing with recurrent drifts. Yet, the same study also shows that the relationship between the overall predictive performance of the approach and the diversity-based memory management strategy is not straightforward. This is because the model selection strategy also plays a key role in contributing to the predictive performance of the approach. For instance, the advantage of using the diversity-based memory management strategy could not be manifested when the model selection strategy does not manage to recover correct relevant knowledge.

In this section, we continue to answer research question 1 posed in Section 1.2.1 by proposing a novel diversity-based framework that aims to handle recurrent, abrupt, gradual, severe and mild-severe drifts. In particular, we propose a novel clustering based method, called the clustering in the model space strategy, to correctly identify recurrent drifts, recover relevant past knowledge and prevent catastrophic drop in predictive performance upon concept drift. The proposed framework is an ensemble approach, where the main ensemble periodically replaces the least performing base learner with a new one to implicitly handle gradual drifts.

The following sub-research questions are derived from research question 1 to guide this study:

- RQ1.2.1: How can diversity be used as memory strategies to deal with multiple types of concept drift, including recurrent and non-recurrent drifts?
- RQ1.2.1: How effective is this framework in dealing with multiple types of concept drift compared to existing approaches?

- RQ1.2.3: Why is this framework successful or unsuccessful in dealing with multiple types of concept drift compared to existing approaches?

The analysis of the experiment results shows that the proposed diversity-based framework usually performed similar or better than existing approaches in both artificial and real-world data streams. It is particularly good at dealing with abrupt and recurrent drifts, in terms of predictive performance recovery speed and the overall prequential accuracy. It is slightly deficient in dealing with extremely severe gradual recurrent drifts.

This section is further branched as follows. Section 4.2.1 presents the detailed description of the proposed diversity framework. Section 4.2.2 provides the experiment setup. Section 4.2.3 presents the analysis of the experiment results. Section 4.2.4 presents the time complexity analysis of the proposed framework with empirical results. Finally, a short summary to this study is presented in Section 4.2.5.

4.2.1 Description of Concept Drift Handling Based on Clustering in the Model Space (CDCMS)

This section details the proposed framework to answer research question 1, in particular, RQ1.2.1. This proposed diversity-based framework is called Concept Drift Handling Based on Clustering in the Model Space (CDCMS). The main novelty is the diversity memory strategy, which includes the clustering in the model space strategy and the memory management strategy aimed at dealing with multiple types of concept drift. The details of CDCMS is presented in Algorithm 3 and its general working mechanism is described as follows.

CDCMS uses a *memory* R of size $e \times r$ to store past models representing different concepts, where e is the predefined maximum size of the ensembles and r is the memory size multiplier. A main ensemble E_{nl} consisting of a new model is initialised to learn the current

Algorithm 3 Concept Drift Handling Based on Clustering in the Model Space - CDCMS

Hyper-parameters: Base learner (c_{type}), Ensemble Size (e), Memory Size ($e \times r$), Window Size (b), Similarity Threshold (θ), Clustering Method ($clusterer$), Data Stream (S)

Variables: New Ensemble (E_{nl}), Old Ensemble (E_{ol}), Highly Diverse Ensemble (E_{nh}), New Model (c_n), Sliding Window (B), Memory (R), Clustering Result (C), Cluster ($cluster$)

$E_{nl} \leftarrow E_{nl}.add(createModel(c_{type}))$
$E_{ol} \leftarrow null; E_{nh} \leftarrow null; c_n \leftarrow createModel(c_{type})$
for $s_t \in S$ **do**
    $B.saveFIFO(s_t)$
    $drift_level \leftarrow DriftDetection(E_{nl}, s_t)$
    if $drift_level == NORMAL$ **then**
        if t mod $b == 0$ **then**
            if t is b time steps after a drift detection **then**
                $C \leftarrow (R \cup c_n).clusteringModels(B)$
                $cluster \leftarrow C.getCluster(c_n)$
                if cluster.size() $== 1$ **then**
                    $E_{nl}.add(c_n)$
                else
                    $E_{nl}.add(cluster.getMostTrained(e-1))$
                end if
            else if $|E_{nl}| \geq e$ **then**
                $c_{worst} \leftarrow E_{nl}.getWorstModel()$
                $c_{worst}.resetWeight()$
                $R.saveByDiversity(c_{worst})$
            end if
            $c_n \leftarrow createModel(c_{type})$; $E_{nl}.add(c_n)$
        end if
        $c_n.updateWeight(s_t); c_n.trainOnInstance(s_t)$
        $drift_level_{previous} \leftarrow NORMAL$
    else if $drift_level == DRIFT$ **then**

26: $E_{ol} \leftarrow E_{nl}$
27: $E_{nl}.resetWeight(); R.saveByDiversity(E_{nl})$
28: $E_{nl}.clear(); E_{nh} \leftarrow createEnsemble(e)$
29: **if** $drift_level_{previous} == NORMAL$ **then**
30: $c_n \leftarrow createModel(c_{type})$
31: $C \leftarrow R.clusteringModels(B)$
32: $E_{nh} \leftarrow C.getRepresentativeModels()$
33: **end if**
34: $E_{nl}.add(c_n); c_n \leftarrow createModel(c_{type})$
35: $E_{nh}.resetWeight(); E_{nl}.resetWeight()$
36: $E_{ol}.resetWeight(); B.clear()$
37: $drift_level_{previous} \leftarrow DRIFT$
38: **end if**
39: **if** $E_{ol} \neq null$ **then** $E_{ol}.updateWeight(s_t)$
40: **end if**
41: **if** $E_{nh} \neq null$ **then** $E_{nh}.updateWeight(s_t)$
42: **end if**
43: $E_{nl}.updateWeight(s_t); E_{nl}.trainOnInstance(s_t)$
44: **end for**

concept (line 1, Algorithm 3). During learning, each training example (line 3, Algorithm 3) is stored into a sliding window B (line 4, Algorithm 3), which will be used by the clustering in the model space strategy to deal with drifts later. A drift detection method is employed (line 5, Algorithm 3) to determine whether strategies to react to concept drift should be triggered. Any drift detection method can be used, e.g., DDM (Gama, Medas, et al., 2004), EDDM (Baena-Garc et al., 2005), EDDM drift detector (Bifet and Gavaldà, 2007), Wilcoxon Drift Detector (Barros, Gonzalez Hidalgo, and D. Cabral, 2018), DetectA (Escovedo et al., 2017), FHDDM (Pesaranghader and Viktor, 2016), etc.

After every b time steps (line 7, Algorithm 3), CDCMS adds a new model c_n to the main ensemble E_{nl} (line 21, Algorithm 3), as a strategy to deal with gradual drifts. Before

adding this model, CDCMS first checks whether E_{nl} has reached the predefined maximum size e (line 16, Algorithm 3). If so, the least performing model c_{worst} from E_{nl} is moved into the memory R using the diversity-based memory management strategy (line 19, Algorithm 3), explained in Section 4.2.1.3. In other time steps, c_n learns alongside with E_{nl} (line 23, Algorithm 3). Together with the strategy of creating a new model at every b time steps, they help to fill up the memory quickly so that the approach holds a good number of past models for the clustering in the model space strategy, explained in Section 4.2.1.1. Alternatively, if the current time step t is b time steps since a drift detection which is the time step triggering the clustering in the model space strategy to recover relevant past models from the memory (line 8, Algorithm 3), the representative ensemble E_{nl} is replaced by $(e-1)$ past models from the memory which hold similar concepts to new model c_n, i.e., the current underlying concept. This design aims to deal with recurrent drifts and drifts leading to a concept that is similar to a past one. That said, in the case that the post-drift concept is very novel, the new model c_n will be in a cluster with itself only. Hence, nothing will be recovered and the new ensemble will just consist of the new model c_n only.

Upon drift detection (line 25, Algorithm 3), the main ensemble E_{nl} is denominated as the old main ensemble E_{ol} (line 26, Algorithm 3). The base learners inside E_{nl} are stored into the memory, using the diversity-based memory management strategy (line 27, Algorithm 3), explained in Section 4.2.1.3. A highly diverse ensemble E_{nh} is built by taking a representative model from each cluster of models from the memory (line 32, Algorithm 3), using the clustering in the model space strategy explained in Section 4.2.1.1. Before building E_{nh}, CDCMS checks whether the previous state of the system was "NORMAL" (line 29, Algorithm 3). This is to prevent triggering the clustering method to rebuild E_{nh} in consecutive time steps caused by false-positive drift detection, which may detrimentally affect the prequential accuracy and running speed of the framework. This highly diverse ensemble is created to prevent a catastrophic drop in performance right after concept drift

At the same time, a new main ensemble E_{nl} is also created, which is initialised with a new model c_n created upon drift detection (line 34, 30, Algorithm 3).

To make predictions, CDCMS usually uses the main ensemble E_{nl} by weighted majority vote over the base learners, where the weights of the base learners are based on their prequential accuracy. During the uncertain period right after the drift detection, all three ensembles are used to make predictions by weighted majority vote until the system deems to have confidence in using E_{nl} only. Similarly, the weights of the ensembles are also based on their prequential accuracy. Section 4.2.1.2 discusses this in details.

The key aspect of CDCMS is the clustering in the model space strategy (lines 9, 31, Algorithm 3). The idea is that, ideally, models learnt from the same concept should make the same set of mistakes when making predictions to the same batch of examples. This means that clustering models by their predictions to the sliding window B, should reveal models holding the same / similar concept(s) in the same cluster. This technique plays an important role in CDCMS because it enables a convenient way to identify recurring concepts and to construct ensembles with different levels of diversity for dealing with concept drifts.

Another key aspect of CDCMS is the diversity-based memory management strategy. It determines which model to remove from the memory when the memory reaches the predefined maximum size (line 19, 27, Algorithm 3), thus maintain a diverse set of concepts in the memory. This strategy is more time-efficient than the preliminary strategy proposed in Section 4.1, and it is discussed in Section 4.2.1.5. CDCMS is further detailed in Sections 4.2.1.1, 4.2.1.2, 4.2.1.3, 4.2.1.4, and 4.2.1.5.

4.2.1.1 Clustering Models in the Diverse Memory

Clustering models in the diverse memory is crucial to CDCMS. It is a key strategy to tackle multiple types of concept drift as part of the unified framework. It can be used to form a highly diverse ensemble by picking a representative model from each cluster, preventing a drastic drop in predictive performance right after concept drift. It can also be used to identify and recovering relevant past knowledge in the memory, accelerating the predictive performance recovery form drifts. This section concentrates on the method for clustering in the model space, whereas Section 4.2.1.2 explains how the results of the clustering are used to deal with concept drift.

The clustering in the model space strategy relies on the predictions made by the models to a batch of examples. Therefore, a sliding window B is needed for this purpose. This sliding window is emptied upon drift detection to prevent it from holding examples that belong to past concepts (line 36, Algorithm 3). That said, we believe that CDCMS has certain robustness to the presence of old examples in the sliding window, because similar models should make similar mistakes on old examples.

Whenever CDCMS needs to cluster the models in the memory, it first requests the models to make predictions to the examples in the sliding window. A data set is then created where each example is a list of correctness of a given model's predictions to the examples in the sliding window. A value of 1 represents correct prediction while a value of 0 represents incorrect prediction. Therefore, if the window has size b, each training example fed to the clustering algorithm is a list of b binary attributes, and if the memory has $e \times r$ models, there are $e \times r$ examples. We refer to these examples as "clustering examples", to avoid confusion with the training examples from the data stream that are stored in the sliding window. Any clustering algorithm can be used to cluster the clustering examples, however, CDCMS only needs a "snapshot" of the model space to build ensembles with different diversity levels

or to identify relevant past knowledge. Thus, an offline clustering algorithm is sufficient. In this study, we used Expectation Maximisation (EM) clustering (Dempster, Laird, and Rubin, 1977) with 10-fold cross-validation to decide the potential number of clusters. EM clustering was chosen not only because it is well-studied in the literature, but also because of its nature which partitions the data into clusters with statistical methods. This is important for clustering in the model space strategy as this strategy relies on the predictions of the models to a window of examples, thus the dimensionality is likely to be high. Hence, distance based or density based offline clustering methods may not be ideal.

4.2.1.2 Drift Handling and Recurring Concepts Identification

DDD has shown that past knowledge can help to speed up learning a new similar concept and to deal with gradual concept drifts (Minku and Yao, 2012). As this could result in swifter recovery from drifts than resetting the learning system upon drift detection (Minku and Yao, 2012), we also create a highly diverse ensemble upon drift detection. However, even though DDD can deal with new concepts well, it cannot handle recurring concepts well, as it does not have a memory of past models. To achieve an approach that can deal with both new and recurring concepts, we hereby propose a new method to create ensembles with different diversity levels by making use of the results of the clustering algorithm explained in Section 4.2.1.1.

Models clustered together are likely to represent the same or similar concepts, whereas models in different clusters are likely to represent different concepts. CDCMS thus creates highly diverse ensembles by taking the model trained with the largest number of examples from each cluster (line 32, Algorithm 3). In the event of having only one cluster, models at every r index interval are taken to build the highly diverse ensemble E_{nh}. The fact that the proposed method creates a highly diverse ensemble consisting of different past concepts

means that, in the case of encountering a recurring concept, one of these models may hold the same or a similar concept, which could help to achieve fast predictive performance recovery from the drift or prevent a catastrophic drop in predictive performance right after the drift. This is a key potential advantage over DDD which creates highly diverse ensembles by enforcing high diversity on an ensemble trained from a single concept. DDD's strategy may be inadequate to handle recurring concepts, as it does not incorporate models representing concepts older than the most recent one.

To identify and deal with recurring concepts, once b new training examples are received after the drift detection (line 8, Algorithm 3), CDCMS clusters the models in the memory with the new model created upon the last drift detection (line 9, Algorithm 3). If the new model is clustered with other past models, the current concept is regarded as a recurring concept (line 10, Algorithm 3). CDCMS then recovers the $(e-1)$ models, which trained with the largest number of training examples from the same cluster to fill up the new representative ensemble (line 14, Algorithm 3), where e is the size of the new representative ensemble. If the new model is not clustered with other past models, the current underlying concept is regarded as a new concept (line 11, Algorithm 3). The new representative ensemble will then carry on to build from scratch, starting as an ensemble containing the new model as its only member (line 12, Algorithm 3). Besides, if the drift leads to a concept that shares similarities with any past concept, this method of using clustering to recover relevant past models can also help the system to adapt to the drift.

4.2.1.3 Diversity-Based Memory Management Strategy

CDCMS periodically (at every b time steps) creates a new model c_n to learn the current concept. If the maximum size e of the representative ensemble E_{nl} is reached, the least performing model $c_{worst} \in E_{nl}$ is moved into the memory R before adding c_n to E_{nl}. If R

had already reached the predefined maximum size ($e \times r$), CDCMS will search for the model $c_{sim} \in R$ that is the most similar to c_{worst}. At last, either c_{worst} or c_{sim} will be kept in R while the other one will be deleted. This strategy will maintain the memory with a maximum size while ensure that diversity is preserved, increasing the chances of having various past knowledge stored in the memory. Yule's Q-statistics (Q) (Kuncheva and Whitaker, 2003) is employed to find the most similar model. It is used to calculate the similarity between two models c_i and c_k as follows:

$$Q_{i,k} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}} \tag{4.2}$$

where $N^{a,b}$ is the number of examples where the classification by c_i is a and the classification by c_j is b, a value of 1 represents a correct classification while a value of 0 represents a misclassification. Q varies between 1 and -1. Models that tend to classify the same examples correctly will have positive values of Q, whereas models that tend to classify different examples incorrectly will have negative values of Q. Additionally, a predefined threshold θ is used to ensure that c_{worst} and c_{sim} have certain similarity. If $Q_{worst,sim} \leq \theta$, the model trained with more examples is kept in memory. Otherwise, c_{worst} is discarded. This memory management strategy is an improvement over DP, which will be discussed in Section 4.2.1.5.

4.2.1.4 Making Prediction on New Examples

Regarding the prediction aspect of CDCMS, Algorithm 4 presents the pseudo-code. Most of time, CDCMS only uses E_nl to make prediction to new examples (line 9, Algorithm 4). Since drift detection, all three ensembles (the highly diverse ensemble E_{nh}, the old and new main ensembles E_{ol}, E_{nl}) make combined predictions until E_{nl} performs better than either of E_{ol} and E_{nh} (line 3-7, Algorithm 4). This is becc

Algorithm 4 CDCMS Makes Predictions

1: **function** MAKEPREDICTION(s_t, $drift_level$)
2: $combinedVote \leftarrow double[]\{0.0, 0.0\}$ ▷ Only two values because of binary classification
3: **if** $drift_level == OUTCONTROL$ **OR** ($drift_level == NORMAL$ **AND** $E_nl.preqAcc() < E_ol.preqAcc()$ **AND** $E_nl.preqAcc() < E_nh.preqAcc()$) **then**
4: $combinedVote[vote] += vote \times E_nl.getWeight()$ ▷ Weight based on E_nl's prequential accuracy
5: $combinedVote[vote] += vote \times E_ol.getWeight()$ ▷ Weight based on E_ol's prequential accuracy
6: $combinedVote[vote] += vote \times E_nh.getWeight()$ ▷ Weight based on E_nh's prequential accuracy
7: **Return** $combinedVote.indexWithMaxValue()$
8: **else**
9: **Return** $E_nl.makePrediction(s_t)$
10: **end if**
11: **end function**

Algorithm 5 presents the pseudo-code of an ensemble making prediction on new example. E_{ol} and E_{nl} give predictions based on weighted majority among their base learners (line 8, Algorithm 5), while E_{nh} gives predictions by a simple majority among its base learners (line 6, Algorithm 5). This is because E_{nh} consists of models from different concepts and CDCMS do not have enough examples from the new concept to assign the base learners with reliable weights right after drift detection.

The final combined prediction is by weighted majority over the predictions given by E_{ol}, E_{nl} and E_{nh} (line 11, Algorithm 5). The weights used for the majority votes are computed as:

$$w_t^i = \begin{cases} 1, & \text{if } c_i \in E_{nh} \\ \frac{acc_t(c_i)}{\sum_{l=1}^{M} acc_t(c_l)}, & \text{otherwise} \end{cases} \tag{4.3}$$

where c_i is one of the L base learners and $M{=}L$ when computing the predictions by E_{ol}, E_{nl} and E_{nh}, or $c_i \in \{E_{ol}, E_{nl}, E_{nl}\}$ and $M{=}3$ when computing the overall prediction by

Algorithm 5 Ensemble Makes Predictions

```
1: function MAKEPREDICTION(s_t)
2:     combinedVote ← double[]{0.0, 0.0}        ▷ Only two values because of binary classification
3:     for each c ∈ Ensemble do
4:         vote ← c.makePredictionOn(s_t)        ▷ vote in either 0 or 1 for binary classification
5:         if thisensembleisE_nh then
6:             combinedVote[vote]+ = vote        ▷ Simple majority vote, See Eq.4.3
7:         else
8:             combinedVote[vote]+ = vote × c.getWeight()   ▷ Weight based on c's prequential accuracy,
   See Eq.4.3
9:         end if
10:    end for
11:    Return combinedVote.indexWithMaxValue()
12: end function
```

CDCMS. The prequential accuracy at time step t ($acc_t(c_i)$) is computed as:

$$acc_t(c_i) = \begin{cases} acc_t^{ex}(c_i), & \text{if } t = f \\ \frac{acc_t^{ex}(c_i) + \alpha \times acc_{t-1}(c_i) \times (t-f)}{t-f+1}, & \text{otherwise} \end{cases} \tag{4.4}$$

where acc_{ex} is 0 if the prediction to the current training example ex before learning is incorrect and 1 if it is correct, f is the first time step used in the calculation, and α is a fading factor. The overall weighted majority vote among M models is computed as:

$$W.M.V._t = \arg\max_{y \in Y} \sum_{i=1}^{M} w_t^i I(c_i(x_t) = y_t) \tag{4.5}$$

where I is the indicator function and x_t are the input attributes of the example being predicted, and y_t is the true label of x_t.

4.2.1.5 Time Complexity

This section presents the high-level time complexity discussion of CDCMS. The time complexity of CDCMS consists of the time complexities of making predictions, training base learners, clustering in the model space and managine the memory by diversity.

To discuss the time complexity of making predictions and training, let us consider each ensemble has e base learner. Each base learners has the time complexity of making a prediction in $O(M_p)$. Thus, the time complexity of CDCMS to make a prediction at any time step during stable concept period is in $O(eM_p)$, as it makes prediction by combined vote of the base learners in the main ensemble. As for the period between the drift detection and until the new main ensemble is deemed to be confident enough in making predictions by itself, the time complexity of CDCMS to make a prediction is $O(3eM_p) = O(eM_p)$ because all three ensemble (E_{nl}, E_{nh} E_{ol}) participate in making combined prediction. The training time complexity of CDCMS can be determined similarly because it always trains the new representative ensemble only. Thus, the training time complexity is $O(eM_t)$, where $O(M_t)$ is the training time complexity of the base learner.

For the time complexity of the clustering in the model space strategy, let us consider H models with the time complexity of making a prediction in $O(M_p)$. These models are first required to make predictions to b examples in the sliding window B to build a set of "clustering example". Each "clustering example" holds the correctness of predictions made by a given model. Thus, time complexity of this procedure is $O(HM_pb)$. A clustering method with time complexity $O(G)$ is then applied to cluster those H models based on this "clustering example" set. Finally, the overall time complexity to obtain the clustering results of those H models based on b examples is in $O(HM_pb + G)$. This time complexity applies to both building the highly diverse ensemble and identifying recurring concepts because the clustering in the model space strategy is used in both situations despite the results of the

clustering being used differently.

Regarding the diversity-based memory management strategy proposed in Section 4.2.1.3, if we consider the memory has $L = e \times r$ models with prediction time complexity $O(M_p)$, the time complexity is in $O(LM_pb)$, because this strategy also relies on the prediction made by L models to a sliding window of b examples.

In contrast, DP uses an exhaustive search method. It simulates all possible memory states by eliminating each model from the memory and then computes their respective diversity level. It then keeps the memory state that has the highest diversity level as the new memory. To compute the diversity level of each possible memory state, the average Q-statistics (Q_{av}) over all pairs of models is used:

$$Q_{av} = \frac{2}{L(L-1)} \sum_{i=1}^{L-1} \sum_{k=i+1}^{L} Q_{i,k} \tag{4.6}$$

which has the time complexity in $O(L^2M_p)$. As the number of possible memory states is L, the overall time complexity of the exhaustive search is $O(L^3M_p)$, which is less efficient than the $O(LM_p)$ time complexity of the proposed memory management strategy.

4.2.2 Design of the Experiment

The section presents the set up of the experiments to evaluate the predictive performance of CDCMS, continue to answer research question 1. CDCMS was first compared against six existing approaches on artificial data streams, to evaluate when it performs the best and worst respectively. It is then compared against the same set of existing approaches on real-world data streams to obtain the general idea of its predictive performance in real-world applications. Massive Online Analysis (MOA) (Bifet, Holmes, et al., 2010) was chosen to be

the platform to perform the experiments. The set up includes the artificial and real-world data streams used in the experiments, the procedure of hyper-parameter tuning and the experiments, which are detailed in Sections 4.2.2.1 and 4.2.2.2 respectively.

4.2.2.1 Data Streams

Ten artificial data streams were created using the Sine (Gama, Medas, et al., 2004), Agrawal (Agrawal, Imielinski, and Swami, 1993), SEA (Street and Y. Kim, 2001) and STAGGER (Schlimmer and Granger Jr., 1986) generators from the Massive Online Analysis (MOA) framework (Bifet, Holmes, et al., 2010). In particular, we have created two version of this set of artificial data streams. The first version consists of abrupt drift only, i.e. each concept drift happens in one time step. The second version consists of gradual drift only, where each concept drift happens over a transitional period of 2k time steps (1k time steps before and after the central drift location respectively). As discussed in Section 4.1.2.1, artificial data streams allows us to specify the severity of the concept drifts and the order of the concepts, enabling us to obtain a detailed understanding of the predictive performance of the proposed approach in different situations. A summary of the artificial data streams is presented in Table 4.8, while the details of the streams are as follows.

Sine1-2 streams consist of two numerical input attributes. Their concepts f1-f4 are defined in MOA, and involve the mathematical sine function in its calculation. Concepts Sine-f1 and Sine-f2 share the same decision boundary but the decision regions of two classes are swapped. Concepts Sine-f3 and Sine-f4 also have the same relationship with each other as Sine-f1 and Sine-f2. Agr1-4 and SEA1-2 streams are Agrawal and SEA streams respectively, which had been discussed in Section 4.1.2.1. STA1-2 streams consist of three categorical input attributes. Their input attributes (size, colour and shape) are the description for a domain of objects. The STAGGER concepts are defined in MOA, where any of one or more

Table 4.8: Summary of Artificial Data Streams

Data Stream	Concept Drift Sequence	#Examples
Sine1	f3→f4→f3	60k
Sine2	f1→f2→f3→f4→f1	100k
Agr1	f1→f3→f4→f7→f10	100k
Agr2	f7→f4→f6→f5→f2→f9	120k
Agr3	f4→f2→f1→f3→f4	100k
Agr4	f1→f3→f6→f5→f4	100k
SEA1	f5→f3→f1→f2→f4	100k
SEA2	f5→f1→f4→f3→f2	100k
STA1	f1→f2→f3→f2	80k
STA2	f2→f3→f1→f2	80k

- Coloured rows (lime / light grey) represent that the stream contains recurring concepts.
- f*n* represents the n-th function of the generator, i.e., f1 in Agrawal1 refers to the first function of Agrawal generator.
- Central position of the all drift is at every 20k time steps.

input attributes are taken into account and the remaining are irrelevant. All synthetic data streams are binary classification problems. Concept drifts were simulated by connecting two data streams with different concepts, using a sigmoid function to decide the probability of data coming from the pre and post drift concepts during the transitional period.

Regarding the severity of drifts in artificial data streams, Table A.1 presents the percentage difference of each pair of concepts used in the experiments. The details of calculating the differences is presented in Section A.

In addition to artificial data streams, we also adopted three real-world data streams to obtain a general idea of the predictive performance of CDCMS in practical applications. These real-world data streams are KDDcup99, Power Supply, and Sensor streams (Zhu, n.d.). KDDcup99 was produced with real-world data but the attacks to a network were simulated. Thus, the concept drifts in this stream can be considered artificial.

4.2.2.2 Experiment Setup

To answer research questions 1.2.2 and 1.2.3, we quantitatively compared the predictive performance of CDCMS against five existing approaches and DP. All approaches used Hoeffding Tree with Naïve Bayes at Leaves (HTNB) as the base learner. The approaches and the reasoning behind their choice are listed below:

- *HTNB* (Domingos and Hulten, 2000): to compare against the base learner.
- *Online Bagging (OzaBag)* (Oza, 2005): to compare against a baseline ensemble approach without any concept drift handling mechanism.
- *OAUE* (Brzeziński and Stefanowski, 2014): to compare against a passive ensemble approach.
- *DDD* (Minku and Yao, 2012): to compare against an active ensemble approach which uses a highly diverse ensemble to handle gradual and mild severity drifts.
- *DP* (Chiu and Minku, 2018): to compare against the preliminary work on maintaining a diverse memory to handle recurrent drifts.
- *RCD* (Gonçalves Jr and Barros, 2013): the compare against an approach designed to handle recurrent drifts.

Similar to the experiment setting of DP, presented in Section 4.1.2.2, accuracy was measured prequentially, i.e., each example in the data stream was used to test the approach before training on it. A fading factor of 0.999 was used to make the past examples less important to the current accuracy (Gama, Rqauel Sebastião, and Rodrigues, 2013). The prequential performance is calculated using Equation 4.1, which is the same we used in Section 4.1.2.2. The prequential accuracy were sampled at every 500th example.

To tune the hyper-parameters of each approach for experiments with artificial data streams, we created four additional artificial data streams (one for each stream type: Sine, Agrawal, SEA, STAGGER) with random orders of concepts, drift width and central drift location. The hyper-parameter values that lead to the highest average prequential accuracy on the randomised data stream for Sine, Agrawal, SEA, and STAGGER were then chosen to be used in the experiments with data streams Sine1-Sine2, Agr1-Agr4, SEA1-SEA2, and STA1-STA2, respectively. For experiments with real-world data streams, the first 10% examples of each stream were used for hyper-parameter tuning purposes. The hyper-parameters' options are presented in Table 4.9 while the chosen value of the hyper-parameters are presented in Tables 4.10 and 4.11.

In the experiments, each approach ran for thirty times on each data stream. To analyse the experiment results, Friedman test was adopted to determine if there is a statistically significance in the average prequential accuracy among all approaches. If there is, Nemenyi post-hoc test was used to determine which approach performed significantly different from the top-ranked approach. In deciding which approach is the best in each data stream, we consider the mean and the standard deviation of their thirty runs average prequential accuracy.

Based on the comparison, a few representative runs from experiments with artificial data streams were chosen to analyse CDCMS's behaviour qualitatively in detail. This will complement the answer to RQ1.2.2 and provide an in-depth understanding of the reasons behind the results, answering RQ1.2.3. Other cases can be explained similarly.

Table 4.9: Options for Hyper-parameters

Hyper-parameter	HTNB	DP	RCD	CDCMS	OAUE	OzaBag	DDD
Base Learner	-	Hoeffding Tree with Naïve Bayes at leaves (HTNB)					
Drift Detector (ddm)	{DDM, EDDM, ADWIN}				-		{DDM, EDDM, ADWIN}
Ensemble Size (n)	-			{5, 10, 15, 20}			
Memory Size (m)	-	{2, 4, 6, 8, 10}	{5, 10, 15, 20}	-			
Diversity Measure (d)	-	{Entropy Measure, Q-Statistics}	-				
Memory size multiplier (r)	-			{4, 6, 8, 10}	-		
Evaluation Frequency (e) / Buffer Size (b)	-	{200, 400, 600, 800, 1000}			-		
Similarity Threshold (sim)	-			{0.4, 0.6, 0.8, 0.9}	-		
p-value Threshold (p)	-		{0.01, 0.05}	-			
λ	-						{0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5}

- Grey rows denote pre-chosen hyper-parameter values.

4.2.3 Analysis

This section presents the analysis of the experiment results to validate the predictive performance of CDCMS against existing approaches. Sections 4.2.3.1 and 4.2.3.2 analyses the experiment results on artificial data streams with gradual drifts and abrupt drifts respectively. Section 4.2.3.3 analyses the experiment results on real-world data streams.

Table 4.10: Chosen Hyper-parameters (Artificial Data Streams)

Approach	Chosen Hyper-parameters
Sine	
HTNB	-
DP	m=4, b=500, ddm=DDM, d=Q-Statistics
RCD	m=5, b=500, e=1000, ddm=DDM, p=0.01
CDCMS	n=20, r=4, b=600, sim=0.8, ddm=ADWIN
OAUE	n=15, e=400
OzaBag	n=15
DDD	n=5, λ=0.1, ddm=DDM
Agrawal	
HTNB	-
DP	m=6, b=400, ddm=EDDM, d=Entropy Measure
RCD	m=5, b=200, e=1000, ddm=EDDM, p=0.01
CDCMS	n=15, r=4, b=400, sim=0.8, ddm=EDDM
OAUE	n=20, e=400
OzaBag	n=5
DDD	n=10, λ=0.001, ddm=EDDM

Approach	Chosen Hyper-parameters
SEA	
HTNB	-
DP	m=8, b=800, ddm=DDM, d=Q-Statistics
RCD	m=5, b=500, e=400, ddm=DDM, p=0.01
CDCMS	n=10, r=8, b=600, sim=0.8, ddm=DDM
OAUE	n=15, e=400
OzaBag	n=20
DDD	n=10, λ=0.05, ddm=DDM
STAGGER	
HTNB	-
DP	m=4, b=600, ddm=ADWIN, d=Entropy Measure
RCD	m=5, b=600, e=1000, ddm=DDM, p=0.05
CDCMS	n=20, r=6, b=800, sim=0.8, ddm=DDM
OAUE	n=10, e=600
OzaBag	n=20
DDD	n=15, λ=0.1, ddm=DDM

4.2.3.1 Results with Artificial Gradual Drift Data Streams

Table 4.13 presents the means and standard deviations to compare the thirty runs average prequential accuracy of CDCMS against existing approaches in artificial data streams with gradual drifts. It shows that CDCMS achieved similar or better average prequential accuracy

Table 4.11: Chosen Hyper-parameters (Real-world Data Streams)

Approach	Chosen Hyper-parameters
KDDcup99	
HTNB	-
DP	m=8, b=200, ddm=ADWIN, d=Q-Statistics
RCD	m=10, b=500, e=800, ddm=ADWIN, p=0.01
CDCMS	n=20, r=6, b=600, sim=0.6, ddm=ADWIN
OAUE	n=20, e=200
OzaBag	n=15
DDD	n=15, λ=0.01, ddm=ADWIN

Approach	Chosen Hyper-parameters
Power Supply	
HTNB	-
DP	m=4, b=1000, ddm=EDDM, d=Q-Statistics
RCD	m=10, b=800, e=1000, ddm=DDM, p=0.01
CDCMS	n=5, r=4, b=200, sim=0.8, ddm=DDM
OAUE	n=5, e=200
OzaBag	n=10
DDD	n=10, λ=0.05, ddm=EDDM

Approach	Chosen Hyper-parameters
Sensor	
HTNB	-
DP	m=2, b=400, ddm=DDM, d=Q-Statistics
RCD	m=10, b=800, e=600, ddm=DDM, p=0.05
CDCMS	n=5, r=10, b=400, sim=0.9, ddm=DDM
OAUE	n=5, e=200
OzaBag	n=20
DDD	n=20, λ=0.05, ddm=EDDM

in most cases, as depicted by bold text cells. CDCMS only performed significantly worse than DDD and DP in Sine1-gradual. In other cases, even though CDCMS was not always the top-ranked, its prequential accuracy had no statistical significant difference with the top-

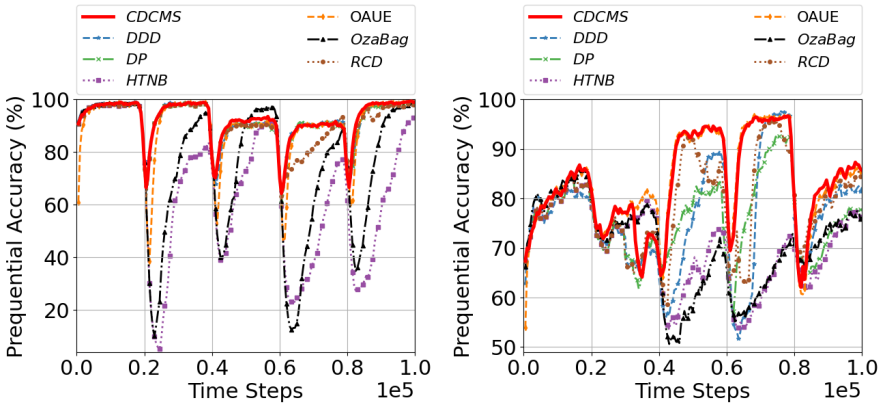
ranked approaches. Over the next paragraphs, we provide a detailed analysis of these results based on the first single run from data streams with representative predictive performance results.

Table 4.12: Average Prequential Accuracy (Standard Deviation) Across 30 Runs For Artificial Data Stream with Gradual Drifts

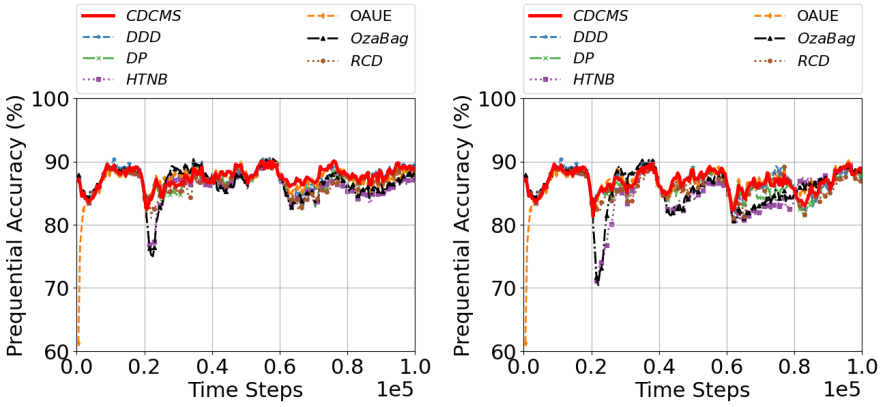
Data Stream	CDCMS	HTNB	OzaBag	DP	OAUE	RCD	DDD
Sine1	87.737% (6.109%)	67.745% (22.178%)	75.283% (23.252%)	**88.3% (5.958%)**	85.92% (9.998%)	87.806% (6.026%)	**87.786% (7.341%)**
Sine2	**92.692% (7.666%)**	65.555% (26.393%)	74.39% (26.552%)	**91.689% (7.935%)**	89.968% (12.008%)	90.39% (8.643%)	**91.632% (9.236%)**
Agr1	**89.53% (8.576%)**	74.003% (11.758%)	71.998% (12.03%)	85.268% (10.152%)	**89.707% (9.193%)**	85.634% (8.302%)	87.293% (10.867%)
Agr2	**78.494% (11.311%)**	67.301% (12.327%)	68.304% (12.175%)	71.877% (9.767%)	**80.458% (9.153%)**	73.954% (10.229%)	74.09% (11.955%)
Agr3	**82.895% (9.108%)**	70.068% (8.178%)	70.295% (8.731%)	75.833% (7.616%)	**83.413% (9.116%)**	78.58% (9.037%)	77.759% (10.457%)
Agr4	**81.187% (11.45%)**	69.749% (14.208%)	67.592% (14.072%)	75.46% (12.894%)	**82.505% (10.119%)**	77.783% (11.093%)	77.647% (13.051%)
SEA1	**87.591% (1.488%)**	86.117% (2.239%)	86.581% (2.777%)	86.879% (1.709%)	87.168% (2.505%)	86.71% (1.855%)	**87.525% (1.652%)**
SEA2	**86.841% (1.652%)**	85.036% (3.392%)	85.733% (3.62%)	86.296% (1.854%)	**86.713% (2.579%)**	86.033% (1.965%)	**86.738% (1.964%)**
STA1	**98.268% (4.193%)**	86.462% (14.325%)	86.782% (13.774%)	**98.183% (4.306%)**	96.898% (7.258%)	97.783% (4.91%)	**98.132% (4.466%)**
STA2	**98.155% (4.623%)**	78.25% (16.809%)	78.511% (16.635%)	**98.079% (4.65%)**	96.842% (7.484%)	97.713% (5.402%)	97.949% (5.028%)

- Lime or light grey in the column of Data Stream means that artificial data stream consists of recurring concepts.
- The p-values of Friedman tests are all $<$2.2E-16.
- Cells with bold text represent the best accuracy (or approaches have no statistical significant difference with best one) in the data stream, based on the p-value ($\geq$ 0.05) in Nemeyi post-hoc test.

4.2.3.1.1 Cases where CDCMS performed better: Sine2-gradual, Agr3-gradual, SEA1-gradual and SEA2-gradual are chosen as the representative cases to show CDCMS performed significantly better than most other approaches. Figure 4.3 presents the prequential accuracy of the approaches over these streams.



(a) Sine2-Gradual (b) Agrawal3-Gradual



(c) SEA1-Gradual (d) SEA2-Gradual

Figure 4.3: Prequential Accuracy in Sine2-gradual, Agr3-gradual, SEA1-gradual and SEA2-gradual streams

Sine2-gradual consists of four gradual severe drifts (see Tables 4.8 and A.1). Figure 4.3(a) shows that CDCMS was usually among the best approaches in Sine2-gradual in terms of both fast reaction and recovery from all drifts. It archived particularly good prequential accuracy at the rear of the data stream, which is a recurring concept. It also archived good

stable accuracy after drifts. The behaviour log of CDCMS shows that it successfully identified the first drift as a drift leading to a new concept. However, it mistakenly considered the second drift as a recurrent drift and recovered a set of models that mainly held the knowledge of concept Sine-f1. This was because concepts Sine-f1 and Sine-f3 are similar, comparing to other concept appeared upon this drift (see Table A.1). Thus, CDCMS performed the best during the period of concept f3, as shown in Figure 4.3(a) (40k-60k time steps). At the third drift, CDCMS also considered this drift as a recurrent drift and recovered a set of models holding the knowledge of concepts Sine-f1 and Sine-f2. Despite this new low diversity ensemble holding a partially different concept (*diff*(f1,f4): 73.2%, *diff*(f2,f4): 26.8%), it still managed to be one of the fastest approaches in reacting to this drift and adapting to concept Sine-f4. At the fourth drift, CDCMS successfully identified the recurring concept and managed to recover corresponding models from the memory. That is why it recovered from this drift faster than the other approaches. DP and RCD could not identify this recurring concept because their methods for identifying recurring concepts are conservative.

This analysis shows that the clustering in the model space strategy together with the diversity-based memory management strategy worked better than DP and RCD in handling gradual severe recurrent drifts occurring after several gradual severe drifts. Interestingly, despite OzaBag having an initially slower recovery (as it is not designed for non-stationary data streams), it eventually managed to recover its accuracy from this drift and surpassed all approaches' prequential accuracy at around 5k-th time steps. This may indicate that OzaBag is better for stationary periods, as it will not be negatively affected by false-positive drift detection, nor by introducing new base models.

As for Agrawal3-gradual, the first drift is mild severe, whereas other three drifts are severe (see Tables 4.8 and A.1). Although Figure 4.3(b) shows that there is a slight drop in CDCMS's prequential accuracy at around the 32,450th time step, the predictive performance of CDCMS and OAUE generally is similar, which is confirmed by the statistical test (see

Table 4.12). Looking back to Figure 4.3(b), CDCMS recovered its prequential accuracy faster than other approaches from the last drift, which is a recurrent drift. It also recovered quite well from the second and the third drifts. Surprisingly, the behaviour log shows that CDCMS only had a delayed drift detections for the first drift but no drift detection at the second and the third drifts. Yet, it could still adapt to the changes at these drifts because the periodic ensemble committee replacement gradually and continuously updates the knowledge held in the main ensemble, enabling it to recover from missed drift detection. In fact, OAUE, which also uses a similar mechanism, obtained similar overall accuracy to CDCMS, suggesting that this type of strategy is the most suitable for this data stream. Besides, a few false-positive drift detection from the 32,450th time step to the 33,278th time step were given, causing CDCMS to have a drop in prequential accuracy. However, the CDCMS considered these false-positive drift detection as recurring changes. It recovered a mixture set of models with knowledge of concepts Agrawal-f4 and Agrawal-f2, which are similar to each other. This shows the robustness of CDCMS to false-positive drift detection. For the last drift, CDCMS successfully identified the recurrent drift and recovered a set of models corresponding to concept Agrawal-f4. This shows that the clustering in the model space strategy was able to distinguish two sets of models holding similar concepts, which coheres the analysis of the experiment with Sine2-gradual.

CDCMS had similar behaviours in learning SEA1-gradual and SEA2-gradual streams. It detected the first concept drift of these two data streams promptly, but then it has no drift detection for the rest of these two data streams. Except that there is a false drift detection were given at 62,571th time step in SEA2-gradual stream, which is after the central drift point of the third drift. CDCMS considered this was a recurrent drift. It recovered a set of models having the knowledge of concept SEA-f1. Although the concept SEA-f1 is not very similar to the concept SEA-f3, it is the most similar one in the memory of CDCMS upon that time step. One potential reason behind no drift detection could be the concepts of SEA are easy to learn,

as they are simple hyper-plane concepts. Another potential reason could be the presence of the 10% noise, which may make the concepts of SEA even more similar. An additional potential reason could be the periodic ensemble committee replacement strategy was enough for CDCMS to adapt to the changes in SEA1-gradual and SEA2-gradual streams. Thus, these indicates that such simple strategy alone, sometimes, is sufficient to handle gradual drifts, especially when the drift not severe.

4.2.3.1.2 Cases where CDCMS performed worse: Table 4.12 shows that Sine1-gradual is the case where CDCMS performed significantly worse than at least one other existing approach. Figure 4.4 presents the prequential accuracy of the approaches over Sine1-gradual.

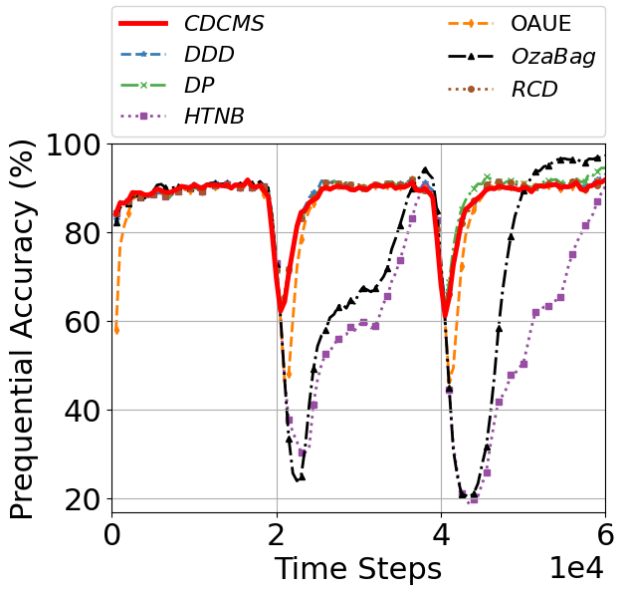


Figure 4.4: Prequential Accuracy in Sine1-gradual stream

Tables 4.8 and A.1 show that Sine1-gradual consists of two extremely severe drifts where the second drift is a recurrent drift. The best approaches for Sine1-gradual were DDD and DP. Figure 4.4 shows that DDD achieved the best recovery at the first drift but the magnitude of the improvement was relatively small. Even though CDCMS and DDD

both have a highly diverse ensemble for prequential accuracy recovery, DDD enforces low diversity learning on the highly diverse ensemble after the drift, whereas CDCMS uses low diversity only in the main ensemble. This potentially enabled DDD to recover better from the drift. The same figure also shows that DP achieved the best recovery from the second drift, which is a recurrent drift leading back to the concept Sine-f3. Both DP and CDCMS successfully identified this recurrent drift. DP correctly recovered a model that purely learnt from the corresponding concept Sine-f3 while CDCMS recovered a set of models that had been learning since the first stable concept period of Sine-f3 and a set of models that had learnt during the transitional period of the first drift. The reason behind such a model recovery of CDCMS at this drift can be explained as follows.

CDCMS considered the false-positive drift detection during the transitional period of the first drift (Sine-f3→Sine-f4) as a recurrent drift and then recovered a set of models corresponding to concept Sine-f3 and a few models created during the transitional period. As this set of mixed concepts models had learnt from this transitional period, the clustering in the model space strategy considered it as an intermediate concept between Sine-f3 and Sine-f4. As the transitional period of the second drift (Sine-f4→Sine-f3) is also composed of these two concepts, CDCMS considered this drift as the recurrence of such intermediate concept, rather than the recurrence of Sine-f3. In other words, the clustering in the model space strategy precisely distinguished models with differences in the underlying concept. While this distinction would normally be advantageous, it was detrimental in this case because the new concept was Sine-f3, and no new drift detection was triggered when moving from the intermediate concept of Sine-f3 and Sine-f4 to the pure Sine-f3 concept. Besides, we can again see that OzaBag performed better than all concept drift adaptation approaches at the end of the concepts, which also happened in Sine2-gradual.

4.2.3.1.3 Short Summary CDCMS usually achieved similar or better results than other approaches on artificial gradual drift data streams, being more robust to and fast recover from multiple types of gradual drift (RQ1.2.2). This is because the clustering in the model space strategy managed to recover relevant past models to handle concept drifts, in particular, severe recurrent drifts and severe drifts taking to a concept that shares similarities with past concepts. However, in cases where there are extremely severe gradual recurrent drifts, CDCMS could recover models that learnt from intermediate concepts, slightly hindering the accuracy (RQ1.2.3).

4.2.3.2 Results with Artificial Abrupt Drift Data Streams

Table 4.13 resents the means and standard deviations to compare the thirty runs average prequential accuracy of CDCMS against existing approaches across artificial abrupt drift data streams. It shows that CDCMS achieved similar or better accuracy than other approaches in all abrupt drift data streams. Thus, the analysis done in this section will focus on why CDCMS performed significantly better than DP and DDD for Sine1-abrupt, given that it had performed significantly worse for Sine1-gradual. Sine2-abrupt has been chosen as an additional representative case. Figure 4.5 presents the prequential accuracy of the approaches over Sine1-abrupt and Sine2-abrupt streams.

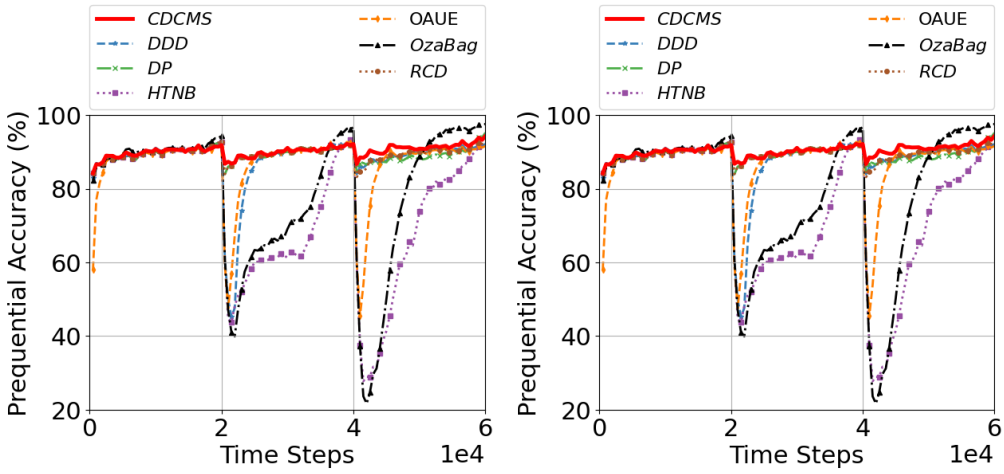
Tables 4.8 and A.1 show that Sine1-abrupt consists of two extremely severe drifts, where the second drift is a recurrent drift. Figure 4.5(a) shows that CDCMS was the best approach in terms of both reacting to and recovering from drifts. At the first drift, CDCMS mistakenly considered it as a recurrent drift and recovered only one model which was created 200 time steps before the drift. Comparing to other models in memory, this model did not learn enough examples from the pre-drift concept. Thus, we can consider that the new representative ensemble consists of new models only, which enabled better adaptation to

Table 4.13: Average Prequential Accuracy (Standard Deviation) Across 30 Runs For Artificial Data Stream with Abrupt Drifts

Data Stream	CDCMS	HTNB	OzaBag	DP	OAUE	RCD	DDD
Sine1	**90.248%**	74.35%	78.58%	89.374%	86.846%	**89.527%**	88.54%
	(1.586%)	(18.25%)	(20.238%)	(1.877%)	(9.164%)	**(1.827%)**	(6.961%)
Sine2	**95.894%**	65.559%	76.702%	91.557%	91.593%	**94.061%**	93.755%
	(3.055%)	(27.081%)	(25.508%)	(7.497%)	(9.918%)	**(4.032%)**	(7.22%)
Agr1	**90.31%**	76.283%	76.282%	87.209%	**90.666%**	86.706%	88.805%
	(8.367%)	(12.768%)	(10.802%)	(8.58%)	**(8.733%)**	(8.735%)	(9.617%)
Agr2	**79.528%**	67.575%	68.871%	72.668%	**81.807%**	74.711%	75.038%
	(10.862%)	(12.765%)	(12.383%)	(10.878%)	**(8.609%)**	(10.684%)	(12.379%)
Agr3	**83.888%**	69.972%	70.404%	76.95%	**84.499%**	77.986%	79.789%
	(9.128%)	(8.402%)	(8.892%)	(8.428%)	**(8.921%)**	(9.576%)	(10.644%)
Agr4	**80.775%**	71.867%	70.141%	78.063%	**82.984%**	77.679%	78.324%
	(12.21%)	(15.6%)	(14.376%)	(12.118%)	**(10.584%)**	(11.969%)	(13.253%)
SEA1	**87.569%**	86.136%	86.58%	86.434%	**87.378%**	86.527%	**87.538%**
	(1.528%)	(2.467%)	(2.974%)	(1.893%)	**(2.551%)**	(1.633%)	**(1.701%)**
SEA2	**86.935%**	85.05%	85.773%	85.976%	**86.953%**	85.846%	**86.884%**
	(1.761%)	(3.605%)	(3.856%)	(2.514%)	**(2.667%)**	(2.105%)	**(2.08%)**
STA1	**99.936%**	90.324%	90.142%	99.904%	98.129%	**99.945%**	98.964%
	(0.223%)	(11.997%)	(11.639%)	(0.291%)	(5.867%)	**(0.203%)**	(3.343%)
STA2	**99.953%**	86.898%	87.342%	99.893%	98.004%	**99.946%**	99.024%
	(0.156%)	(15.764%)	(16.241%)	(0.3%)	(6.281%)	**(0.161%)**	(3.2%)

- Lime or light grey in the column of Data Stream means that artificial data stream consists of recurring concepts.
- The p-values of Friedman tests are all $<$2.2E-16.
- Cells with bold text represent the best accuracy (or approaches have no statistical significant difference with best one) in the data stream, based on the p-value ($\geq$ 0.05) in Nemeyi post-hoc test.

the post-drift concept. A further investigation of the ensembles' weights right after this drift shows that the new representative ensemble obtained a weight of around 56% while the highly diverse ensemble helped to recover the accuracy, with the weight of around 38%.



(a) Sine1-Abrupt

(b) Sine2-Abrupt

Figure 4.5: Prequential Accuracy in Sine1-abrupt and Sine2-abrupt Streams

In contrast, the old representative ensemble performed poorly, having a very low weight of around 5.6%. As this drift is abrupt, it is reasonable that training new models is the best strategy (Minku and Yao, 2012). CDCMS successfully benefited from that and recovered well from this drift.

On the contrary, OAUE and DDD had a drastic drop in prequential accuracy at the first drift. For OAUE, it usually has adaptation lags in handling abrupt drifts, as it is a implicit concept drift adaptation approach. For DDD, it was because its highly diverse ensemble was not diverse enough to have a good generalisation when handling the drift, i.e., the λ value suggested by the hyper-parameter tuning procedure was not small enough. Thus, the highly diverse ensemble did not perform well and could not draw a higher weight. This indirectly led DDD to put too much emphasis on the old representative ensemble, being unable to recover well from this drift. An additional experiment using a smaller λ value showed that DDD could place a higher emphasis on the highly diverse ensemble, performing more similarly to CDCMS .

For the second drift, which is a recurrent drift, CDCMS was the best one in adapting to this change. This was because all the drifts in Sine1-abrupt are abrupt, which have no transitional period where intermediate concepts could be formed. Thus, CDCMS managed to identify the recurrent drift and recover corresponding models easily. Besides, its highly diverse ensemble also helped to prevent a catastrophic drop in prequential accuracy right after the drift by participating in making predictions with a weight of 49% (the new representative ensemble weighted 46%). OAUE performed poorly for the same reason as the first drift, while DDDperformed satisfactorily despite not being designed for handling recurring concepts.

Sine2-abrupt is the representative case showing CDCMS performed significantly better than other existing approaches. Tables 4.8 and A.1 show that all the concept drifts in Sine2 are severe. Even though the statistical test result did not capture any statistical significant difference in the overall accuracy between CDCMS and DDD, Figure 4.5(b) shows that CDCMS started to perform considerably well after the second drift (around 41k-th time step). The details of CDCMS's behaviour reveal that it considered this drift as a recurrent drift. It then recovered a vast majority of models holding the knowledge of concept Sine-f1, despite the actual underlying concept being Sine-f3. CDCMS behaved similarly at the third drift. A vast majority of models holding the knowledge from Sine-f2 were recovered, despite the actual underlying concept being Sine-f4. This happened because the concepts of Sine-f1 and Sine-f3 are quite similar, while the concepts of Sine-f2 and Sine-f4 are also quite similar (see Table A.1). CDCMS had not seen the concepts of f3 and f4 upon the time it encountered them but managed to exploit relevant past knowledge and successfully recover the prequential accuracy from these two drifts. This reflects CDCMS's ability in dealing with mild severity abrupt drifts.

Short Summary: CDCMS always performed similar or better than other existing approaches in abrupt drift data streams, indicating that it was more robust to and fast recover from multiple types of abrupt drift (RQ1.2.2). This is because the highly diverse

ensemble, containing relevant knowledge to the concept after the drift, helped to prevent a drastic drop in prequential accuracy. Besides, CDCMS handled abrupt drifts better than handling gradual drifts because it would not recover models that learnt from intermediate concepts (RQ1.2.3).

4.2.3.3 Results with Real-World Data Streams

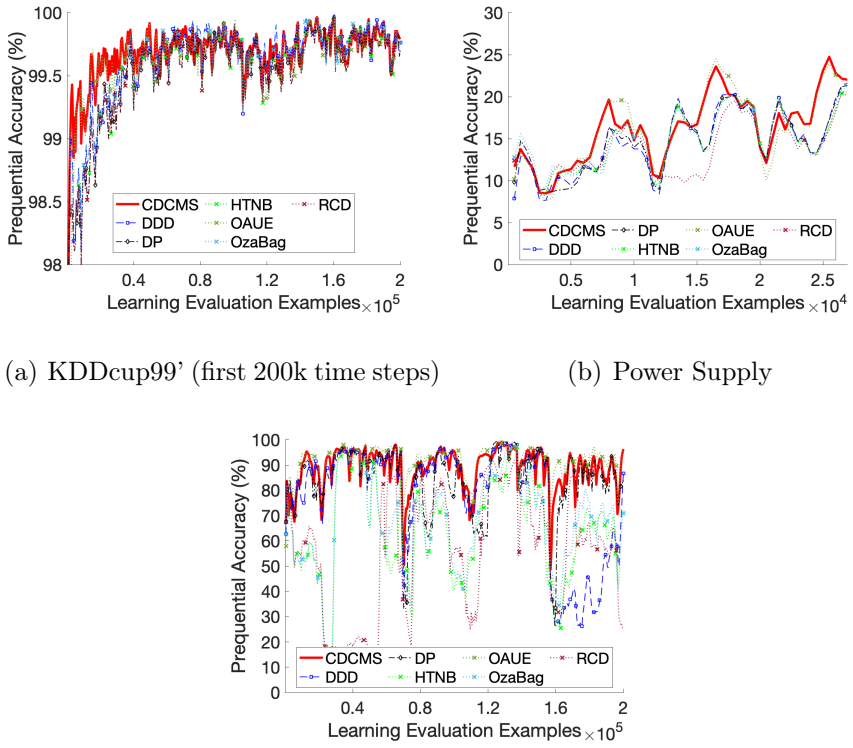
Table 4.14 compares the 30 runs average prequential accuracy of the approaches on the real-world data streams. Figure 4.6 presents the their prequential accuracy over the streams. Note that, due to the length of KDDcup99' and Sensor streams, Figures 4.6(a) and 4.6(c) only present the prequential accuracy of the approaches during first 200k time steps where they performed most differently. Table 4.14 shows that CDCMS performed similarly or significantly better than all other approaches in the real-world data streams.

Table 4.14: Average Prequential Accuracy (Standard Deviation) Across 30 Runs For Real World Data Streams

Data Stream	CDCMS	HTNB	OzaBag	DP	OAUE	RCD	DDD
KDD Cup99	**99.738% (0.191%)**	99.65% (0.291%)	**99.728% (0.278%)**	99.65% (0.291%)	99.663% (1.073%)	99.65% (0.291%)	99.717% (0.277%)
Power Supply	**16.247% (4.115%)**	14.833% (3.359%)	14.907% (3.28%)	14.637% (3.671%)	**16.237% (4.523%)**	13.84% (3.226%)	14.838% (3.503%)
Sensor	**89.38% (8.867%)**	56.283% (17.35%)	71.126% (15.96%)	83.504% (16.628%)	**92.332% (7.306%)**	53.402% (18.793%)	85.838% (12.99%)

- The p-values of Friedman tests are all <2.2E-16.
- Cells with bold text represent the best accuracy (or approaches have no statistical significant difference with best one) in the data stream, based on the p-value (≥ 0.05) in Nemeyi post-hoc test.

Despite Table 4.14 shows that OzaBag is the most competitive approach against



(a) KDDcup99' (first 200k time steps)

(b) Power Supply

(c) Sensor (first 200k time steps)

Figure 4.6: Prequential Accuracy with Real-World Data Streams

CDCMS, Figure 4.6(a) shows that OAUE was actually more competitive against CDCMS at the beginning of KDDcup99 stream, while CDCMS performed well and more stable than other approaches at the beginning of the KDDcup99 stream. The standard deviation of CDCMS's accuracy shown in Table 4.14 also supports this observation. The behaviour of CDCMS on this data stream was similar to its behaviour in the second and third drifts of Agr3-gradual stream. Thus, theoretically, the prequential accuracy of CDCMS and OAUE should have no statistical significant difference in this data stream but Table 4.14 shows that CDCMS performed significantly better. This suggests that prequential accuracy may be a better weighting metric for this data stream than using Mean Squared prediction Error (MSE).

For Power Supply stream, Table 4.14 shows that no statistical significant difference

had been found between the prequential accuracy of CDCMS and OAUE. Figure 4.6(b) shows that the prequential accuracy of CDCMS and OAUE are indeed quite similar throughout the stream. We believe that this is mainly because CDCMS and OAUE both keep updating their main ensemble over time. When compared against other approaches, the same Figure shows that there are periods where HTNB and OzaBag performed better than CDCMS by a small magnitude. However, CDCMS performed better than HTNB and OzaBag in other periods by a large magnitude.

When looking into the details of CDCMS's behaviour in Power Supply stream, it shows that there were four drift detection given at around 2,358th, 2,796th, 11,516th and 20,046th time steps respectively. To compare against OAUE at these four time steps, Figure 4.6(b) also shows that CDCMS reacted accordingly. It recovered early and avoided further drop in prequential accuracy, especially at 20,046th time step. This may suggest that the simple strategy of periodic ensemble committee replacement helped CDCMS to improve the prequential accuracy over time, but on top of that, it may worth to use a drift detector for swift reaction to abrupt drifts. On the other hand, Figure 4.6(b) also shows that CDCMS performed worse than OAUE at around 8,000th and 17,000th time step respectively. The details of OAUE and CDCMS at these two time steps show that OAUE maintained a set of newly created models in its representative ensemble whereas CDCMS still maintained a few old models, which were created a few thousand time steps before these two time steps respectively. Thus, OAUE was able to maintain its accuracy for a short while. Yet, the old knowledge held in the representative ensemble of CDCMS was not helpful at these two time steps, which indicates potential concept drifts but were not captured by the drift detector. Also, the metric used in the ensemble committee replacement of OAUE may be better in capturing these two potential drifts.

For Sensor stream, Table 4.14 shows that CDCMS performed significantly better than other approaches while it performed statistically similar to but slightly worse than OAUE.

When looking closely at Figure 4.6(c), the reason for CDCMS did not achieve better average prequential accuracy than OAUE because it had frequent drops in prequential accuracy, caused by drift detection. With such high frequency of drift detection, they are likely false-positives. CDCMS can normally deal with false-positives drift detection by emphasising old main ensemble after drift detection. However, the large number of false-positives may have caused none of the existing models to represent the current concept well enough, causing the approach struggle to put emphasis.

4.2.4 Experiments to Analyse the Time-Memory-Accuracy Relationship in CDCMS

This section extends the time complexity analysis of CDCMS presented in Section 4.2.1.5 with empirical results. CDCMS was evaluated prequentially with different hyper-parameter values that control its memory requirements and may affect its run-time, based on the five representative synthetic data streams that were analysed in Sections 4.2.3.1 and 4.2.3.1: Sine1-(gradual, abrupt), Sine2-(gradual, abrupt), Agr3-gradual. We recorded the information about the average prequential accuracy, total runtime (CPU seconds) and model cost (RAM-hours) across thirty runs for each hyper-parameter combination per data stream. Each run was performed using the University of Birmingham's BlueBEAR HPC with 8 CPU cores and 8GB memory in a single computing node.

4.2.4.1 Experiment Setup

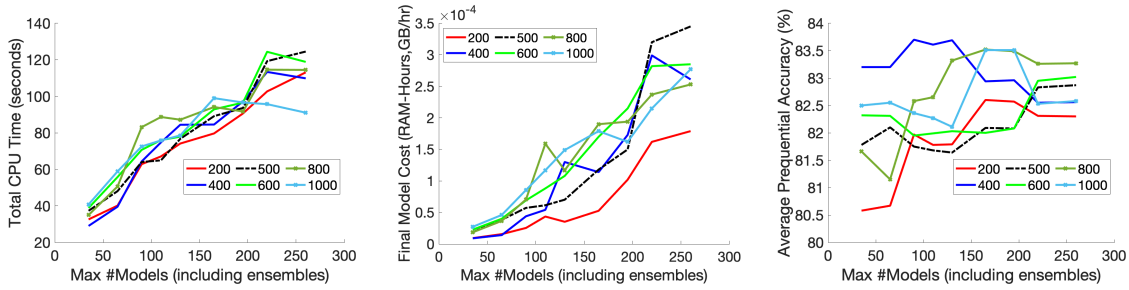
The hyper-parameters of CDCMS that control the memory restrictions and may affect its run-time are the maximum total number of models that CDCMS can hold at a time ($3e+e\times r$) and the size of the most recent window of data (b). The options for these hyper-parameters

were the same as those in Section 4.2.2.2, except that we selected a subset of ensemble size-memory size multiplier (e-r) combinations that lead to roughly equal intervals for the maximum total number of models. In summary, the (e-r) combinations chosen to control the maximum total number of models were 5-4 (35), 5-10 (65), 10-6 (90), 10-8 (110), 10-10 (130), 15-8 (165), 15-10 (195), 20-8 (220), 20-10 (260), where the values in brackets are the resulting total maximum number of models. Other non-memory related hyper-parameters were kept with the values previously selected through hyper-parameter tuning in Section 4.2.2.2

4.2.4.2 Results with Synthetic Data Streams

Figure 4.7 presents the plots of time, memory and accuracy relationships obtained when restricting the maximum total number of models to different values in Agrawal3-gradual stream. Other cases are available in the Appendix D, as they present similar trends unless stated otherwise. Figures 4.7(a) and 4.7(b) show that increasing the maximum number of models ($3e + e \times r$) in general leads to an increase in CDCMS's memory cost (RAM-hours) and total run-time (CPU seconds). This is because CDCMS can hold and is required to process more information. However, the relationship between sliding window size (b), the memory cost and the total run-time was less clear. It depends on the data stream and the total maximum number of models restrictions.

Despite the higher memory cost and run-time, increasing the maximum number of models was in general beneficial to the overall prequential accuracy (see Figure 4.7(c)). This could be the fact that a larger memory can store more concepts which increases the chances of keeping various past knowledge. Thus, it can better handle recurrent drifts and drifts leading to a concept that shares similarities with past concepts. In addition, increasing the size of the sliding window leads to a decrease in the frequency of evaluating the representative



(a) Runtime (b) Memory (c) Accuracy

Figure 4.7: Overall runtime, memory and accuracy of CDCMS for different maximum total numbers of models in Agrawal3-gradual. The different coloured lines represent different sliding window sizes.

ensemble and introducing new base models. Thus, the overall prequential accuracy will be less affected by the initial poor accuracy of new base models when there is no concept drift. In some cases (Sine1-gradual, Agr3-gradual and Sine1-abrupt) using a sliding window size of 400 or 600 led to a subtle decrease in overall prequential accuracy as the maximum number of models increased, showing that larger sliding windows do not necessarily lead to more accurate models. Figure 4.7(c) shows that a sliding window of size 400 can lead to more accurate models than sliding windows of size 500-1000. Thus, larger sliding window sizes did not necessarily lead to better overall prequential accuracy as expected, because different sliding window sizes will be better for different types and frequencies of drift.

In summary, the analysis above shows that increasing the maximum total number of models could slightly improve the prequential accuracy but the trade-offs are the increase in the memory cost and the total run-time.

4.2.5 Summary of Diversity-based Memory Recovery Study

This study continue answers research question 1 by proposing a novel clustering in the model space strategy which is incorporated with the improved diversity-based memory management strategy and the periodic ensemble committee replacement strategy to a novel diversity-based framework called CDCMS. CDCMS can handle recurrent, abrupt, gradual, severe and mild-severe drifts well (RQ1.2.1). The analysis to the experiment results showed that CDCMS achieved similar or better prequential accuracy than existing approaches in both artificial and real-world data streams. It performed particularly well in data streams with abrupt and recurrent drifts, in terms of the speed in recovering prequential accuracy from drifts and the overall predictive performance (RQ1.2.2). The main reason is that the novel clustering in the model space strategy and diversity-based memory management strategy were able to keep, identify and recover relevant past knowledge to handle concept drifts. The highly diverse ensemble created based on such strategies also helped to prevent drastic drops in prequential accuracy after concept drifts (RQ1.2.3). However, CDCMS is disadvantageous when the data stream presents extremely severe gradual recurrent drifts, as it could recover models that learnt from intermediate concepts, slightly hindering the accuracy (RQ1.2.3).

Besides, we also extended the theoretical time complexity analysis of CDCMS with empirical results. It has been shown in Section 4.2.1.5 that the predictive performance of CDCMS can be slightly improved by increasing the total number of models that it can maintain at time. Yet, this increases the memory cost and the total run-time. The same empirical results show that there is no clear relationship between the predictive performance of CDCMS and the size of its sliding window. It depends on the type of the data stream and the frequency of concept drift.

4.3 Summary and Discussion

The main contributions of this chapter include: 1) a preliminary study to compare the impact of memory management strategies on the predictive performance, including a novel memory management strategy based on diversity, 2) a detailed study to investigate the potential of using diversity as a memory-based framework to deal with recurrent, abrupt, gradual, severe and mild-severe drifts. In particular, a novel clustering in the model space strategy was proposed. Both proposed approaches are likely to perform similar or better than existing approaches and they are robust to false-positive drift detection. CDCMS performed better than DP as it was designed to handle recurrent, abrupt, gradual, severe and mild-severe drifts. These studies answer research question 1 posed in Section 1.2.1: "Can diversity be used to create a memory-based framework able to tackle different types of drifts, including recurrent, abrupt, gradual, severe and mild-severe drifts more efficiently and effectively? How well does this framework perform?"

The preliminary study compared the novel diversity-based memory management strategy with other three types of memory management strategies: First in First Out (FIFO), Least Recently Used (LRU), and Elitism. Existing work, such as the work by Minku, White, and Yao (2010) and Minku and Yao (2012), only investigated the impact of different diversity measures to the predictive performance of an ensemble. They did not provide a detailed investigation on why diversity can be helpful. Most importantly, none of the existing work in the literature uses diversity as a memory management strategy to deal with concept drift, especially recurrent drift. Experiments with artificial data streams show that using diversity as memory management strategy usually has more advantages than using FIFO, LRU and Elitism, especially in data streams with recurrent drifts. It is because the diversity-based memory management strategy can maintain various past knowledge in the memory, whereas FIFO, LRU and Elitism cannot. Thus, it helps the learning system to recover from the

recurrent drift effectively and swiftly. However, the relationship between the overall predictive performance and the diversity-based memory strategy is not straightforward, as it also depends on the effectiveness of the model selection strategy in identifying relevant past knowledge from the memory.

The second study proposed a novel clustering in the model space strategy which can effectively and efficiently identify and recover relevant knowledge, and create ensemble with different diversity level to deal with concept drifts. This strategy is also combined with an improved diversity-based memory management strategy and a periodic ensemble committee replacement strategy to be a diversity-based framework, aiming to deal with recurrent, abrupt, gradual, severe and mild-severe drifts. Through the experiments with artificial and real-world data streams, it has been shown that the proposed diversity-based framework is likely to have superior predictive performance over existing approaches in the literature. It is particularly good at dealing with abrupt drift and recurrent drift, while slightly deficient on data streams presenting several extremely severe gradual drifts where intermediate concepts could form. Last but not least, the proposed diversity-framework was shown to be robust to false-positive drift detections.

In addition to the analysis of the predictive performance of the proposed diversity-based framework, a time complexity analysis with empirical results has also been done. It shows that, increasing the maximum number of models that the framework can hold at a time can slightly improve the predictive performance with the price of increasing memory cost and total run-time. The impact of the size of the sliding window to the predictive performance depends on the data stream type and the frequency of concept drifts.

Future work includes investigation to improve the clustering in the model space strategy in handling data streams with multiple extremely severe gradual drifts. Also, one may notice that the dimensionality of the "clustering examples" for clustering in the model space

is quite high (depending on the sliding window size). This could possibly weaken the effectiveness clustering method, thus affecting the effectiveness of the strategy in identify and recover relevant knowledge for concept drift adaptation. Therefore, a future work is needed to devise an alternative to cluster in the model space without relying on a set of high dimensional "clustering examples". Chapter 7 discusses the direction for the future work in more details.

Chapter Five

Diversity-based Framework for Concept Drift Adaptation in Class Imbalanced Data Streams

RQ2: How can memory strategies be used to tackle class imbalanced data streams with concept drifts? When and why are they helpful?

Data stream learning usually not just suffer from concept drift but also class imbalance at the same time. It is particularly difficult to learn the minority class of the post-drift concept because we have very few information (examples) about it. Several approaches from the literature had been proposed, which exploit relevant past knowledge to tackle this difficulty. However, the methods they used usually either store all the past minority class examples in the memory or manage a memory of past base learners based on their creation time or predictive performance to the current underlying concept. These strategies may not always be ideal.

First, storing all the past minority class examples is impractical because the memory

is always finite in real-world applications. Also, the minority class of the data stream may not always be the same because concept drift may affect the class imbalance ratio, such as $P(Y)$ drift. This means that the data stream learning approach has to store all the past examples from both classes in order to handle this type of drift. Besides, age-based or performance-based strategies to manage a memory of past base learners are likely to forget past base learners holding the knowledge that appeared a long period ago or is very different from the current underlying concepts. Thus, data stream learning approaches employing them are unlikely to have relevant past knowledge in the memory to exploit when those past concepts recur. Section 3.3 presents a detailed review to the existing approaches which were proposed to deal with concept drift in class imbalanced data streams.

Chapter 4 shows how diversity can be used to constitute a memory-based framework to handle multiple types of concept drifts in class balanced data streams. In particular, the idea of optimising the chance of having relevant past knowledge to exploit could potential be helpful in dealing with class imbalanced data steams, where the information of the minority class is particularly few after concept drift. However, this idea proposed in Chapter 4 cannot be applied to deal with class imbalanced data streams directly because it relies on the most recent window of examples to measure the diversity level between base learners. This most recent window of examples may not always cover all the decision areas learnt by the base learners, especially when it is class imbalanced. Thus, modifications have to be made before applying the diversity-based memory framework in learning class imbalanced data streams.

A diversity-based framework called Concept Drift Handling Based on Concept Drift Handling Based on Clustering in the Model Space for Class Imbalanced Learning (CDCMS.CIL) is proposed in this chapter, answering the research question 2 presented in Section 1.2.2: *"How can memory strategies be used to tackle class imbalanced data streams with concept drifts? When and why are they helpful?"* In particular, novel methods and designs to allow the clustering in the model space strategy and the diversity-based memory manage-

ment strategy are proposed and used in this diversity-based framework for class imbalanced learning. Three sub research questions have been derived from the research question 2 to guide the study done in this chapter systematically:

- RQ2.1: How can diversity be used as a memory strategy to deal with multiple types of concept drift in class imbalanced data streams?
- RQ2.2: How do the other class imbalance strategies impact the effectiveness of the proposed strategies in terms of helping the system in recovering from concept drift in class imbalanced data streams?
- RQ2.3: How does the proposed approach perform compared to existing approaches on drifting class imbalance data streams? When and why it is successful or unsuccessful?

This chapter focuses on concept drifts affecting the posterior probabilities of classes $P(Y|X)$, i.e., *Real drifts*. The class imbalance ratio is stable throughout the stream. Other types of drift could be considered as future work.

To answer RQ2.1, the diversity-based memory strategies had been modified to rely on sets of micro-clusters of data rather than the most recent window of examples. The use of micro-clusters allows us to create a class balanced set of synthetic examples covering most areas of the input feature space. Thus, we can measure the diversity between base learners without bias towards either class. In contrast, the original version of the strategies used in CDCMS (proposed in Sections 4.2.1.2 and 4.2.1.3) rely on the most recent window of examples, which could potential be class imbalanced and likely covers a certain area of the input feature space. Thus, it is likely to have bias towards a certain area when comparing the diversity between base learners. In short, this novel design allows the diversity-based memory strategies to measure the diversity between base learners more accurately, despite the data stream is class imbalanced.

In order to validate modifications are needed for the framework to learn class imbalanced data streams, we compared CDCMS.CIL with prequential accuracy as weighting metric and using no resampling method against its predecessor, CDCMS. The analysis shows that the proposed diversity-based strategies for class imbalanced learning are more effective in recovering relevant knowledge from the memory than their predecessors when the data stream is severely class imbalanced. However, they also inherited the disadvantage of their predecessor which is slightly prone to data streams with multiple extremely severe gradual drifts.

In the study to answer RQ2.2, it shows that CDCMS.CIL works the best with oversampling in most cases, because most useful knowledge is retained. On the other hand, CDCMS.CIL could only work well with undersampling when the concept drift leads to novel concepts. This is because the aggressive nature of undersampling maybe more advantageous than oversampling in learning new concepts from scratch. As for RQ2.3, we compared the proposed CDCMS.CIL against existing approaches on both artificial and real-world data streams. The analysis shows that CDCMS.CIL is usually one of the best performing approaches. This is because the proposed diversity-based strategies for class imbalanced learning were effective in recovering relevant past knowledge to deal with concept drift under class imbalance. Also, the simple strategies of periodic substitution in the main ensemble and employing a resampling method helped to deal with concept drift under class imbalance when the concept drift detector failed to recognise concept drift.

This chapter is further branched as follows: Section 5.1 describes CDCMS.CIL. Section 5.2 presents the design of the experiments. Section 5.3 presents the analysis of the experiment results to validate CDCMS.CIL. Section 5.4 presents the summary and discussion of this study.

5.1 Description of Concept Drift Handling Based on Clustering in the Model Space for Class Imbalanced Learning (CDCMS.CIL)

This section presents the detailed description of the proposed diversity framework, called CDCMS.CIL which aims to handle multiple types of concept drift in class imbalanced data streams. Algorithm 6 presents the pseudo-code, over-viewing CDCMS.CIL. The main novelty of this framework is the proposed method of measuring diversity level between base learners based on micro-clusters, which allows the diversity-based memory strategies to maintain and to recover relevant knowledge from the memory properly under class imbalance. In contrast, the diversity-based memory strategies in CDCMS rely on the most recent window of examples, which could be class imbalanced. Thus, they are unlikely to maintain and to recover relevant knowledge from the memory properly when the data stream is class imbalanced.

As this work extends the idea of using diversity as an overall framework to handle multiple types of drift in class imbalanced data stream learning, the general working mechanism of CDCMS.CIL is similar to CDCMS (presented in Section 4.2.1). It also has the same mechanism in making prediction to new examples as CDCMS, which is presented in Section 4.2.1.4. Thus, this section mainly focuses on describing the novel designs which enable the key components of the framework to handle drift class imbalanced data streams. Algorithm 6 presents an overview of the proposed framework, where highlighted code fragments with lime/light grey refer to the changes over the previous diversity-based framework which is proposed in Section 4.2. CDCMS.CIL's working mechanism can be generally described as follows.

CDCMS.CIL uses a *memory* R of size $e \times r$ to store past models representing different

Algorithm 6 Concept Drift Handling Based on Clustering in the Model Space for Class Imbalanced Learning - CDCMS.CIL

Hyper-parameters: Base learner (c_{type}), Ensemble Size (e), Memory Size ($e \times r$), Similarity Threshold (θ), Stream Clustering Method (sc), Class Size Fading Factor(α), Weighting Metric (w), Resampling Method ($resampler$), Data Stream (S)

Variables: New Ensemble (E_{nl}), Old Ensemble (E_{ol}), Highly Diverse Ensemble (E_{nh}) New Model (c_n), Memory (R), Clustering Result (C), Cluster ($cluster$), Stream Clustering Methods array ($SC[]$)

$SC[] \leftarrow createStreamClusteringMethods(sc, 2)$ ▷ "2" refers to the number of SC to create.
$E_{nl} \leftarrow E_{nl}.add($ $createLearnerWithInfo(c_{type}, resampler, \alpha, SC[].copy())$)
$E_{ol} \leftarrow null$
$E_{nh} \leftarrow null$
$c_n \leftarrow$ $createLearnerWithInfo(c_{type}, resampler, \alpha, SC[].copy())$
for $s_t \in S$ **do**
    $drift_level \leftarrow DriftDetection(E_{nl}, s_t)$
    if $drift_level == NORMAL$ **then**
        if t mod $b == 0$ **then**
            if t is b time steps after a drift **then**
                $C \leftarrow (R \cup c_n).clusteringModelsBasedOnMicroClusters()$
                $cluster \leftarrow C.getCluster(c_n)$
                if cluster.size() $== 1$ **then**
                    $E_{nl}.add(c_n)$
                else
                    $E_{nl}.add(cluster.getMostTrained(e-1))$
                end if
            else if $|E_{nl}| \geq e$ **then**
                $c_{worst} \leftarrow E_{nl}.getWorstModel()$
                $c_{worst}.resetWeight()$
                $R.saveByDiversityBasedOnMicroClusters(c_{worst}, \theta)$
            end if
            $c_n \leftarrow$ $createLearnerWithInfo(c_{type}, resampler, \alpha, SC[].copy())$
            $E_{nl}.add(c_n)$

25: **end if**
26: $c_n.updateWeight(s_t)$
27: $c_n.trainOnInstance(s_t)$
28: $drift_level_{previous} \leftarrow NORMAL$
29: **else if** $drift_level == DRIFT$ **then**
30: $E_{ol} \leftarrow E_{nl}$
31: $E_{nl}.resetWeight()$
32: $R.saveByDiversityBasedOnMicroClusters(E_{nl})$
33: $E_{nl}.clear()$
34: $E_{nh} \leftarrow createEnsemble(e)$
35: **if** $drift_level_{previous} == NORMAL$ **then**
36: $c_n \leftarrow createLearnerWithInfo(c_{type}, resampler, \alpha, SC[].copy())$
37: $C \leftarrow R.clusteringModelsBasedOnMicroClusters(B)$
38: $E_{nh} \leftarrow C.getRepresentativeModels()$
39: **end if**
40: $E_{nl}.add(c_n)$
41: $c_n \leftarrow createLearnerWithInfo(c_{type}, resampler, \alpha, SC[].copy())$
42: $E_{nh}.resetWeight(); E_{nl}.resetWeight(); E_{ol}.resetWeight()$
43: $drift_level_{previous} \leftarrow DRIFT$
44: **end if**
45: **if** $E_{ol} \neq null$ **then**
46: $E_{ol}.updateWeight(s_t)$
47: **end if**
48: **if** $E_{nh} \neq null$ **then**
49: $E_{nh}.updateWeight(s_t)$
50: **end if**
51: $E_{nl}.updateWeight(s_t)$
52: $E_{nl}.trainOnInstance(s_t)$
53: **end for**

concepts, where e is the predefined maximum size of the ensembles and r is the memory size multiplier. A main ensemble E_{nl} only having a new model is initialised to learn the current

concept (line 2, Algorithm 6). Different from CDCMS, CDCMS.CIL does not employ a sliding window to store incoming training examples for the use of the clustering in the model space strategy and the diversity-based memory management strategy. Instead, it associates the base learners with information representing the knowledge they have learnt. This modification aims to solve the issue that the aforementioned strategies relying on the most recent sliding window would bias towards a certain decision area and a particular base learner, especially when the data stream is class imbalanced. Section 5.1.1 discusses the reason behind this proposed modification in details.

Figure 5.1 presents the class diagram of the augmented base leaner. Each base learner is associated with two stream clustering methods ($SC[]$) to record the information that it has encountered per class, particularly the locations of decision areas in the feature space. Stream clustering methods store these class-wise information as a set of micro-clusters and decide when to expand, remove or merge them. A micro-cluster usually includes vectors of average, linear sum, and squared sum of locally close examples in the feature space, i.e., it is a compression of the locally close examples in the feature space (Silva et al., 2013; Moulton et al., 2019). They will further be used by the diversity-based memory strategies for concept drift adaptation, which are detailed in Sections 5.1.1 and 5.1.2, respectively. Any stream clustering methods from the literature which are able to adapt to concept drift and allow to limit the maximum number of micro-clusters can be held at a given time step. In this study, CluStream (Aggarwal et al., 2003) is considered as it has the aforementioned functionalities.

Limiting maximum number of micro-clusters is important to CDCMS.CIL because the clustering in the model space strategy for class imbalanced learning relies on the prediction by the models to a batch of synthetic data. If the number of micro-clusters is large, this batch of synthetic data becomes large, meaning that the clustering in the model space will be in high dimensional. The details of the the clustering in the model space strategy for class imbalanced learning is presented in Section 5.1.1. Also, it is worth noting that most stream

clustering methods cannot handle data with input attributes in different scales. Thus, the data stream has to be normalised before training CDCMS.CIL to avoid the stream clustering methods biases towards input attributes with large scales.

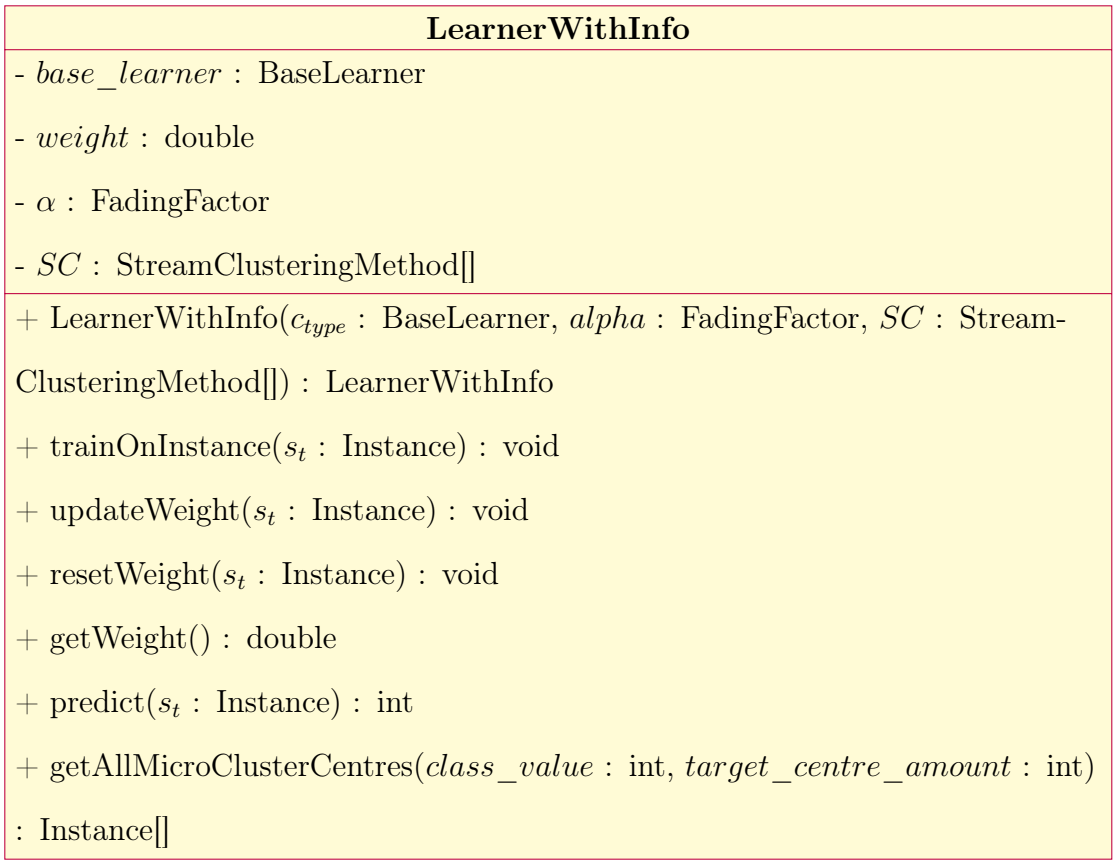


Figure 5.1: Class diagram of Class LearnerWithInfo (The First Method is the Constructor)

Algorithm 7 presents the training procedure of an augmented base learner. It first trains the actual base learner with s_t (line 2, Algorithm 7). After that, it makes a copy of s_t and prepares it to be ready for training the stream clustering method (line 3, Algorithm 7). We denotes the copy of s_t to as $s_t^{Bin,noClass}$. As most stream clustering methods cannot handle nominal attributes directly and they should not take the class attribute into the account when creating and managing the micro-clusters, all the nominal attributes in $s_t^{Bin,noClass}$ have to be converted to binary attributes using one-hot encoding, and the class attribute has to be removed (line 3-4, Algorithm 7). Lastly, the stream clustering method, which corresponds

to the class value of s_t, trains on $s_t^{Bin,noClass}$ (line 5, Algorithm 7).

Algorithm 7 Train LearnerWithInfo on Instance s_t

1: **function** TRAINONINSTANCE(s_t)
2: $base_learner.trainOnInstance(s_t)$
3: $s_t^{Bin} \leftarrow nominalToBinary(s_t.copy())$
4: $s_t^{Bin,noClass} \leftarrow s_t^{Bin}.deleteClassAttribute()$
5: $SC[s_t.classValue()].trainOnInstance(s_t^{Bin,noClass})$
6: **end function**

To reduce the bias towards the majority class, CDCMS.CIL employs a resampling method to train the main ensemble E_{nl}. This resampling method can also be any method from the literature but, for simplicity, only simple oversampling, simple undersampling and no resampling are considered in this study. The investigation on the impact of different resampling methods to the predictive performance of CDCMS.CIL is presented in Section 5.3.2. Figure 5.2 presents the class diagram of the ensemble associated with a resampling method and other additional information, such as weight of the ensemble, time decay factors etc. Algorithm 8 presents the training procedure of the main ensemble. First of all, the time decay class size is updated using the most recent examples s_t (line 2, Algorithm 8) by Equation 5.1.

$$classSize(k)_t = \theta \times classSize(k)_{(t-1)} + [(s_t, c_k)] \tag{5.1}$$

where $classSize(k)_t$ refers to the class size of the k-th class at time step t. $[(s_t, c_k)] = 1$ if the true class label of s_t is c_k, otherwise 0. θ $(0 < \theta < 1)$ is a predefined time decay factor to make the past examples less important to the current class size.

After updating the class size, the algorithm then determines the weight for s_t depending on the resampling method employed and the current class size (line 3-9, Algorithm 8). If simple undersampling is used, the weight of the majority class examples is reduced by the

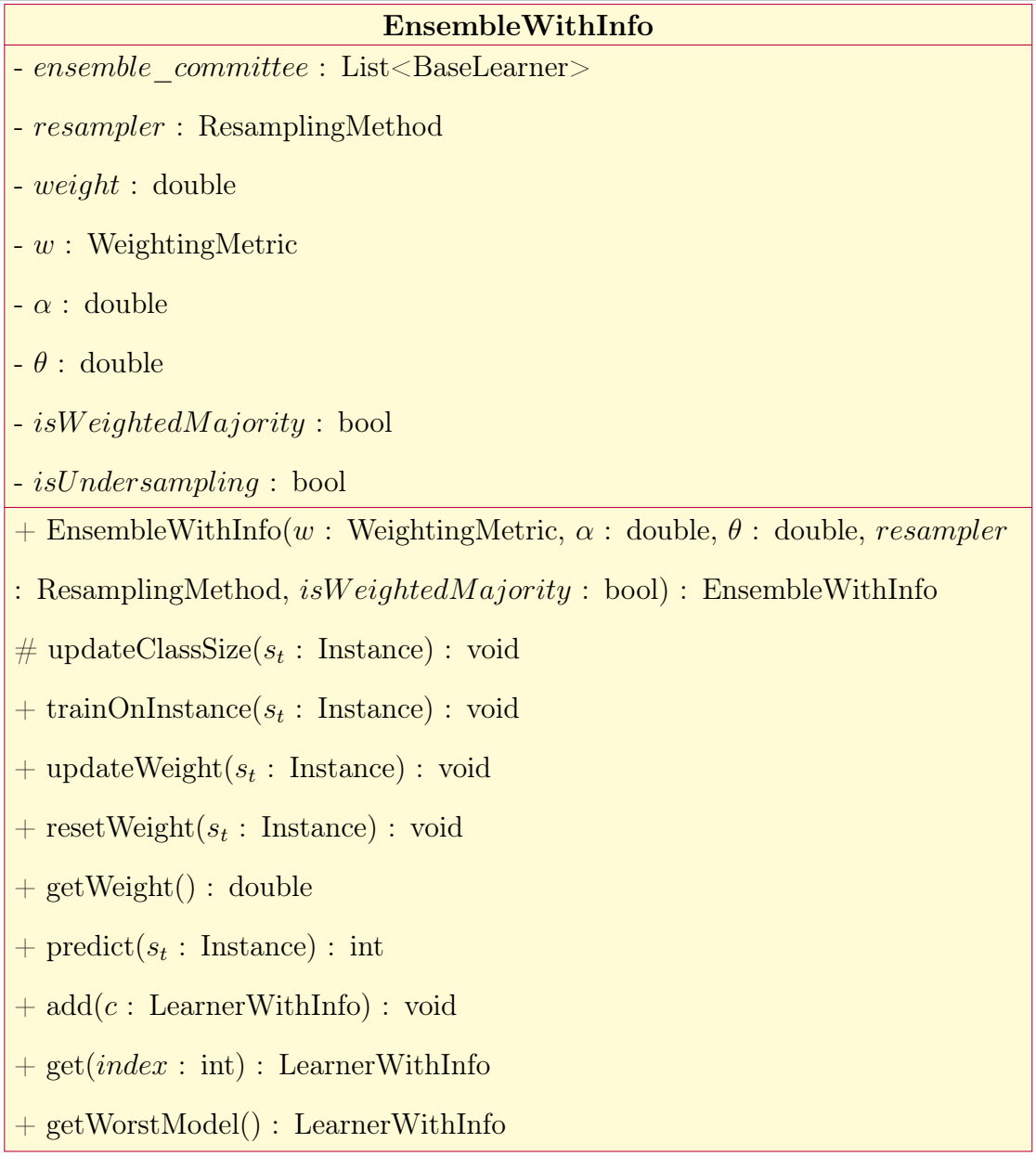


Figure 5.2: Class diagram of Class EnsembleWithInfo (The First Method is the Constructor)

class size ratio of minority class to majority class while the weight of minority class examples remains unchanged (line 4, Algorithm 8). If simple oversampling is used, the weight of the minority class examples is increased by the class size ratio of majority class to minority class while the weight of minority class examples remains unchanged (line 6, Algorithm 8). In the case of no resampling method is used, the weight of the examples remains unchanged (line

Algorithm 8 Train EnsembleWithInfo on Instance s_t

```
1: function TRAINONINSTANCE(s_t)
2:     updateClassSize(s_t)
3:     if resampler == Undersampling then
4:         inst_weight_multiplier ← classSize(MinoirtyClass)/classSize(s_t.classValue())
5:     else if resampler == Oversampling then
6:         inst_weight_multiplier ← classSize(MajorityClass)/classSize(s_t.classValue())
7:     else
8:         inst_weight_multiplier = 1
9:     end if
10:    inst_to_train ← s_t.copy()
11:    inst_to_train.setWeight(s_t.getWeight() * inst_weight)
12:    for c ∈ ensemble_committee do
13:        c.trainOnInstance(inst_to_train)
14:    end for
15: end function
```

8, Algorithm 8). Then, the algorithm set the new weight to a copy of s_t and uses this copy to train the base learners in the main ensemble (line 10-14, Algorithm 8).

CDCMS.CIL employs a drift detector to explicitly handle concept drift by monitoring the predictive performance of the main ensemble (line 7, Algorithm 6). This drift detector can be any existing methods from the literature, for examples, DDM (Gama, Medas, et al., 2004), EDDM (Baena-Garc et al., 2005), ADWIN (Bifet and Gavaldà, 2007), DDM-OCI (S. Wang, Minku, Ghezzi, et al., 2013), PAUC (Brzezinski and Stefanowski, 2017), etc. Upon concept drift detection (line 29, Algorithm 6), the clustering in the model space strategy for class imbalanced learning will be triggered to help the CDCMS.CIL to adapt to concept drift. Similar to CDCMS, the main ensemble E_{nl} is denominated as the old main ensemble E_{ol} (line 30, Algorithm 6) and the base learners in E_{nl} are then stored into the memory to enrich the potential number of concepts in the system's memory, using the diversity-based

memory management strategy for class imbalanced learning (detailed in Section 5.1.2).

Meanwhile, CDCMS.CIL checks whether the last state of the system was "NORMAL" (line 34, Algorithm 6). If it was, CDCMS.CIL triggers the clustering in the model space strategy for class imbalanced learning and take a base learner from each cluster in the model space that had trained with the most number of examples to build a highly diverse ensemble E_{nh} (line 38, Algorithm 6). Similar to CDCMS, this highly diverse ensemble participates in making predictions until the new main ensemble E_{nl} is deemed to confidence to make predictions by itself, preventing a catastrophic drop in predictive performance right after the drift. i.e. E_{nl} performs better than either of E_{ol} and E_{nh} (explained in Section 4.2.1.2). If the last state of the system was not "NORMAL", CDCMS.CIL will not build the highly diverse ensemble E_{nh} and waiting until the last state of the system before the current time step was "NORMAL". Same reason as CDCMS, this checking is to prevent running the clustering method to rebuild E_{nh} in consecutive time steps, caused by false-positive drift detection, as it may detrimentally affect the predictive performance of the approach and the running speed. At the same time, a new main ensemble E_{nl} is created to acquire knowledge from the concept after the drift (line 41, Algorithm 6). This new main ensemble E_{nl} could be helpful in adapting concept drift that leads to a novel concept, as it learns the the concept after the drift from scratch. E_{nl} is initialised with a new model c_n created upon drift detection which has acquired a small portion of knowledge from the concept after the drift (line 36, Algorithm 6). This new model c_n is also be used for the clustering in the model space strategy to identify relevant past knowledge from the system's memory.

After b time steps from the drift detection, the clustering in the model space strategy for class imbalanced learning is triggered again. This time, the new base learner c_n joins the base learners in the memory when clustering in the model. If c_n is clustered with other past base learners in the memory, past base learners from this cluster are recovered to the new main ensemble E_{nl} as a strategy to handle recurrent drifts or drifts leading to a concept that

is similar to any past concept. In the case that the number of base learners in this cluster is larger than the predefined ensemble size e, the first e base learners that were trained with the most number of examples are recovered to E_{nl}. As when c_n does not clustered with any past base learners in the memory, CDCMS.CIL considers this drift leads to a novel concept and the new main ensemble E_{nl} will be built from scratch (line 11-16, Algorithm 6). The details of the clustering in the model space strategy for class imbalanced learning is presented in Section 5.1.1.

In addition to explicit drift handling, CDCMS.CIL also has a strategy to implicitly handle concept drifts (likely gradual drifts) that might not be captured by a drift detector. It periodically evaluates the predictive performance of base learners in the main ensemble E_{nl} and replaces the least performing one c_{worst} with a new base learner c_n (line 9). The metric to evaluate the predictive performance of the base learners can be any from the literature for data stream learning, which takes time steps order into the account. For examples, prequential accuracy, time decay G-Mean, time decay minority class recall, time decay F1-measure, prequential AUC, etc. This metric is also used in the weighted majority votes when CDCMS.CIL makes a prediction. The procedure of CDCMS.CIL making a prediction is the same as CDCMS, presented in Section 4.2.1.2. We investigate the impact of different weighting metrics to the predictive performance of CDCMS.CIL in Section 5.3.2.

After the least performing model c_{worst} is removed from the main ensemble E_{nl}, it is then stored into the memory R using the diversity-based memory management strategy for class imbalanced learning (line 21, Algorithm 6), explained in Section 5.1.2. In the case where the current time step t is b time steps after a drift detection (line 10, Algorithm 6), the main ensemble E_{nl} is replaced by $(e-1)$ models from the memory that have learnt similar concepts to the current underlying concept, based on the clustering in the model space strategy for class imbalanced learning (detailed in Section 5.1.1). This strategy aims to deal with recurrent drifts and drifts leading to a concept that is similar to any past concepts.

5.1.1 Clustering in the Model Space for Class Imbalanced Learning

The original clustering in the model space strategy in CDCMS (proposed in Section 4.2.1.1) projects the decision areas of the base learners to the most recent sliding window of examples. When the data stream is class imbalanced, this sliding window of examples is also likely to be class imbalanced. Thus, they are very unlikely to cover most decision areas of the minority class learnt by the base learners. Thus, the clustering method relying on this window is likely biased towards decision areas of a particular class and a particular base learner. At the result, the clustering result may not truly reflect the similarities and differences between base learners. Figure 5.3 illustrates why projecting the decision areas of the base learners to the most recent sliding window of examples may be problematic when this window is class imbalanced.

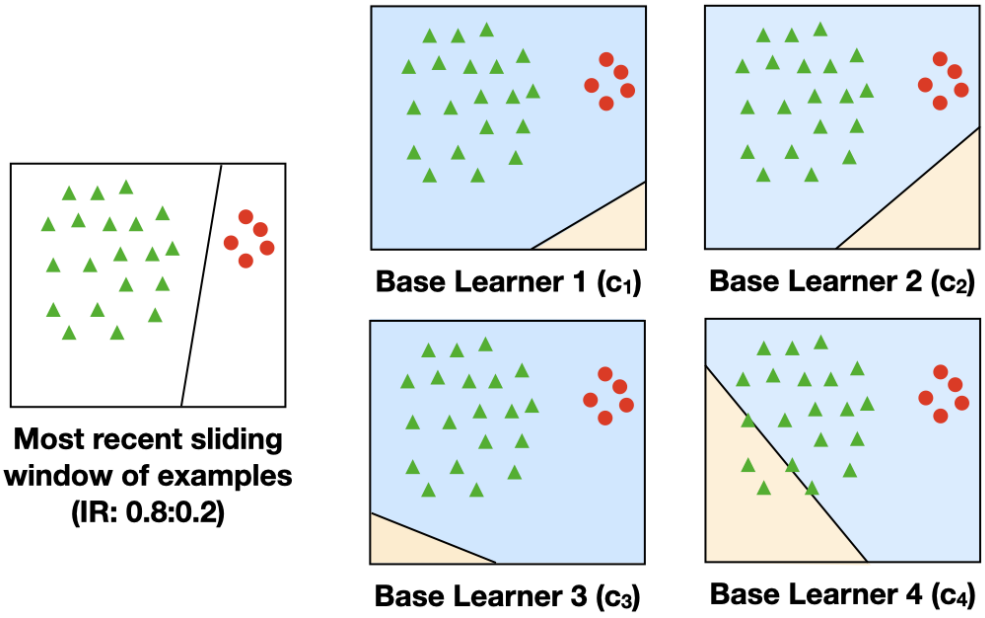


Figure 5.3: Illustration of Comparing Similarities between Base Learners based on the Most Recent Sliding Window of Examples

The green triangles and red circles in Figure 5.3 represent the majority and minority class examples from the most recent sliding window respectively. Light blue areas of base

learners c_1, c_2, c_3 and c_4 represent their learnt decision areas in the majority class whereas the light orange areas represent their learnt decision areas in the minority class. In this case, the examples in the sliding window only cover the learnt majority class areas of c_1, c_2 and c_3. If we cluster c_1, c_2, c_3 and c_4 based on their predictions to the sliding window, c_1, c_2, c_3 are likely to be determined as in the same cluster in the model space while c_4 is determined as in another cluster in the model space. Obviously, this clustering result does not reflect their similarities correctly because we can see that c_1 and c_2 should belong to one cluster in the model space, while c_3 and c_4 should belong to another cluster in the model space. Thus, we can see that clustering in the model space relying on the most recent sliding window of examples may be problematic because it may not always consider the decision areas learnt by each base learner in both classes fairly.

As we can see, this issue can be considered as related to intra-class imbalance issue, where where some decision regions of the classes are underrepresented. Therefore, simple oversampling the minority class examples or undersampling the majority class examples in this most recent sliding window could not solve this issue because they do not change the feature space that this most recent window covers. These methods only balance the emphasis between the green triangles' and red circles' areas. Also, it is worth noting that, this issue can happen in class imbalanced data stream. Yet, class imbalanced data streams would exacerbate this issue as there exists particularly few number of examples in the minority class than the majority class.

As we can see from the above, modifications are needed to enable the clustering in the model space strategy to work under class imbalance. Specifically, an alternative is needed to replace the most recent sliding window of examples such that the clustering method can compare the similarities between base learners in both classes fairly, despite the data stream is class imbalanced. In this section, we propose to create a class balanced synthetic data set, using the centres of micro-clusters (i.e. the vector of average) that are associated to each base

learner, to be this alternative. As these micro-clusters are the abstraction of the decision areas learnt by each base learner, thus, the synthetic data set created based on them is likely to cover all the decision areas of the base learners in both classes. Therefore, clustering the model space based on this synthetic data set is less likely biased towards a decision area of a particular class and a particular base learner. Figure 5.4 provides an illustrative example to explain why the centre of micro-clusters enable the clustering in the model space to work under class imbalance.

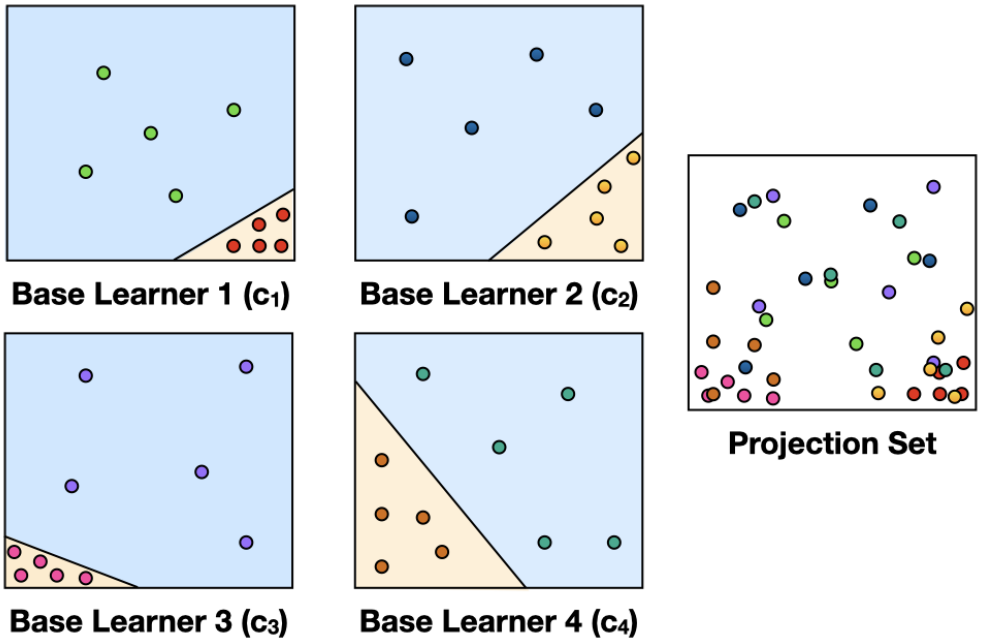


Figure 5.4: Illustration of Comparing Similarities between Base Learners based on Micro-Clusters

In Figure 5.4, circles or dots in different colours represent the micro-clusters which are the abstraction of the decision areas learnt by the base learners c_1, c_2, c_3 and c_4. If we use the centre of these micro-clusters (i.e. the vector of average) to create a class balanced synthetic data set (i.e. the projection set in Figure 5.4, which will be formally defined later in this section), this data set covers the decision areas learnt by the base learners well. Thus, clustering c_1, c_2, c_3 and c_4 based on their predictions to this projection set will always

consider the decision areas learnt by each base learner in both classes fairly, which means the result of clustering in the model space can truly reflect the similarities or differences between c_1, c_2, c_3 and c_4.

Over the next paragraphs, we discuss the details of creating the project set and how it is used by the clustering in the model space strategy. Also, we name this modified strategy as clustering in the model space strategy for class imbalanced learning to distinguish it from it predecessor proposed in Section 4.2.1.1. Figure 5.5 illustrates its working flow.

When the strategy is triggered, the base learners in the memory are first required to retrieve the centres of their micro-clusters to create a class balanced synthetic data set, using the method "getAllMicroClusterCentres" shown in Figure 5.1 (Step 1 in Figure 5.5). This data set is assumed to be covering most decision areas learnt by the base learners, thus, it can be used by the clustering in the model space strategy to compare the similarities between base learners in both classes fairly.

To prevent clustering in the model space from biasing towards a particular class and a particular base learner, the base learners are required to retrieve an equal number of micro-clusters' centre. However, the number of micro-clusters associated with base learners may not always equal. For example, a given base learner is likely to have more micro-clusters in the majority class than the minority class. Also, a base learner may have more micro-clusters than the others if it learnt from the data stream for a longer period. Thus, to solve this issue, CDCMS.CIL "oversamples" the centres of the micro-clusters for each set. Algorithm 9 presents the procedure of retrieving the centres of the micro-clusters of a base learner by class and it works as follows.

CDCMS.CIL first searches for the set with the largest number of micro-clusters and records the size of this set to as $target_centre_amount$. To retrieve a set of micro-clusters' centre with size $target_centre_amount$ by class per base leaner, CDCMS.CIL

Algorithm 9 Retrieve Centre of Micro-clusters associated with the Base Learner

```
1: function getAllMicroClusterCentres(class_value, target_centre_amount)
2:     microClusters[] ← SC[class_value].getMicroClusteringResult()
3:     k = ⌊target_centre_amount / microClusters.length⌋ × microClusters.length
4:     for i ∈ range(0..k) and microClusters.length > 1 do
5:         centre ← microClusters[(i mod microClusters.length)].getCentre()
6:         centre_insts.add(createInstance(centre, class_value))
7:     end for
8:     for j ∈ range(0..∞) and centre_insts.length < target_centre_amount do
9:         if random.nextBoolean() == True then
10:            centre ← microClusters[(j mod microClusters.length)].getCentre()
11:            centre_insts.add(createInstance(centre, class_value))
12:        end if
13:    end for
14:    return centre_insts
15: end function
```

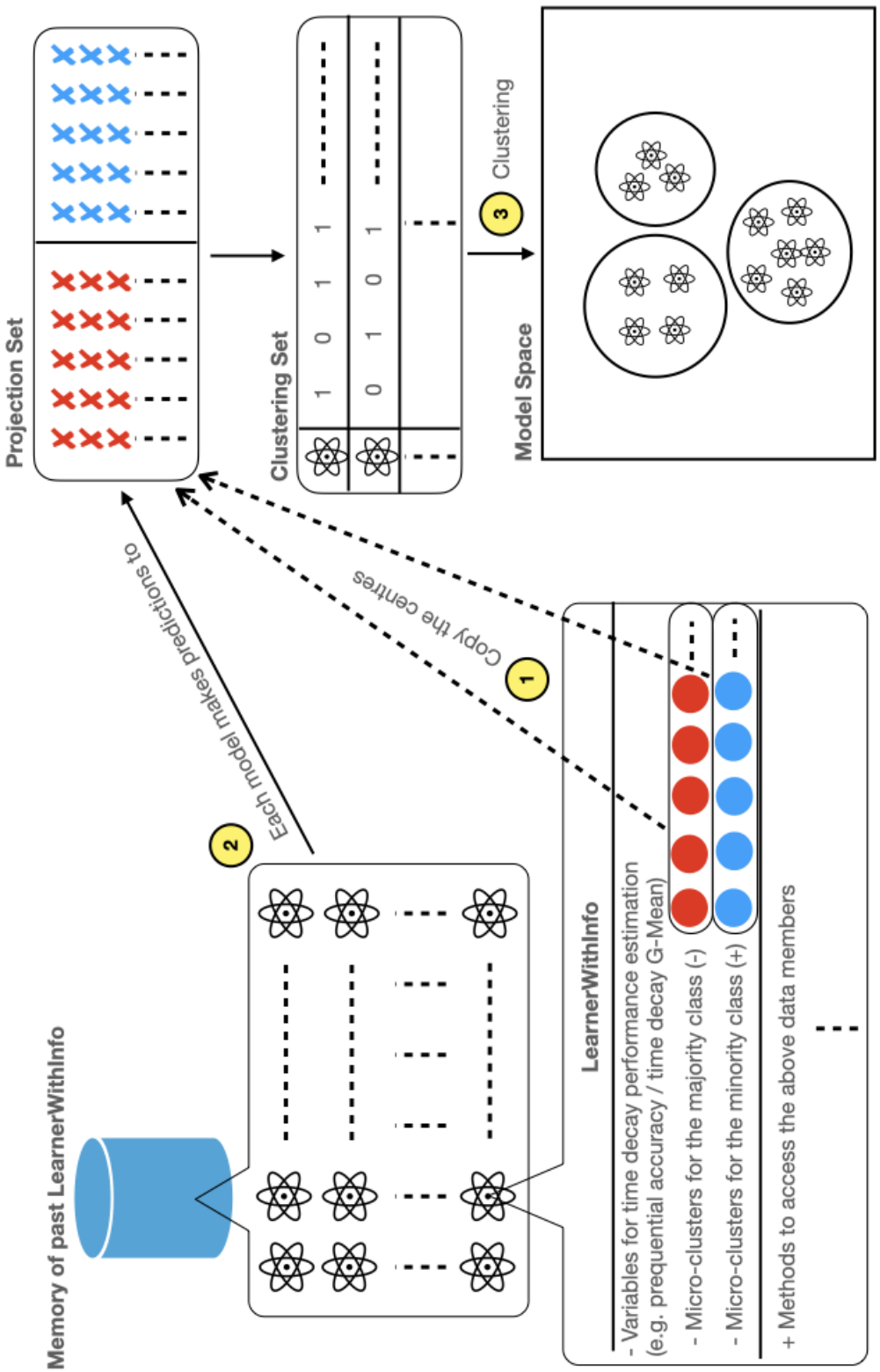


Figure 5.5: Illustration of the Clustering in the Model Space Strategy for Class Imbalanced Learning

retrieves the micro-clusters from the stream clustering method corresponding to the class (line 2, Algorithm 9). We denote this retrieved set of micro-clusters as $mClusters_{retrieve}$. If $|mClusters_{retrieve}| \leq target_centre_amount$, CDCMS.CIL first retrieves the centre of all micro-clusters in $mClusters_{retrieve}$ for $x = \lfloor \frac{target_centre_amount}{|mClusters_{retrieve}|} \rfloor$ times and create synthetic examples based on these centres and the class label (line 3-7, Algorithm 9). The remaining $(target_centre_amount - |mClusters_{retrieve}| \times x)$ spaces or when $|mClusters_{retrieve}| > target_centre_amount$, CDCMS.CIL randomly retrieves the centre of micro-clusters from $mClusters_{retrieve}$ and create synthetic examples based on these centres and the class label (line 8-13, Algorithm 9). That said, the case of $|mClusters_{retrieve}| > target_centre_amount$ is not going to happen because $target_centre_amount$ is the size of the set with the largest number of micro-clusters, which is always larger than or equal to any given set of micro-clusters.

CDCMS.CIL uses the aforementioned procedure to retrieve all centres of the micro-clusters associated with base learners in the memory to create a data set. We denote this data set as the "projection set" ($S_{proj.}$) while the synthetic examples inside this projection set are denoted as "projection examples" ($s^{proj.} \in S_{proj.}$). The i-th projection example can be formally written as $s_i^{proj.} = (x_i, y_i)$, where $(x_i, y_i) \in X \times Y$. X is the input feature space with n dimensions while Y is the output class space. $Y = \{0, 1\}$ as we only consider binary classification in this work. As the projection examples are created based on the micro-clusters and the micro-clusters were created based on the training examples from the data stream, thus, the projection examples have the same number of dimensions as the training examples. The projection set can be formally written as $S_{proj.} = \{(x_i, y_i)\}_{i=1}^{M}$ where $M = |Y| \times target_centre_amount \times L$ is the size of the projection set and L is the number of base learners to be clustered.

In the next step (Step 2 in Figure 5.5), CDCMS.CIL requests the base learners in the memory to make predictions to the projection set. If this strategy is triggered to identify

recurring concept after b time steps from the drift detection, the new base learner created upon last drift detection is also requested to make predictions to the projection set. The correctness of the predictions are then recorded to create a "clustering set" (S_{clust}). Each example inside the clustering set represents a set of correctness of predictions by a given base learner to the projection set. We denote these example as "clustering example" ($s_{clust} \in S_{clust}$). Thus, if there are M projection examples, the number of dimensions of the clustering examples is M. i.e. $s_{clust} = \{0,1\}^M$. A value of 1 refers to a correct prediction while a value of 0 refers to an incorrect prediction. Based on this definition of the clustering examples, the clustering set can then be written as $S_{clust} = \{\{0,1\}^M\}^L$, where L is the number of base learners to be clustered.

Finally, CDCMS.CIL performs clustering in the model space based on the clustering set (Step 3 in Figure 5.5). Consider the aforementioned clustering set $S_{clust} = \{\{0,1\}^M\}^L$, a clustering model $g : \{0,1\}^M \mapsto C$ is built, optimising the objective (cost) function $q(\theta, C)$. $C : \{1,2,3,...,N\}$ is a set of cluster labels where N can be decided via cross validation if the clustering method used could not decide it automatically. θ_{c_i} is a set of parameters to describe the cluster with label $c_i \in C$. As each s_{clust} is the projection of a base learner's decision areas, clustering $s_{clust} \in S_{clust}$ approximates clustering L base learners in the model space indirectly. i.e. $g \approx h : f \mapsto C$, where $f : X \mapsto Y$ is a function representing a base learner. This method of approximating clustering in the model space has the advantage of clustering arbitrary type of base learner.

Depending on the situation triggering this strategy, the result of clustering will then be used to create a highly diverse ensemble upon concept drift detection, or to identify recurring concept after b time steps from the drift detection. These procedures are explained in the general description of CDCMS.CIL presented in Section 5.1. It is also worth noting that the dimensionality of the clustering set could potentially be high if the size of the projection set is large. To prevent this, the maximum size of the micro-cluster set per class

per base learner has to be limited. In this work, this limit was set to ten micro-clusters. Improvements to avoid clustering in the model space on a high dimensional clustering set is planned as a future work.

5.1.2 Diversity-based Memory Management Strategy for Class Imbalanced Learning

Similarly to the clustering in the model space strategy, the diversity-based memory management strategy in CDCMS (proposed in Section 4.2.1.3) also relies on the most recent sliding window of examples. Thus, it suffers the same issue as the clustering in the model space strategy. The Q-statistics computed based on this window to measure the diversity level between two base learners is likely biased towards a decision areas of a particular class and a particular base learner. Similarly, simple oversampling the minority class examples or undersampling the majority class examples in this most recent sliding window could not solve this issue because they do not change the decision areas that this most recent window covers. That said, this issue is exacerbated when this most recent sliding window is severely class imbalanced.

Following the modifications to clustering in the model space strategy in Section 5.1.1, we also proposed to modify the diversity-based memory management strategy to rely on the micro-clusters rather than the most recent sliding window. We name this modified memory management strategy to as diversity-based memory management strategy for class imbalanced learning. Its working flow is illustrated in Figure 5.6.

The diversity-based memory management strategy for class imbalanced learning is triggered periodically to decide which past base learner to remove from the memory or to delete the least performing base learner c_{worst} which was just removed from the main

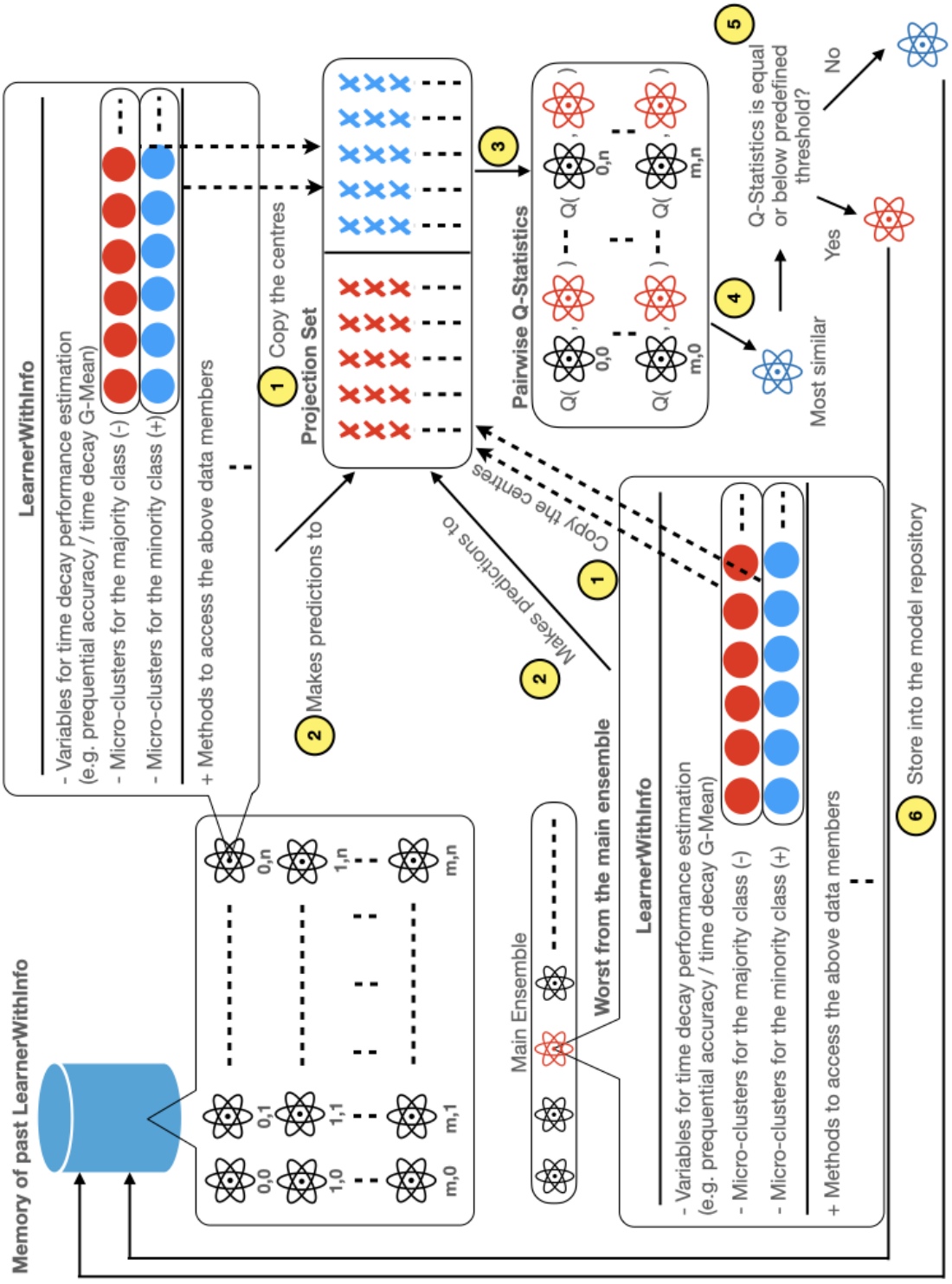


Figure 5.6: Illustration of the Diversity-based Memory Management Strategy for Class Imbalanced Learning

ensemble. It will only be triggered for this purpose when both the main ensemble and the memory of past model reach their predefined maximum size. When the memory has not reached the predefined maximum size, c_{worst} will always be stored into the memory.

When this strategy is triggered, CDCMS.CIL first pairs the least performing base learner c_{worst} from the main ensemble with a past model in the memory $c_{i,j}^{memory}$. It then requests this pair of models to retrieve centres of their classwise micro-clusters to form a "projection set", using the procedure described in Section 5.1.1 and setting $target_centre_amount$ to be the largest number of micro-clusters among four sets (Step 1 in Figure 5.6). These two models are then required to make predictions to this projection set (Step 2 in Figure 5.6). Two sets of prediction correctness are then obtained which represents the decision areas corresponding to c_{worst} and $c_{i,j}^{memory}$ respectively. The Q-Statistics (Yule, 1900) between c_{worst} and $c_{i,j}^{memory}$ is then computed based on these two sets of correctness, using Equation 5.2.

Consider two models c_i and c_j, Yule's Q-Statistics (Yule, 1900) between them is computed as follows.

$$Q_{i,k} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}} \tag{5.2}$$

where $N^{a,b}$ is the number of examples where the classification by c_i is a and the classification by c_j is b, a value of 1 represents a correct classification while a value of 0 represents a misclassification. Q varies between 1 and -1. Models that tend to classify the same examples correctly will have positive values of Q, whereas models that tend to classify different examples correctly will have negative values of Q.

CDCMS.CIL iteratively use the aforementioned procedure to compute the Q-Statistics between c_{worst} and each $c_{i,j}^{memory}$ from the memory, where $0 \leq i \leq m$ and $0 \leq j \leq n$, to obtain a list of values of pairwise Q-Statistics. It then search for the past model that has the largest

value of Q-Statistics with c_{worst}, i.e. the past model that is the most similar to c_{worst} (Step 4 in Figure 5.6). We denote this past model as c_{sim}. CDCMS.CIL then checks if c_{worst} and c_{sim} are similar enough by comparing the Q-Statistics against the predefined threshold (Step 5 in Figure 5.6). If that is the case and c_{worst} is trained with more examples than c_{sim}, c_{worst} will be kept in the memory while c_{sim} will be discarded. For other situations, c_{worst} will be discarded while c_{sim} is preserved in the memory (Step 6 in Figure 5.6).

5.2 Design of the Experiment

This section presents the experiment design in this comparative study with three analytical perspectives, showing CDCMS.CIL is an answer to research question 2 , posed in Section 1.2.2.

In the first perspective, we compared the performance of the base version of the proposed CDCMS.CIL which uses prequential accuracy as the weighting metric and no resampling method against the CDCMS. This is to show the advantages or disadvantages of the redesigned diversity-based memory strategies in learning drifting class imbalanced data streams on their own, answering the RQ2.1. In the second perspective, we compared the performance of CDCMS.CIL with different choice of weighting metric and resampling methods to answer RQ2.2. This is to investigate whether using a class imbalance insensitive weighting metric in the main ensemble and a resampling method are necessary to support the proposed CDCMS.CIL and how they impact the predictive performance of CDCMS.CIL. In particular, we investigated whether the effectiveness of the novel diversity-based memory strategies will be affected by the choice of weighting metric and resampling method. In the last perspective, we validated CDCMS.CIL by comparing its performance against six existing data stream approaches, answering the RQ2.3. These six approaches are GH-VFDT$_{\mathrm{d}}$,

HD-VFDT$_{\text{d}}$, Online Bagging$_{\text{d}}$, OOB$_{\text{d}}$, UOB$_{\text{d}}$ and CDCMS. The approaches' name with a subscript "d" were not designed to deal with concept drift originally, so we used a wrapper which allows them to employ a drift detector. Their system will be reset upon concept drift detection. The decision of whether to use or which drift detector to use is based on the result of hyper-parameter tuning, detailed in Section 5.2.2.

Massive Online Analysis (MOA) (Bifet, Holmes, et al., 2010) was chosen to be the platform to perform the experiments. Section 5.2.1 presents the data streams used in the experiments. Section 5.2.2 details the hyper-parameter tuning and experiments procedure.

5.2.1 Data streams

Tables 5.1 and 5.2 present a summary of the artificial and real-world data streams used in the experiments, respectively. Table 5.3 shows the confusion matrices between the concepts before and after each drift in the artificial data streams. These data streams have been normalised in advance as required by the proposed approach, explained in Section 4.2.1.

We adopted the twenty artificial data streams presented in Section 4.2.2 but enforced four different class imbalance ratios: 0.5:0.5, 0.7:0.3, 0.8:0.2 and 0.9:0.1. Thus, a total of $20 \times 4 = 80$ artificial data streams have been used. As for the experiments with real-world data streams, we adopted seven streams which have been discussed by Souza et al. (2020). Their background are as follows.

The Airline stream (Ikonomovska, Gama, and Džeroski, 2011) is originally from Data Expo competition 2009. It consists of a large number of flight arrival and departure records for all the commercial flights within the USA, October 1987 to April 2008. The version of the Airlines stream used in this work was clean and sorted according to arrival/departure date, by Ikonomovska, Gama, and Džeroski (2011). The learning task is to predict whether

Table 5.1: Summary of Artificial Data Streams

Stream	#Examples	#Nominal Attributes	#Numeric Attributes	Concept Sequence	Imbalance Ratio (IR)
Sine1	60k	0	4	f3→f4→f3	
Sine2	100k	0	4	f1→f2→f3→f4→f1	
Agrawal1	100k	3	6	f1→f3→f4→f7→f10	
Agrawal2	120k	3	6	f7→f4→f6→f5→f2→f9	{0.5:0.5 /
Agrawal3	120k	3	6	f4→f2→f1→f3→f4	0.7:0.3 /
Agrawal4	100k	3	6	f1→f3→f6→f5→f4	0.8:0.2 /
SEA1	100k	0	3	f5→f3→f1→f2→f4	0.9:0.1}
SEA2	100k	0	3	f5→f1→f4→f3→f2	
STAGGER1	80k	3	0	f1→f2→f3→f2	
STAGGER2	80k	3	0	f2→f3→f1→f2	

- Total number of attributes = #Nominal attributes + #Numeric attributes + Class attribute.
- Coloured rows (lime / light grey) highlight data streams with recurring concepts. f*n* represents the n-th function of the stream type, i.e., f1 in Agrawal1 refers to the first function of the Agrawal generator.

a given flight will be delayed based on the information of the scheduled departure.

The NOAA stream (Elwell and Polikar, 2011) contains weather records collected over five decades (1949-1999) at the Offutt Air Force Base in Bellevue, Nebraska. Daily weather measurements include a variety of features, such as temperature, pressure, wind speed, etc. Precipitation indicators and other weather-related events are also included. The learning task of NOAA stream is to predict whether the next day will rain or not, based on the weather measurements.

Forest Covertype (Covtype) stream (Blackard and Dean, 1999) contains the cartographic information about the forest cover type of 30 × 30-meter cells, which includes elevation, slope, wilderness, soil type etc. The actual forest cover type for a given cell was determined from the US Forest Service (USFS) Region 2 Resource Information System (RIS)

Table 5.2: Summary of Real-World Data Streams

Stream	#Examples (Pre)	#Examples (Actual)	#Nom. Attr.	#Num. Attr.	Imbalance Ratio (Pre)	Imbalance Ratio (Actual)
Airlines	53,938	485,445	4	3	0.646:0.354	0.544:0.456
NOAA	1,815	16,344	0	8	0.698:0.303	0.685:0.315
Covtype$_{(c_1=\{1\text{-}6\})}$	58,101	522,911	2	10	0.785:0.215	0.619:0.381
Covtype$_{(c_1=1)}$	58,101	522,911	2	10	0.595:0.405	0.524:0.476
Covtype$_{(c_1=2)}$	58,101	522,911	2	10	0.963:0.037	0.936:0.064
Covtype$_{(c_1=3)}$	58,101	522,911	2	10	0.963:0.037	0.999:0.001
Covtype$_{(c_1=4)}$	58,101	522,911	2	10	0.958:0.042	0.987:0.014
Covtype$_{(c_1=5)}$	58,101	522,911	2	10	0.963:0.037	0.971:0.029
Covtype$_{(c_1=6)}$	58,101	522,911	2	10	0.963:0.037	0.965:0.035
INSECTS$^{inc.}$	45,204	406,840	0	33	0.899:0.101	0.905:0.095
INSECTS$_{abr.}$	35,527	319,748	0	33	0.912:0.088	0.907:0.093
INSECTS$^{inc.}_{grad.}$	14,342	128,981	0	33	0.921:0.079	0.899:0.101
INSECTS$^{inc.}_{abr.\ re.}$	45,204	406,840	0	33	0.895:0.105	0.905:0.095
INSECTS$^{inc.}_{re.}$	45,204	406,840	0	33	0.895:0.105	0.905:0.095

- Total number of attributes = #Nominal attributes + #Numeric attributes + Class attribute.
- "Pre" refers to hyper-parameter tuning sets (i.e. the first 10% of the original data set). "Actual" refers to actual experiment sets (i.e. the remaining 90% of the original data set).
- Covtype$_{(c_1=x)}$: "c_1=x" refers to the class 1 is the class x in the original data set while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream. "c_1=$\{x_0 - x_n\}$" refers to the class 1 is the class x_0-x_n in the original data set combined while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream.
- For all INSECTS data streams, "ae-albopictus" is the class 1. "inc." refers to incremental, "abr." refers to abrupt, "grad" refers to gradual, and "re." refers to recurring.

data. The learning task is to predict the forest cover type based on the given cartographic information of a 30 × 30-meter cell. Forest Covertype stream is a multi-classification problem, where there were seven forest cover types. To make it suitable for this study, it has been converted into seven binary classification streams. Each of them takes one of the forest cover types as one class while combining other forest cover types to be the other class. Thus,

Table 5.3: Confusion Matrices between Concepts Before and After Drifts

Concept Drift	00	01	10	11
Sine1				
f3→f4	0%	56.44%	43.57%	0%
f4→f3	0%	56.44%	43.57%	0%
Sine2				
f1→f2	0%	46.05%	53.95%	0%
f2→f3	0%	56.44%	43.57%	0%
f3→f4	0%	56.44%	43.57%	0%
f4→f1	8.23%	37.82%	35.34%	18.61%
Agrawal1				
f1→f3	25.88%	41.27%	12.61%	20.23%
f3→f4	20.54%	32.95%	17.89%	28.62%
f4→f7	16.82%	21.62%	31.53%	29.72%
f7→f10	8.23%	37.82%	35.34%	18.61%
Agrawal2				
f7→f4	16.82%	21.62%	31.85%	29.72%
f4→f6	21.46%	16.97%	18.57%	43.00%
f6→f5	15.97%	24.00%	24.06%	35.97%
f5→f2	15.41%	23.08%	24.56%	36.95%
f2→f9	16.93%	21.56%	31.76%	29.75%
Agrawal3				
f4→f1	28.20%	10.30%	10.24%	51.27%
f1→f2	25.88%	41.27%	12.61%	20.23%
f2→f3	33.77%	33.38%	19.72%	13.13%
f3→f4	20.54%	32.95%	17.89%	28.62%

Concept Drift	00	01	10	11
Agrawal4				
f1→f3	33.77%	33.38%	19.72%	13.13%
f3→f6	21.14%	32.35%	18.89%	27.61%
f6→f5	15.97%	24.00%	24.06%	35.97%
f5→f4	15.37%	23.07%	24.60%	36.97%
SEA1				
f5→f3	8.02%	16.52%	0%	75.46%
f3→f1	24.54%	7.43%	0%	68.03%
f1→f2	31.97%	0%	8.53%	59.50%
f2→f4	40.50%	0%	4.61%	54.89%
SEA2				
f5→f1	8.02%	23.95%	0%	68.03%
f1→f4	31.97%	0%	13.14%	54.89%
f4→f3	24.54%	0%	20.57%	54.89%
f3→f2	24.54%	15.96%	0%	59.50%
STAGGER1				
f1→f2	3.71%	7.45%	51.89%	36.95%
f2→f3	37.06%	18.54%	29.58%	14.82%
f3→f2	37.06%	18.54%	29.58%	14.82%
STAGGER2				
f2→f3	37.06%	18.54%	29.58%	14.82%
f3→f1	0%	11.16%	66.64%	22.20%
f1→f2	3.71%	7.45%	51.89%	36.95%

we have seven Forest Covertype streams in this study, as shown on Table 5.2. They allow us to evaluate how well a given data stream learning approach performs in learning each of the seven classes.

INSECTS streams (Souza et al., 2020) were constructed using a smart trap with optical sensor to collect the flying data of three different species of insects in a non-stationary environment for around three months. The temperature of the data collection environment was controlled to simulate concept drifts. The data collection were performed at the University of São Paulo, Brazil. These data streams consists of information extracted from the audio file of the sound produced by the insects, such as wing-beat frequency, harmonics positions etc. INSECTS streams originally have six classes: three species of mosquitoes with two genders. We converted them into binary classification tasks by combining classes belonging to species of ae-albopictus as the minority class while combining the rest of the classes as majority class. Also, it has to note that Souza et al. (2020) originally proposed seven INSECTS streams but we only adopted six of them which contain concept drifts and left the INSECT-out-of-control stream unused as it does not contain any concept drift.

5.2.2 Experiment Setup

This section presents the general experiment setup for the comparative studies, which includes the procedure of hyper-parameter tuning and the experiments with artificial and real-world data streams. This setup applies to all the comparative studies which answer RQ2.1, RQ2.2, and RQ2.3 respectively. In particular, the third study to answer RQ2.3 compares the proposed approach against several existing approaches from the literature. These compared approaches and their reasons of choice are as follows.[1]

- *GH-VFDT$_d$ and HD-VFDT$_d$* (Lyon et al., 2014): to compare against two algorithm-

[1]Note that approaches' name with a subscript "d" were not designed to deal with concept drift originally, except OOB and UOB can deal with drifts affecting $P(Y)$ only. Hence, we used a wrapper which allows them to employ a drift detector. Their system will be reset upon concept drift detection. However, experiments with their original version were also performed. These results are presented in Tables G.1-G.3

level single learner approaches that only aim to deal with class imbalance in data stream learning. These two approaches are also used as baselines.

- *Online Bagging$_d$* (Oza, 2005): to compare against a baseline ensemble approaches that was not originally designed to concept drift and class imbalance in data stream learning.
- *OOB$_d$ and UOB$_d$* (S. Wang, Minku, and Yao, 2015): to compare against two simple yet efficient approaches that were originally designed for class imbalanced data stream learning and able to deal with concept drifts affecting $P(Y)$.
- *CDCMS* (Chiu and Minku, 2022): to compare against the predecessor of the proposed approach to validate the proposed modifications are necessary for the diversity based framework to deal with drifting class imbalanced data streams.

Following the procedure in Chapter 4 to tune the hyper-parameters of the approaches. For artificial streams, we created four additional streams, one for each generator: Sine, Agrawal, SEA and STAGGER. These streams have random sequences of concepts, width and position of the concept drifts. We refer to these additional streams as "hyper-parameter tuning streams". To produce each artificial example, the generators sample the input feature space uniformly and then determine the class label of the example based on the concept function. Thus, the hyper-parameter tuning streams do not necessarily class balanced. For real-world data streams, the first 10% of examples in each stream are used as the hyper-parameter tuning stream. The set of values that leads to the highest ten runs average time decay G-Mean on the hyper-parameter tuning streams were then used in the experiments with the corresponding data streams presented in Section 5.2.1. In the experiments, the approaches were compared using thirty runs average time decay G-Mean. Results of time decay recalls in both classes will also be provided in the detailed analysis when necessary, giving a more detailed understanding of the approaches' performance.

The options for the hyper-parameters of the approaches are presented in Table 5.4 while the chosen hyper-parameters for the experiments are presented in Tables 5.5 and 5.6. Values for memory related hyper-parameters were fixed to the values which were commonly used in the work of CDCMS, presented in Section 4.2.

Table 5.4: Options for Hyper-parameters

Hyper-parameter	GH-VFDT$_d$ HD-VFDT$_d$	Online Bagging$_d$	OOB$_d$ UOB$_d$	CDCMS.CIL	CDCMS
Base Learner	-	Hoeffding Tree with Naïve Bayes at leaves (HTNB)			
Ensemble Size	-	Sine/Agrawal/STAGGER/real-world: {15}, SEA: {10}			
Memory Size Multiplier	-			Sine/real-world: {6}, Agrawal/SEA: {4}, STAGGER: {10}	
Batch Size / Evaluation period	-			Sine/STAGGER/real-world: {600}, Agrawal: {400}, SEA: {800}	
Class Size Fading Factor (θ)	-		{0.9}*		-
Performance Fading Factor (α)	-			{0.9,0.99,0.999}	
Similarity Threshold (sim)	-			{0.4,0.6,0.8,0.9}	
Drift Detection Method	{No drift detection, DDM, EDDM}*				

- Grey rows denote pre-chosen hyper-parameter values.

* For CDCMS.CIL, this only applies to versions with resampling method.

** A few existing drift detector for class imbalanced data stream learning were initially considered in this study: PAUC-PH (Brzezinski and Stefanowski, 2014), DDM-OCI (S. Wang, Minku, Ghezzi, et al., 2013) and DDM-GMean^2. However, they exhibited a large amount of false-positive drift detections in the experiments with artificial data streams. They eventually took a very long period to complete a single experiment run. This is why they were not included in the experiments at last.

Table 5.5: Chosen Hyper-parameters of Existing Approaches

Stream Type	GH-VFDT$_d$	HD-VFDT$_d$	Online Bagging$_d$	OOB$_d$	UOB$_d$	CDCMS
Sine	DDM	DDM	DDM	DDM	DDM	DDM,0.6,0.999
Agrawal	DDM	DDM	DDM	EDDM	EDDM	No,0.6,0.999
SEA	No	No	DDM	DDM	No	DDM,0.8,0.999
STAGGER	EDDM	EDDM	EDDM	EDDM	EDDM	EDDM,0.6,0.99
Airlines	DDM	EDDM	No	EDDM	EDDM	EDDM,0.9,0.99
NOAA	No	No	No	No	EDDM	DDM,0.6,0.9
Covtype$_{(c_1=\{1\text{-}6\})}$	DDM	DDM	DDM	DDM	DDM	EDDM,0.4,0.9
Covtype$_{(c_1=1)}$	DDM	DDM	DDM	DDM	EDDM	DDM,0.6,0.99
Covtype$_{(c_1=2)}$	DDM	DDM	EDDM	No	No	EDDM,0.4,0.999
Covtype$_{(c_1=3)}$	DDM	DDM	DDM	No	No	EDDM,0.6,0.99
Covtype$_{(c_1=4)}$	EDDM	EDDM	EDDM	EDDM	DDM	EDDM,0.6,0.999
Covtype$_{(c_1=5)}$	EDDM	EDDM	EDDM	EDDM	EDDM	EDDM,0.8,0.9
Covtype$_{(c_1=6)}$	DDM	DDM	No	DDM	DDM	No,0.9,0.999
INSECTS$^{inc.}$	DDM	DDM	No	No	No	No,0.4,0.9
INSECTS$_{abr.}$	DDM	DDM	No	No	No	No,0.8,0.99
INSECTS$^{inc.}_{grad.}$	No	No	No	No	No	No,0.9,0.999
INSECTS$^{inc.}_{abr.\ re.}$	DDM	DDM	No	No	No	No,0.6,0.9
INSECTS$^{inc.}_{re.}$	DDM	DDM	No	No	No	No,0.4,0.999

- Covtype$_{(c_1=x)}$: "$c_1=x$" refers to the class 1 is the class x in the original data set while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream. "$c_1=\{x_0 - x_n\}$" refers to the class 1 is the class x_0-x_n in the original data set combined while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream.
- For all INSECTS data streams, "ae-albopictus" is the class 1. "inc." refers to incremental, "abr." refers to abrupt, "grad" refers to gradual, and "re." refers to recurring.
- For CDCMS, each row has the format of: {drift detection method},{similarity threshold},{performance fading factor (α)} while for other approaches, each row presents their chosen drift detection method only; "No" refers to use no drift detection method.

For each run, the performance in G-Mean and recalls of the approach were measured prequentially and sampled at every 500 time steps, except they were sampled at every 50 time steps for NOAA stream because the NOAA stream has a lot less examples than other real-world data streams (see Table 5.2). Reducing the sampling interval of predictive performance can increase the number of observations across the stream, allowing us to compare and to evaluate data stream learning approaches in details in short data streams. We do not reduce the sampling interval of predictive performance for long data streams because it will increase the total running time of each experiment run.

The prequential performance is calculated using Equation 5.3, which is the same

Table 5.6: Chosen Hyper-parameters of CDCMS.CIL

Stream Type	Preq. Acc.	Preq. Acc. OS	Preq. Acc. US	GMean	GMean OS	GMean US
Sine	DDM,0.4,0.99	DDM,0.8,0.99	DDM,0.6,0.999	DDM,0.6,0.999	DDM,0.9,0.99	DDM,0.9,0.99
Agrawal	No,0.6,0.999	No,0.8,0.99	EDDM,0.6,0.999	No,0.8,0.99	EDDM,0.4,0.99	No,0.4,0.99
SEA	DDM,0.6,0.99	DDM,0.9,0.999	DDM,0.9,0.99	DDM,0.4,0.99	DDM,0.9,0.99	DDM,0.6,0.99
STAGGER	EDDM,0.6,0.999	DDM,0.6,0.9	EDDM,0.6,0.99	EDDM,0.4,0.99	DDM,0.8,0.99	EDDM,0.8,0.99
Airlines	DDM,0.9,0.99	DDM,0.9,0.99,0.9	No,0.9,0.9	EDDM,0.9,0.99	EDDM,0.4,0.99	No,0.8,0.9
NOAA	DDM,0.4,0.9	DDM,0.6,0.9,0.9	EDDM,0.4,0.9	EDDM,0.6,0.9	DDM,0.6,0.99	DDM,0.9,0.9
$\text{Covtype}_{(c_1=\{1\text{-}6\})}$	DDM,0.6,0.9	No,0.4,0.9	DDM,0.9,0.9	EDDM,0.9,0.9	EDDM,0.9,0.99	No,0.8,0.9
$\text{Covtype}_{(c_1=1)}$	DDM,0.9,0.9	No,0.8,0.9	EDDM,0.6,0.9	EDDM,0.9,0.999	EDDM,0.4,0.9	No,0.4,0.9
$\text{Covtype}_{(c_1=2)}$	EDDM,0.4,0.999	No,0.4,0.9	No,0.6,0.999	DDM,0.9,0.999	No,0.9,0.999	No,0.8,0.999
$\text{Covtype}_{(c_1=3)}$	No,0.6,0.999	No,0.8,0.9	No,0.9,0.9	DDM,0.6,0.9	No,0.6,0.9	No,0.8,0.99
$\text{Covtype}_{(c_1=4)}$	EDDM,0.4,0.99	No,0.9,0.999	No,0.8,0.9	EDDM,0.9,0.9	No,0.4,0.9	No,0.4,0.9
$\text{Covtype}_{(c_1=5)}$	EDDM,0.4,0.9	EDDM,0.8,0.999	No,0.9,0.99	EDDM,0.4,0.999	EDDM,0.4,0.999	No,0.4,0.99
$\text{Covtype}_{(c_1=6)}$	No,0.6,0.999	DDM,0.9,0.999	No,0.6,0.9	No,0.9,0.9	DDM,0.6,0.9	EDDM,0.8,0.9
$\text{INSECTS}_{\text{inc.}}$	No,0.6,0.9	No,0.9,0.9	No,0.9,0.99	No,0.6,0.99	No,0.8,0.9	No,0.8,0.999
$\text{INSECTS}_{\text{abr.}}$	No,0.6,0.99	No,0.4,0.99	No,0.4,0.99	No,0.4,0.999	No,0.4,0.999	No,0.9,0.999
$\text{INSECTS}^{\text{inc.}}_{\text{grad.}}$	No,0.8,0.999	No,0.6,0.9	No,0.6,0.9	No,0.4,0.9	No,0.6,0.999	No,0.8,0.999
$\text{INSECTS}^{\text{inc.}}_{\text{abr. re.}}$	No,0.8,0.9	No,0.6,0.999	No,0.6,0.999	No,0.6,0.999	No,0.4,0.999	No,0.9,0.999
$\text{INSECTS}^{\text{inc.}}_{\text{re.}}$	No,0.8,0.999	No,0.4,0.999	No,0.9,0.999	No,0.4,0.999	No,0.4,0.999	No,0.6,0.999

- $\text{Covtype}_{(c_1=x)}$: "c_1=x" refers to the class 1 is the class x in the original data set while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream. "c_1=$\{x_0 - x_n\}$" refers to the class 1 is the class x_0-x_n in the original data set combined while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream.
- For all INSECTS data streams, "ae-albopictus" is the class 1. "inc." refers to incremental, "abr." refers to abrupt, "grad" refers to gradual, and "re." refers to recurring.
- Each row has the format of: {drift detection method},{similarity threshold},{performance fading factor (α)}; "No" refers to use no drift detection method.

equation used in Sections 4.1.2.2 and 4.2.2.2.

$$preq_perform_t(D) = \begin{cases} preq_perform^{ex}_t(D), & \text{if } t = f \\ \frac{preq_perform^{ex}_t(D) + \alpha \times preq_perform_{t-1}(D) \times (t-f)}{t-f+1}, & \text{otherwise} \end{cases} \tag{5.3}$$

where t is the current time step, $preq_perform^{ex}_t$ is 0 if the prediction to the current training

example ex before learning is wrong and 1 if it is correct, f is the first time step used in the calculation, and α is the time decaying factor. In this work, we adopted $\alpha = 0.999$

After running the experiments, Friedman test with a level of significance of 0.05 was applied to each data stream to confirm if there is any statistical significance between the predictive performance of the data stream learning approaches. If there is, Nememyi post-hoc test was used to determine which approach is significantly different from the top-ranked approach. In the statistical tests, each group corresponds to a data stream learning approach while each observation within a group corresponds the average prequential predictive performance across the a given data stream in a single run. Note that, in the comparison between CDCMS.CIL and CDCMS (the first study which answers RQ2.1), we applied Wilcoxon rank-sum test instead of Friedman test with Nememyi post-hoc test because it is a pairwise comparison.

5.3 Analysis

This section presents the analysis done to evaluate the proposed diversity-based framework for drifting class imbalanced data stream learning – CDCMS.CIL. This analysis is divided into three parts to answer RQ2.1, RQ2.2 and RQ2.3 respectively. Lastly, it concludes the findings and results to answer the research question second research question presented in Section 1.2.2.

5.3.1 Comparison between CDCMS.CIL and CDCMS

In this section, CDCMS.CIL was compared against its predecessor, CDCMS. This comparison aims to find out when and why the proposed memory strategies for class imbalanced

learning are helpful and detrimental without other strategies for addressing class imbalance. Therefore, CDCMS.CIL used in this comparison which uses prequential accuracy as the weighting metric and no resampling method.

Concretely, we calculated the thirty runs average differences in time decay G-Mean, class 0 and class 1 recalls between CDCMS.CIL$^{\text{Preq. Acc.}}$ and CDCMS on artificial streams. These results are presented in Tables 5.7. Wilcoxon signed rank test is also performed to confirm if there is any statistical significant difference between the performance of the two approaches. The null hypothesis is rejected when the p-value returned by the Wilcoxon signed rank test is less than or equal to 0.05. In Table 5.7, cells with positive performance difference mean that CDCMS.CIL$^{\text{Preq. Acc.}}$ performed better than CDCMS while cells with negative performance difference mean that CDCMS.CIL$^{\text{Preq. Acc.}}$ performed worse than CDCMS. The performance difference is said to be large when the absolute value of the difference is larger than or equal to 1%, and thus are highlighted in lime (positive difference) or orange (negative difference).

Note that, we only compare CDCMS.CIL$^{\text{Preq. Acc.}}$ and CDCMS on Sine and STAGGER streams. This is because they did not employ any drift detector for Agrawal streams (See Tables 5.5 and 5.6) and drift detections were rarely triggered in SEA streams. Therefore, their performance on Agrawal and SEA streams were governed by the strategy of periodically introducing new base learners to the main ensemble. So, we cannot evaluate which version of the memory strategies is better based on the results of CDCMS.CIL$^{\text{Preq. Acc.}}$ and CDCMS on Agrawal and SEA streams.

Focusing on the performance differences of time decay G-Mean in the artificial data streams, Table 5.7 shows that CDCMS.CIL$^{\text{Preq. Acc.}}$ usually performed similar or worse than CDCMS, but the margins were mostly small. In contrast, when CDCMS.CIL$^{\text{Preq. Acc.}}$ performed similar or better than CDCMS in severe class imbalanced cases (class imbalance

Table 5.7: Difference in Time Decay Performance in Artificial Data Streams (CDCMS.CIL$^{\text{Preq. Acc.}}$ - CDCMS)

Stream	Abrupt (0.5:0.5)	Abrupt (0.7:0.3)	Abrupt (0.8:0.2)	Abrupt (0.9:0.1)	Gradual (0.5:0.5)	Gradual (0.7:0.3)	Gradual (0.8:0.2)	Gradual (0.9:0.1)
Time Decay G-Mean								
Sine1	-	-0.2658%	-0.2095%	-0.3721%	-	-0.3247%	-0.3945%	-0.3068%
Sine2	-	-	-	-0.2896%	-	0.3616%	-	-0.7599%
STAGGER1	-	-	0.0211%	0.4003%	-	-	0.1320%	0.9297%
STAGGER2	-	-	-0.0183%	0.2876%	-	-	-	1.0446%
Time Decay Recall (Class 0)								
Sine1	-	0.0449%	-	0.0253%	-	-	0.1496%	0.0348%
Sine2	-	-	0.0501%	-	-	-	0.0626%	0.0439%
STAGGER1	-	-	-	-	-	-	-	-
STAGGER2	-	-	-0.0165%	-	-	-	-	-
Time Decay Recall (Class 1)								
Sine1	0.0937%	-0.5100%	-0.3920%	-0.5741%	-0.0807%	-0.5683%	-0.6676%	-0.4460%
Sine2	-	-	-	-0.4743%	0.1298%	0.5696%	-0.4770%	-1.1806%
STAGGER1	-	0.0256%	0.0411%	0.7809%	-	-	0.2663%	1.7075%
STAGGER2	-0.0128%	-	-0.0201%	0.5320%	-	-	-	1.8736%

- Hyphen (-) refers to no statistical significance based on Wilcoxon signed rank test.
- Based on Wilcoxon signed rank test, positive values denote CDCMS.CIL$^{\text{Preq. Acc.}}$ has better performance, while negative values denote CDCMS has better performance.
- Cells with absolute difference $\geq 1.0\%$ are highlighted in lime / light grey when CDCMS.CIL$^{\text{Preq. Acc.}}$ was significantly better and in orange / dark grey cells when CDCMS was significantly better.

ratio: 0.9:0.1), it was by a large margin, especially in gradual drift streams.

These results preliminarily show that the novel diversity-based memory strategies for class imbalanced learning can help the framework to better manage the memory and recover relevant knowledge under severely class imbalanced condition than their predecessors in CDCMS. However, given most cases show that the performance difference between them was usually small. This indicates that an appropriate weighting metric and strategy to reduce learning bias towards the majority class are needed to enable the memory strategies fully support class imbalanced learning. One intuitive reason is that, despite the memory strategies recovered correct models to deal with drift, these recovered models are still not quite helpful because they learnt with bias towards the majority class of the corresponding

concept.

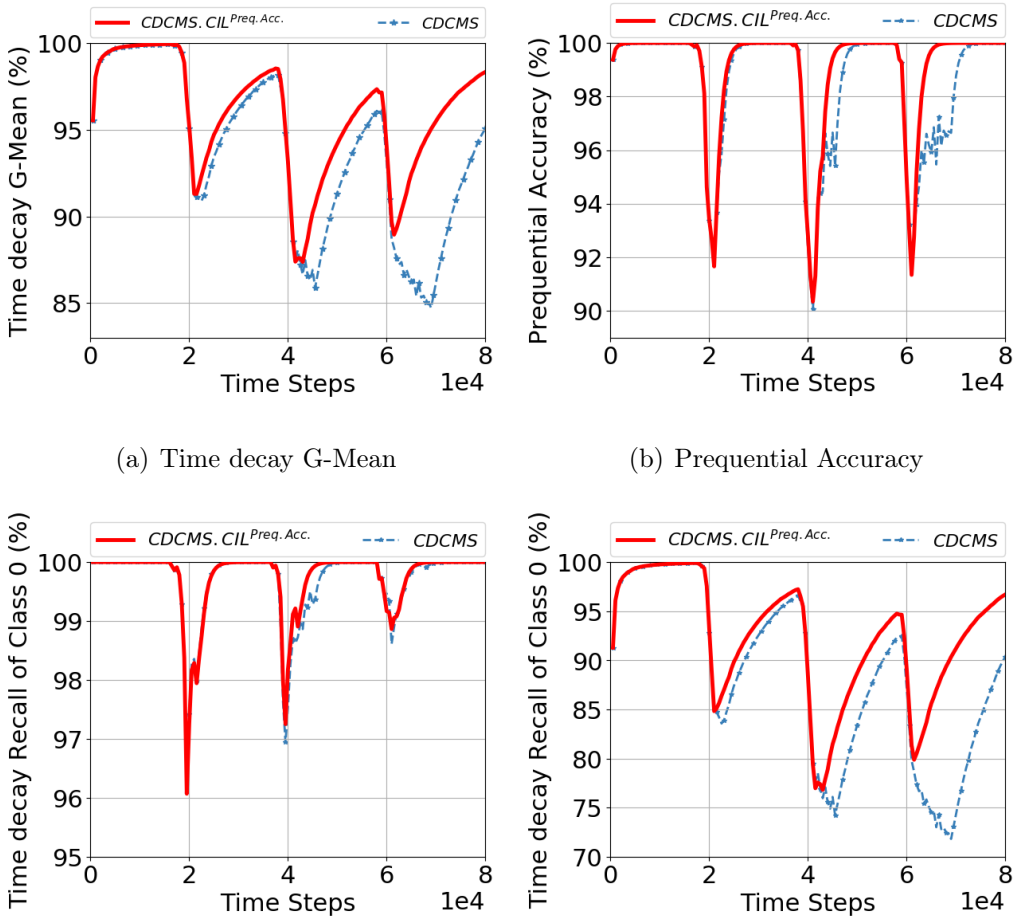
Over the next sections, representative cases were chosen to discuss the performance differences between CDCMS.CIL$^{\text{Preq. Acc.}}$ and CDCMS on artificial and real-world data streams in details. Section 5.3.1.1 discusses the result in STAGGER2 gradual (0.9:0.1) stream as the representative cases where CDCMS.CIL$^{\text{Preq. Acc.}}$ performed better than CDCMS. Section 5.3.1.2 discusses the results in Sine2-gradual (0.9:0.1) stream as the representative case where CDCMS.CIL$^{\text{Preq. Acc.}}$ performed worse than CDCMS. Lastly, Section 5.3.1.3 gives a short summary to the findings.

5.3.1.1 Case where CDCMS.CIL performed better

This section discusses the result on STAGGER2-gradual (0.9:0.1) as the representative case where CDCMS.CIL$^{\text{Preq. Acc.}}$ performed better than CDCMS.

Table 5.7 shows that CDCMS.CIL$^{\text{Preq. Acc.}}$ performed better than CDCMS in STAGGER2-gradual (0.9:0.1) by 1.0446% in time decay G-Mean. At the bottom part of the same table, it also shows that the performance of CDCMS.CIL$^{\text{Preq. Acc.}}$ in time decay class 1 recall is better than that of CDCMS in both STAGGER1-gradual (0.9:0.1) and STAGGER2-gradual (0.9:0.1). Thus, this analysis focuses on discussing when and why CDCMS.CIL$^{\text{Preq. Acc.}}$ performed better than CDCMS in STAGGER streams, using STAGGER2-gradual (0.9:0.1) as the representative case. Figure 5.7 compares their time decay G-Mean, class 0 and class 1 recall, and prequential accuracy over the STAGGER2-gradual (0.9:0.1) stream in the first run. Other STAGGER streams can be explained similarly. The concept sequence of STAGGER2 is: STAGGER-f2 $\rightarrow$ STAGGER-f3 $\rightarrow$ STAGGER-f1 $\rightarrow$ STAGGER-f2.

CDCMS.CIL$^{\text{Preq. Acc.}}$ had five drift detections during the first drift of the STAGGER2-gradual (0.9:0.1) stream (f2 $\rightarrow$ f3), where the second to the fifth detection were given before



(a) Time decay G-Mean

(b) Prequential Accuracy

(c) Time decay Recall in Class 0

(d) Time decay Recall in Class 1

Figure 5.7: CDCMS.CIL vs CDCMS: STAGGER2 (Gradual, 9-1)

the clustering in the model space strategy for class imbalanced learning being triggered. At last, it correctly recognised that this drift leads to a new concept. On the other hand, CDCMS had seven drift detections and it recovered a set of models that were created and learnt from the transitional period of this drift. Thus, CDCMS.CIL$^{\text{Preq. Acc.}}$ recovered slightly better than CDCMS from this drift in Figures 5.7(a) and 5.7(d).

In the second drift (f3 $\rightarrow$ f1), CDCMS.CIL$^{\text{Preq. Acc.}}$ had nine drift detections, which were all given during the transitional period of this drift. For the first eight drift detections, it kept recovering a small number of models that were created and learnt from the transitional

period of this drift. For the last drift detection, it just recovered two models that were created and learnt near the end of this transitional period, enabling CDCMS.CIL$^{\text{Preq. Acc.}}$ to adapt better than CDCMS to the concept after this drift. In contrast, CDCMS had twelve drift detections. These drift detections were given since the start of this drift until 4k time steps after the drift was completed, meaning that the last few drift detections were false-positives. Figures 5.7(a) - 5.7(d) also show that CDCMS struggled for a longer period than CDCMS.CIL$^{\text{Preq. Acc.}}$ in all metrics and recovered its predictive performance worse than that of CDCMS.CIL$^{\text{Preq. Acc.}}$. The reason behind the false-positive drift detections is the fact that CDCMS kept recovering models that learnt from the transitional period of the first drift, meaning that they held the intermediate concept between STAGGER-f2 and STAGGER-f3 (the first drift is: STAGGER-f2 $\rightarrow$ STAGGER-f3). According to the Table A.1, the percentage difference between STAGGER-f2 and STAGGER-f1, STAGGER-f3 and STAGGER-f1 are 59.3% and 77.8% respectively, which are quite large. Therefore, the intermediate concept hindered CDCMS from adapting to this drift by leading frequent false-positive drift detections.

The last drift of STAGGER2-gradual (0.9:0.1) (f3 $\rightarrow$ f2) is a recurrent drift leading back to the concept STAGGER-f2. CDCMS.CIL$^{\text{Preq. Acc.}}$ had four drift detection which were all given within the transitional period of this drift. At the third drift detection, it recovered a mixed set of models that learnt from the transitional period of the first drift and the second drift. At the last drift detection, it correctly and timely recovered a set of models that learnt from the concept STAGGER-f2 to handle this recurrent drift. In contrast, CDCMS had a lot of drift detections since the start of this drift until around 7k time steps after the drift was completed, meaning that the last few drift detections were false-positives. Before the last drift detection, it kept recovering models that learnt from the transitional period of previous two drifts, leading to more false-positive drift detections. For the final drift detection, although it correctly recovered a set of models that learnt from the concept

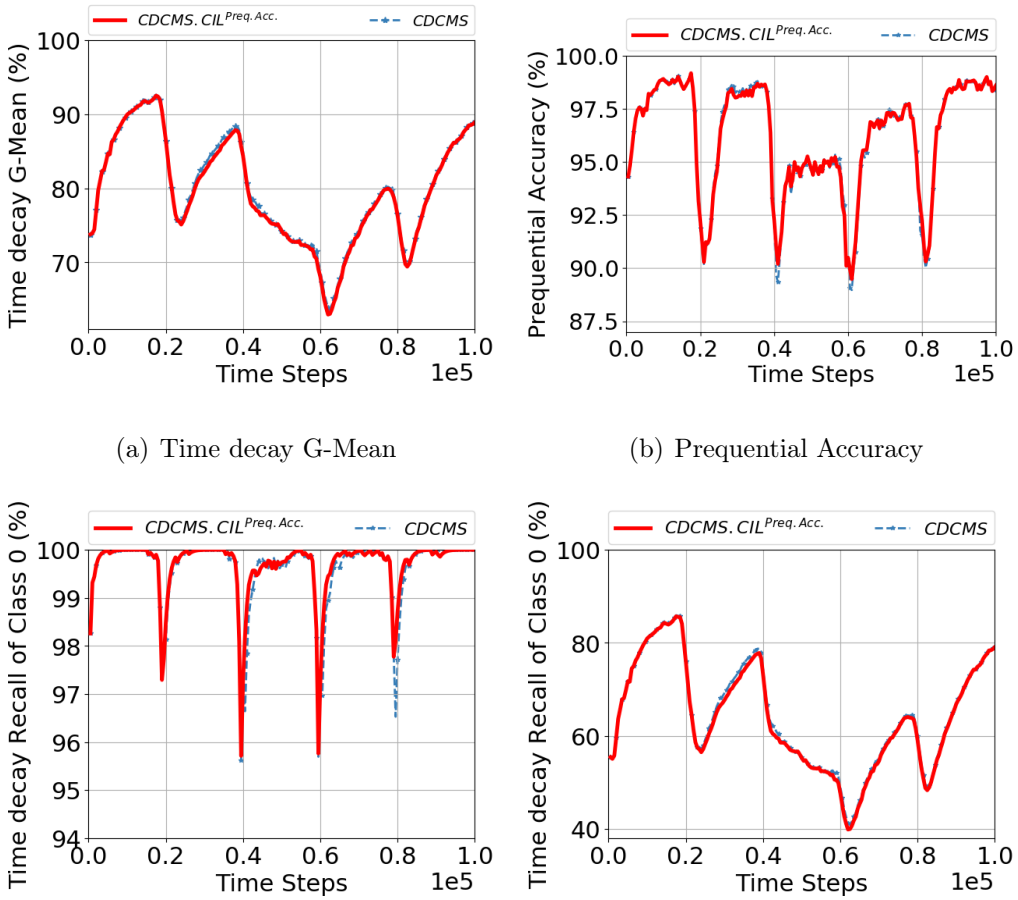
STAGGER-f2 to handle this recurrent drift, it had already struggled for an extended long period of at least 7k time steps, since the actual drift time step. Thus, as shown on Figures 5.7(a) and 5.7(d), CDCMS.CIL$^{\text{Preq. Acc.}}$ reacted and recovered its performance a lot better than that of CDCMS.

To summarise this analysis on the case of STAGGER2-gradual (0.9:0.1) stream, the advantages of the novel clustering in the model space strategy for class imbalanced learning over its predecessor have been manifested. The behaviour of CDCMS.CIL$^{\text{Preq. Acc.}}$ in STAGGER2-gradual (0.9:0.1) has shown that this novel strategy is more effective than its predecessor in recovering relevant knowledge and identifying new concepts despite the data stream being severely class imbalanced.

5.3.1.2 Case where CDCMS.CIL performed similar (marginally worse)

The lower part of Table 5.7 shows that CDCMS.CIL$^{\text{Preq. Acc.}}$ performed worse than CDCMS in the time decay class 1 recall by 1.1806% in Sine2-gradual (0.9:0.1) stream, which is the only orange cell in the table. However, Figure 5.8(d) shows that time decay class 1 recall of CDCMS.CIL$^{\text{Preq. Acc.}}$ was similar to CDCMS throughout this stream. So, in this section, we are going to further investigate why they performed similarly rather than arguing CDCMS.CIL$^{\text{Preq. Acc.}}$ is underperformed solely based on Table 5.7. According to Table 5.1, the concept sequence of this stream is: Sine-f1 $\rightarrow$ Sine-f2 $\rightarrow$ Sine-f3 $\rightarrow$ Sine-f4 $\rightarrow$ Sine-f1, where all drifts are severe.

During the first drift (f1 $\rightarrow$ f2), CDCMS.CIL$^{\text{Preq. Acc.}}$ had three drift detection where the last two were given before the clustering in the model space for class imbalanced learning being triggered. At last, it only recovered one model that was created and learnt around the start of this drift. CDCMS had the same number of drift detections as CDCMS.CIL$^{\text{Preq. Acc.}}$. It recovered three models where the first one was created and learnt at the start of this drift



(a) Time decay G-Mean

(b) Prequential Accuracy

(c) Time decay Recall in Class 0

(d) Time decay Recall in Class 1

Figure 5.8: CDCMS.CIL vs CDCMS: Sine2 (Gradual, 9-1)

while the other two models learnt from Sine-f1. The reason for recovering these models was because their drift detection were both given near the start of the drift (i.e. around 1000 time steps before the central drift point). This means that there were not enough examples from the post-drift concept for their clustering in the model space strategy to identify whether it is a new or a recurring concept. Therefore, they performed similarly. During the second concept, Figure 5.8 shows that CDCMS performed marginally better than CDCMS.CIL$^{\text{Preq. Acc.}}$ at around 30-40k time steps. The behaviour log reveals that it was because CDCMS maintains base learners in the main ensemble longer than CDCMS.CIL$^{\text{Preq. Acc.}}$. So, its base learners had longer time to learn from the current concept. This could be the fact that CD-

CMS used a larger performance time decaying factor than CDCMS.CIL$^{\text{Preq. Acc.}}$ (CDCMS: 0.999, CDCMS.CIL$^{\text{Preq. Acc.}}$: 0.99, see Tables 5.5 and 5.6). This means that it put slightly more emphasis to the past performance than CDCMS.CIL$^{\text{Preq. Acc.}}$, being less sensitive to predictive performance changes of base learners. Thus, mature models were likely to stay longer in the main ensemble.

CDCMS had two drift detections during the second drift (f2 $\rightarrow$ f3). For the first drift detection, it recovered a set of models that learnt from Sine-f2. Thus, the second drift detection was then triggered soon at the end of this drift. It then recovered one model that learnt briefly from Sine-f1, and five models that had been recovered at the first drift detection where four of them also learnt from the concept of Sine-f2 briefly. Given the information from Table A.1, Sine-f2 and Sine-f3 have 73.2% differences. That was why it did not manage to recover its predictive performance from this drift. However, Figure 5.8 shows that CDCMS.CIL$^{\text{Preq. Acc.}}$ also performed similarly to CDCMS, despite it had only one drift detection and recovered three models that have learnt briefly from the concept Sine-f1 (Sine-f1 is more similar to Sine-f3 with difference in 26.8%, while Sine-f2 is quite different from Sine-f3 with difference in 73.2%, see Table A.1). This indicates that the recovered models were not so helpful in adapting to Sine-f3, potentially because there was no strategy to address class imbalance learning. In other words, strategy to deal with class imbalance is necessary to enable the new memory strategies fully support class imbalance learning.

One may wonder why CDCMS.CIL$^{\text{Preq. Acc.}}$ can exploit relevant past knowledge to deal with the recurrent drift in STAGGER2-gradual (0.9:0.1), without any strategy to address class imbalance learning, but it did not work in this case where relevant past models were also recovered correctly. The potential reason here could be STAGGER concepts are easier to learn than Sine concepts. This is deduced by the fact that both approaches' time decay G-Mean were in the range of 85%-100% in Figure 5.7 but their time decay G-Mean in Figure 5.8 were only in the range 60%-95%. Therefore, strategies might not needed to deal with

class imbalance when learning class imbalanced STAGGER concepts. Nevertheless, class imbalance still affected the effectiveness of the old memory strategies in CDCMS to recover relevant past knowledge through the most recent batch of examples. Therefore, we can see that CDCMS usually struggled longer period than CDCMS.CIL$^{\text{Preq. Acc.}}$ in dealing with drifts.

In the third drift (f3 $\rightarrow$ f4), CDCMS.CIL$^{\text{Preq. Acc.}}$ had three drift detection where the last two were given before the clustering in the model space for class imbalanced learning being triggered. At last, it recovered a set of models that purely learnt from Sine-f2. This behaviour is also reasonable because Sine-f2 is similar to the concept after this drift, which is Sine-f4 (see Table A.1). CDCMS had two drift detections where the last one was given before the clustering in the model space being triggered. It then recovered a set of models that had briefly learnt from the concepts Sine-f2 and Sine-f3. Again, the behaviour of CDCMS.CIL$^{\text{Preq. Acc.}}$ was more reasonable than CDCMS, but they still performed similarly, indicating that resampling strategy is necessary to enable the new memory strategies fully support class imbalance learning.

The last drift (f4 $\rightarrow$ f1) is a recurrent drift leading back to the concept Sine-f1. CDCMS.CIL$^{\text{Preq. Acc.}}$ and CDCMS both had several drift detections. At their last drift detection, CDCMS.CIL$^{\text{Preq. Acc.}}$ recovered a set of models that learnt a mixed concept of Sine-f2 and Sine-f4, whereas CDCMS recovered a set of models that learnt a mixed concept of Sine-f2, Sine-f3 and Sine-f4. As these drift detection were all given early in the transitional period of this drift, it is reasonable that both approaches tended to recover knowledge of concepts Sine-f2 and Sine-f4, as most examples received upon this time step belong to the pre-drift concept (Sine-f4). Yet, the concept after this drift is Sine-f1 which is quite different from the knowledge they recovered. As the result, they took a while to recover their predictive performance from this drift. From here, we can see that the new clustering in the model space strategy for class imbalanced learning also inherited the disadvantage of its predecessor,

which is prone to multiple extremely severe gradual drifts, as explained in Section 4.2.3.1.2.

5.3.1.3 Short Summary

In this analysis, the performance of CDCMS.CIL$^{\text{Preq. Acc.}}$ and CDCMS have been compared on Sine and STAGGER streams with different severity of class imbalance and concept drifts. It has been shown that the novel memory strategies for class imbalanced learning were more advantageous than their predecessors in severely class imbalanced data streams. The main reason is that they managed to recover relevant knowledge to handle concept drift without being affected by class imbalance (RQ2.1). Nevertheless, this analysis also shows that the effectiveness of the recovered models in dealing with drifts depends on how well did they learn from their corresponding concept. Specifically, if class imbalance is not addressed by any strategies, past models could not be helpful in dealing with future drifts despite the concepts they learnt.

5.3.2 Comparing Predictive Performance of Different Versions of CDCMS.CIL

This section compares the predictive performance of CDCMS.CIL when using different resampling methods and weighting metrics. To analyse quantitatively, their thirty runs predictive performances were grouped and compared by different factors: class imbalance ratio, drift speed and stream type. Friedman test was performed on each group to confirm if there is any statistical significance between their performance. After that, Nemenyi post-hoc test was performed to confirm if there is any statistical significance between the pairs of approaches' performances. Again, the null hypothesis is rejected when the p-value returned by the statistical tests is less than or equal to 0.05. Tables 5.8, 5.9 and 5.10 present

the performance rankings of different versions of CDCMS.CIL by factors in time decay G-Mean, majority class and minority class recall respectively. The smaller ranking in number, the better performance. We denote different versions of CDCMS.CIL in the format of "CDCMS.CIL$^{\{\text{weighting metric}\}}_{\{\text{resampling method}\}}$", where "Preq. Acc.", "GMean", "OS" and "US" refer to prequential accuracy, time decay G-Mean, oversampling and undersampling respectively. If neither "OS" nor "US" presents in the name, it means that this version of the approach does not employ any resampling method.

Over the next paragraphs, a general discussion on their predictive performance is first given, followed by a detailed qualitative analysis to further confirm their behaviours in Section 5.3.2.1. Lastly this part of study is shortly summarised in Section 5.3.2.2

Table 5.8: Statistical (Friedman) ranking of time decay G-Mean on streams grouped by factors (Comparison of weighting metric and resampling method choices)

Groups	Preq. Acc.	Preq. Acc. OS	Preq. Acc. US	GMean	GMean OS	GMean US
Grouped by imbalance ratio (Artificial data streams)						
0.5:0.5	3.98	2.6183	5.015	2.6833	2.7167	3.9867
0.7:0.3	5.445	2.2733	3.4267	5.0917	2.4267	2.3367
0.8:0.2	5.3417	2.3683	3.575	5.19	2.625	1.9
0.9:0.1	5.0817	2.445	3.5683	5.0033	3.1267	1.775
Grouped by concept drift speed						
Abrupt	4.9725	2.0975	4.1342	4.3775	2.6217	2.7967
Gradual	4.9517	2.755	3.6583	4.6067	2.8258	2.2025
Real-World	4.0	2.6429	4.0	3.5714	2.7857	4.0
Grouped by streams						
Sine	5.0458	2.0375	4.175	4.825	1.5125	3.4042
Agrawal	5.1625	1.9333	4.2458	4.2875	3.5125	1.8583
SEA	5.2625	3.6958	3.9292	4.2083	1.9542	1.95
STAGGER	4.55	2.5333	2.9667	4.95	2.6208	3.3792
Airlines	6.0	5.0	3.0	4.0	2.0	1.0
NOAA	4.0	3.0	5.0	6.0	1.0	2.0
Covtype	4.1429	2.7143	3.0	4.2857	3.2857	3.5714
INSECTS	3.4	2.0	5.4	2.0	2.6	5.6
All streams	4.7337	2.5649	3.8904	4.3018	2.7862	2.723

- The p-values of Friedman tests are all $\leq$2.2E-16.
- Highlighted ranks denote significant superior performance.

First and foremost, Table 5.8 shows that CDCMS.CIL$^{\text{Preq. Acc.}}_{\text{OS}}$ performed the best in

Table 5.9: Statistical (Friedman) ranking of time decay class 0 (majority class) recall on streams grouped by factors (Comparison of weighting metric and resampling method choices)

Groups	Preq. Acc.	Preq. Acc. OS	Preq. Acc. US	GMean	GMean OS	GMean US
Grouped by imbalance ratio (Artificial data streams)						
0.5:0.5	4.4683	3.105	4.2433	3.2083	3.0433	2.9317
0.7:0.3	1.67	3.1375	4.9467	1.7917	3.8508	5.6033
0.8:0.2	1.3667	3.0883	4.7667	2.1033	3.9717	5.7033
0.9:0.1	1.2617	2.9275	4.7783	2.22	3.9208	5.8917
Grouped by concept drift speed						
Abrupt	2.45	2.8983	4.695	2.4258	3.5033	5.0275
Gradual	1.9333	3.2308	4.6725	2.2358	3.89	5.0375
Real-World	2.0938	2.1562	5.0312	2.7188	3.5938	5.4062
Grouped by streams						
Sine	1.6625	3.0167	5.0	2.3958	3.2958	5.6292
Agrawal	2.2708	2.8458	5.0625	2.0083	3.6667	5.1458
SEA	1.8833	4.15	3.1083	2.4125	4.6542	4.7917
STAGGER	2.8708	2.5333	5.2042	3.0958	2.3708	4.925
Airlines	1.0	3.0	5.0	2.0	4.0	6.0
NOAA	3.0	4.0	1.0	2.0	6.0	5.0
Covtype	2.1429	2.1429	5.0	3.0	2.7143	6.0
INSECTS	1.6	1.6	6.0	3.0	3.8	5.0
All streams	2.1753	2.9132	4.7417	2.3955	3.6795	5.0948

- The p-values of Friedman tests are all $\leq$2.2E-16.
- Highlighted ranks denote significant superior performance.
- For class balanced cases, this table shows the rankings in the time decay class 0 recall.

terms of time decay G-Mean in general. Tables 5.9 and 5.10 show that, although it usually achieved higher rankings in the majority class than the minority class by factor, its overall rankings in time decay recalls of both classes are similar (both are around 2.9). This result is quite surprising, as it was expected that CDCMS.CIL$_{\text{OS}}^{\text{GMean}}$ or CDCMS.CIL$_{\text{US}}^{\text{GMean}}$, which use a weight metric that is more appropriate for class imbalance, should perform the best in general.

In contrast, Table 5.8 shows that CDCMS.CIL$^{\text{Preq. Acc.}}$ performed the worst in terms of time decay G-Mean in general. This is expected as it did not employ any resampling method to deal with class imbalance and its weighting metric was class imbalance sensitive. When looking into its rankings of time decay recalls in both classes, Tables 5.9 and 5.10

Table 5.10: Statistical (Friedman) ranking of time decay class 1 (minority class) recall on streams grouped by factors (Comparison of weighting metric and resampling method choices)

Groups	Preq. Acc.	Preq. Acc. OS	Preq. Acc. US	GMean	GMean OS	GMean US
Grouped by imbalance ratio (Artificial data streams)						
0.5:0.5	3.41	2.105	4.9083	2.885	3.0233	4.6683
0.7:0.3	5.6033	3.1217	2.6033	5.26	2.9283	1.4833
0.8:0.2	5.5433	2.8083	3.11	5.2717	2.8483	1.4183
0.9:0.1	5.21	2.7967	3.2467	5.0667	3.2933	1.3867
Grouped by concept drift speed						
Abrupt	4.9192	2.4133	3.6875	4.56	2.9242	2.4958
Gradual	4.9642	3.0025	3.2467	4.6817	3.1225	1.9825
Real-World	4.4375	3.875	2.5	4.25	3.0625	2.875
Grouped by streams						
Sine	5.1417	2.9708	3.7083	4.7458	2.1625	2.2708
Agrawal	5.0833	2.3708	3.6958	4.4333	3.6208	1.7958
SEA	5.0917	2.85	4.0708	4.5083	2.2875	2.1917
STAGGER	4.7167	2.5792	2.3	5.1708	3.0917	3.1417
Airlines	6.0	4.0	2.0	5.0	3.0	1.0
NOAA	4.0	3.0	5.0	6.0	1.0	2.0
Covtype	4.1429	3.0	2.8571	4.2857	3.4286	3.2857
INSECTS	5.8	5.0	1.0	4.0	3.2	2.0
All streams	4.8576	2.9024	3.3059	4.559	3.0299	2.3451

- The p-values of Friedman tests are all $\leq$2.2E-16.
- Highlighted ranks denote significant superior performance.
- For class balanced cases, this table shows the rankings in the time decay class 1 recall.

show that CDCMS.CIL$^{\text{Preq. Acc.}}$ usually achieved the highest ranking in time decay recall of the majority class, while achieved the lowest ranking in time decay recall of the minority class, confirming that it had stronger bias towards the majority class than other versions of CDCMS.CIL.

If we solely concern the time decay minority class recall, Table 5.10 shows that CDCMS.CIL$^{\text{GMean}}_{\text{US}}$ performed the best in general. Table 5.8 also shows that, in terms of time decay G-Mean, it performed better than other versions of CDCMS.CIL in highly class imbalanced cases and gradual drift cases. However, Table 5.10 shows that it performed poorly in the majority class, which means it sacrificed its performance in the majority class to achieve the good performance in the minority class.

The analysis in Section 5.3.1 suggests that a resampling method is needed to enable the proposed memory strategies to work well. However, different types of resampling method may impact the proposed memory strategies differently. When comparing rankings of time decay G-Mean by different resampling methods, Table 5.8 shows that CDCMS.CIL$^{\text{Preq. Acc.}}$ and CDCMS.CIL$^{\text{GMean}}$, which did employ any resampling method, were the least performing versions, which coheres with the findings in Section 5.3.1. Between the choice of oversampling or undersampling, Table 5.8 shows that it depends on the choice of weighting metric in the main ensemble. When the weighting metric is prequential accuracy, oversampling is more advantageous than undersampling. When the weighting metric is time decay G-Mean, the overall rankings of CDCMS.CIL$_{\text{OS}}^{\text{GMean}}$ and CDCMS.CIL$_{\text{US}}^{\text{GMean}}$ are similar to each other. Tables 5.9 and 5.10 reveal more information about the performance between CDCMS.CIL$_{\text{OS}}^{\text{GMean}}$ and CDCMS.CIL$_{\text{US}}^{\text{GMean}}$. They show that oversampling achieved a balanced rankings in time decay recalls of both classes whereas undersampling achieved better rankings in time decay minority class recall than the time decay majority class recall. Nonetheless, it is difficult to know which resampling method works better with the proposed memory strategies only based on the rankings. Therefore, further investigation has been done and is presented in Section 5.3.2.1.

When comparing the rankings of time decay G-Mean by different weighting metrics, Table 5.8 shows that, using time decay G-Mean usually achieved better rankings than using prequential accuracy, except CDCMS.CIL$_{\text{OS}}^{\text{GMean}}$ which usually achieved lower rankings than CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$. When comparing the rankings of the time decay recalls of both classes between these two versions, Tables 5.9 and 5.10 show that CDCMS.CIL$_{\text{OS}}^{\text{GMean}}$ usually achieved lower rankings than CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ in the majority class while their rankings in the minority class were close. This indicates that the use of time decay G-Mean as weighting metric with oversampling may overly focus on the minority class performance.

As the above quantitative analysis shows that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ usually performs

the best, we further analyse its behaviour in Section 5.3.2.1 using representative cases. In particular, we compare it against CDCMS.CIL$^{\text{Preq. Acc.}}$ and CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ to see how the resampling method impacts the effectiveness of the proposed memory strategies in terms of concept drift adaptation in class imbalanced data stream learning.

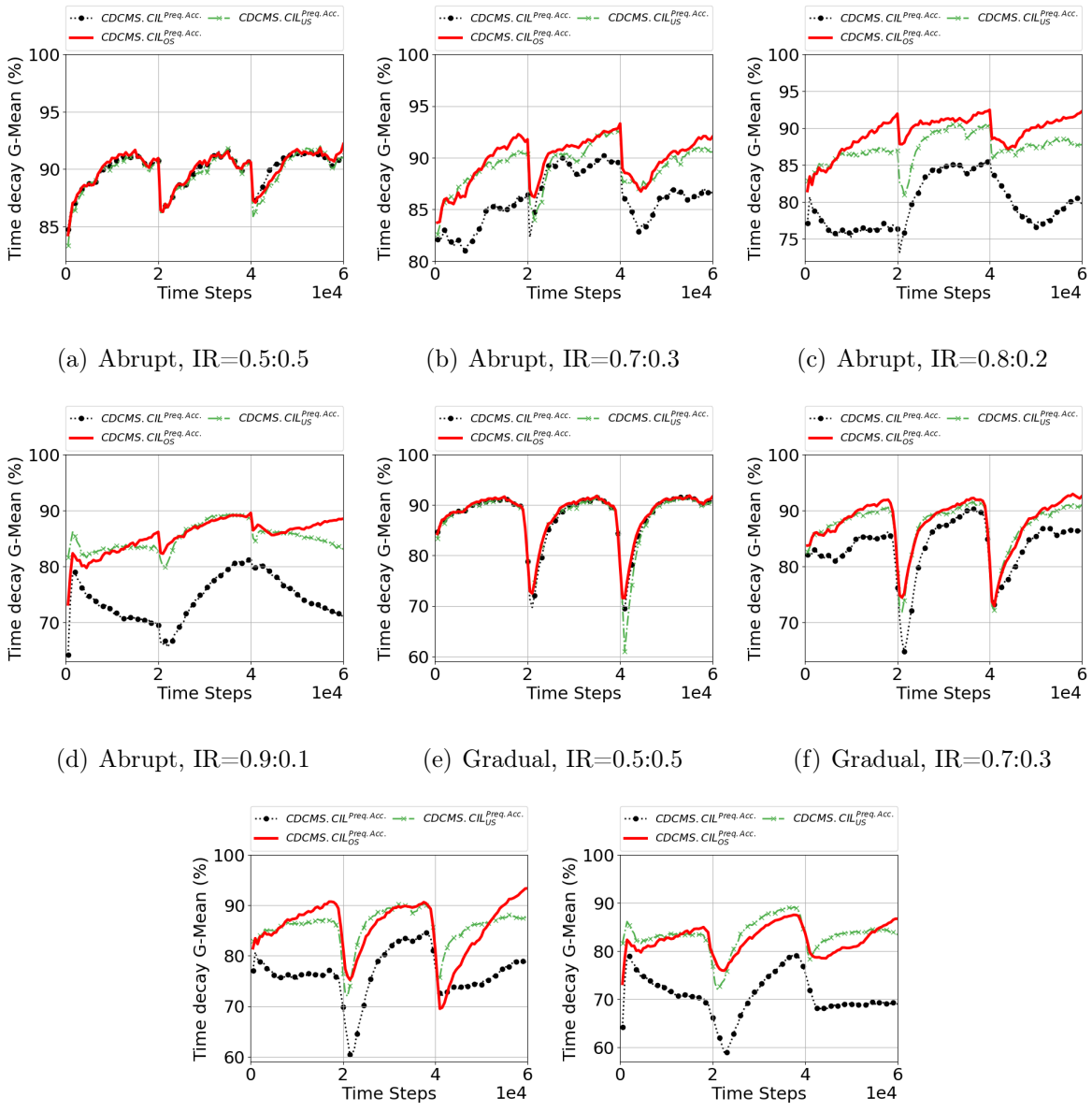
5.3.2.1 Impact of Different Resampling Methods to CDCMS.CIL

This section investigates the impact of different resampling methods to CDCMS.CIL by comparing the predictive performance of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$, CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ and CDCMS.CIL$^{\text{Preq. Acc.}}$. Specifically, we are interested in how do they impact the effectiveness of the proposed memory strategies in terms of concept drift adaptation.

Table 5.8 shows that the overall rankings of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ in time decay G-Mean performance is the best one, followed by CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ then CDCMS.CIL$^{\text{Preq. Acc.}}$. These results indicate that a resampling method is indeed required to enable CDCMS.CIL performs well in class imbalanced data streams.

To further confirm this deduction, Sine streams are chosen to study the their time decay G-Mean performance further. The reason for choosing Sine stream is because the time decay G-Mean rankings of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$, CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ and CDCMS.CIL$^{\text{Preq. Acc.}}$ present a similar trend to their overall rankings, as shown on Table 5.8. Also, Tables 5.6 shows that all these approaches employed a drift detector for Sine streams. Thus, analysing Sine streams allows us to obtain a general idea on when and why oversampling worked better than undersampling in CDCMS.CIL. Figures 5.9 and 5.10 present the time decay G-Mean of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$, CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ and CDCMS.CIL$^{\text{Preq. Acc.}}$ in Sine1 and Sine2 streams respectively.

Figure 5.9 and 5.10 show that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ and CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ per-



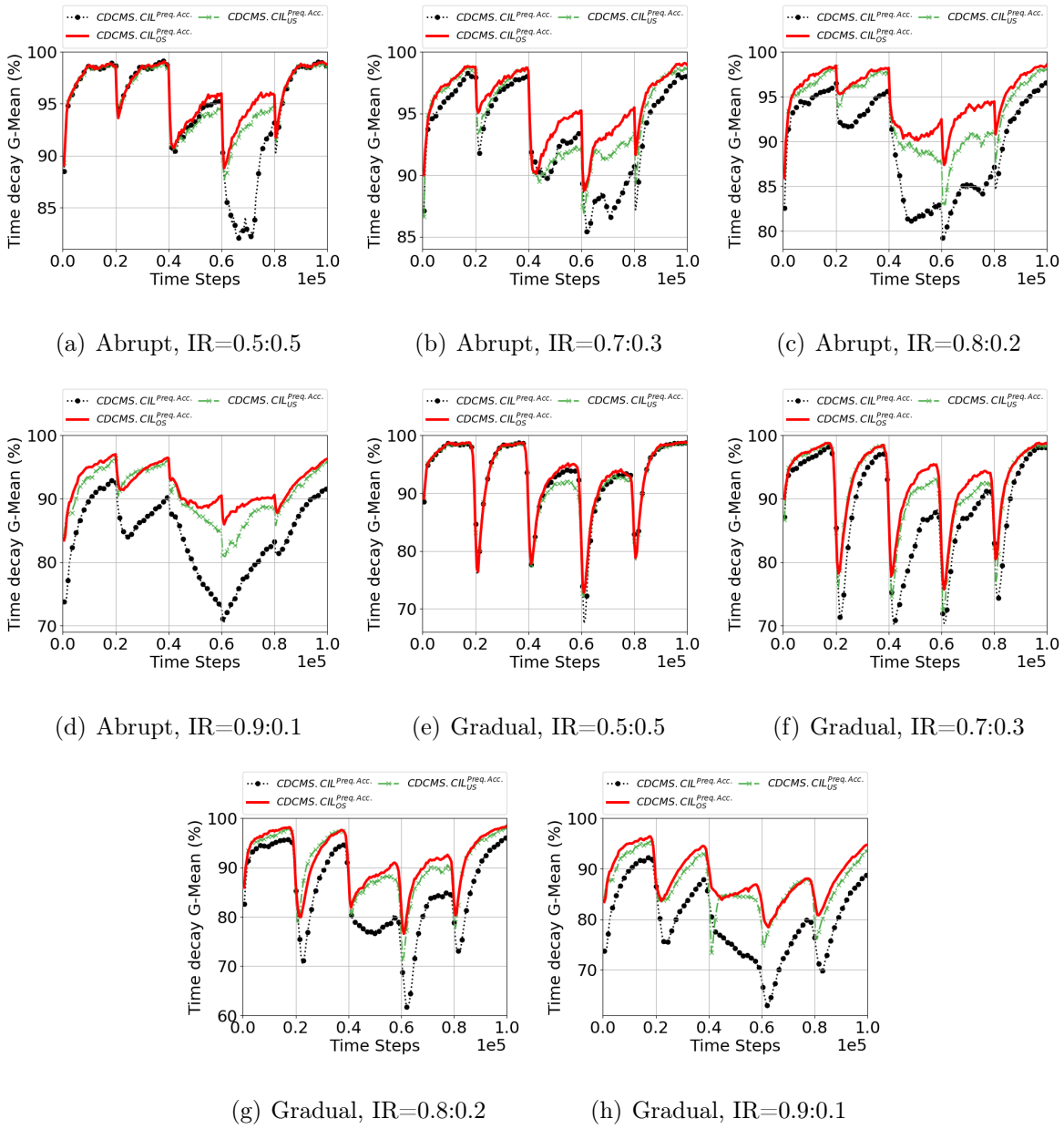
(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure 5.9: CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ vs CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ vs CDCMS.CIL$^{\text{Preq. Acc.}}$: Sine1 streams (Time Decay G-Mean)

formed better than CDCMS.CIL$^{\text{Preq. Acc.}}$ in the most Sine streams. CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ was usually the best one in reacting to and recovering from the drift, except in cases of Sine1-gradual (0.8:0.2) and Sine1-gradual (0.9:0.1). Figures 5.9(g) and 5.9(h) show that



(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure 5.10: CDCMS.CIL$_{OS}^{Preq.\ Acc.}$ vs CDCMS.CIL$_{US}^{Preq.\ Acc.}$ vs CDCMS.CIL$^{Preq.\ Acc.}$: Sine2 streams (Time Decay G-Mean)

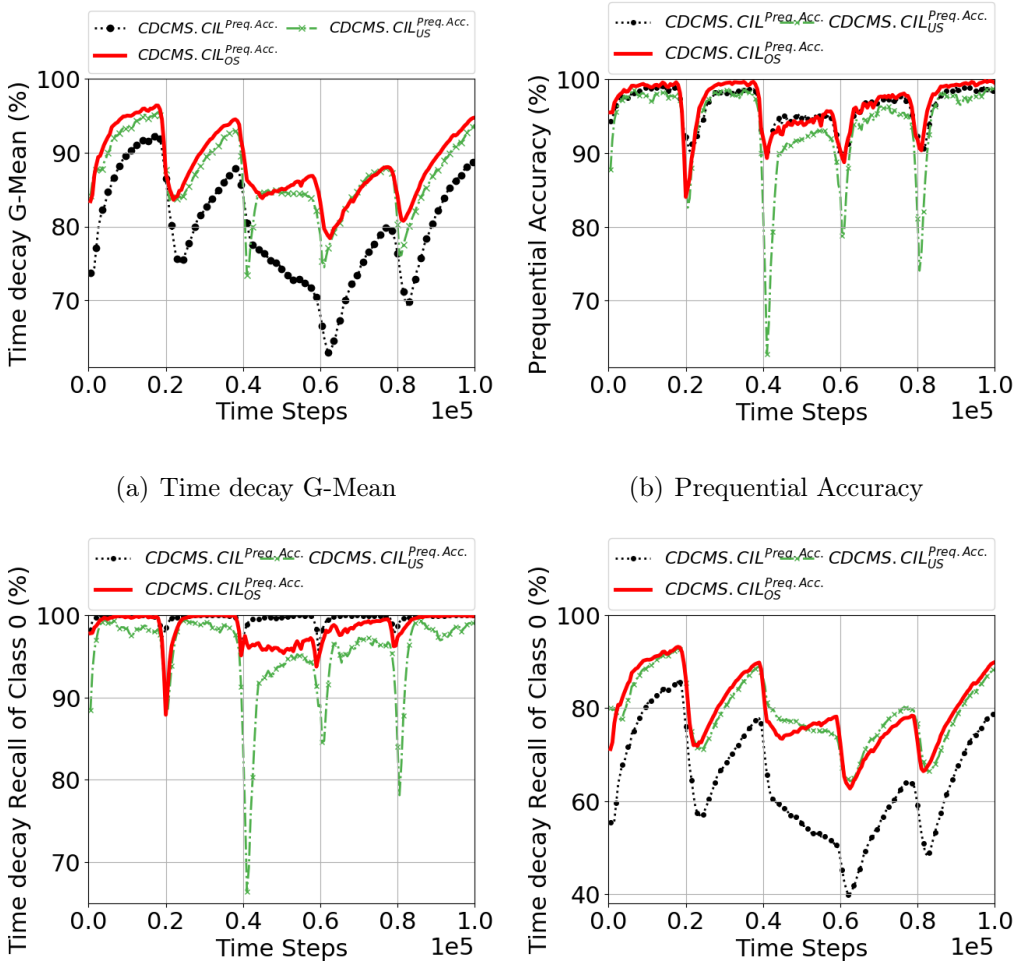
CDCMS.CIL$_{OS}^{Preq.\ Acc.}$ either had late drift detection or recovered worse than CDCMS.CIL$_{US}^{Preq.\ Acc.}$ in these two cases. In particular, the time decay G-Mean of CDCMS.CIL$_{OS}^{Preq.\ Acc.}$ dropped worse than that of CDCMS.CIL$^{Preq.\ Acc.}$ at the second drift of Sine1-gradual (0.8:0.2).

Based on the above observations, the case of Sine1-gradual (0.8:0.2) was chosen to be the representative case, further discussing why CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$, sometimes, performed poorly in terms of reaction to and recovery from drift. Meanwhile, Sine2-gradual (0.9:0.1) was chosen to be the representative case where CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ performed better because it is the most challenging Sine stream (consists of the most number of Sine gradual drifts and the class imbalance is 0.9:0.1, which is severe) that shows CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ performed well.

The discussion on representative cases where CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ performed better and worse are presented in Section 5.3.2.1.1 and 5.3.2.1.2 respectively.

5.3.2.1.1 Case where CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ performed better: This section compares the predictive performance between CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$, CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ and CDCMS.CIL$^{\text{Preq. Acc.}}$ on the representative case, Sine2 gradual (0.9:0.1) stream, to investigate why oversampling works better than undersampling and no resampling in CDCMS.CIL. Figure 5.11 presents the time decay G-Mean, class 0 and class 1 recalls and prequential accuracy of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$, CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ and CDCMS.CIL$^{\text{Preq. Acc.}}$ on the Sine2 gradual (0.9:0.1) stream. The concept sequence of this stream is: Sine-f1 $\rightarrow$ Sine-f2 $\rightarrow$ Sine-f3 $\rightarrow$ Sine-f4 $\rightarrow$ Sine-f1. The behaviours of the approaches are described as follows.

At the first drift (f1 $\rightarrow$ f2), all approaches had several drift detections with the last one given near the central point of the drift. CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ recovered three models that learnt from the concept Sine-f1 briefly, while CDCMS.CIL$^{\text{Preq. Acc.}}$ recovered just one model that learnt from the concept Sine-f1 briefly. CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ is recovered five models that learnt near the start of this drift. The reason for recovering these models was because their drift detection were all given near the start of the drift, meaning that they did not receive enough examples from the concept after the drift to identify if the new concept is a new one. Therefore, Figure 5.11(a) shows that the time decay G-Mean of all approaches



(a) Time decay G-Mean

(b) Prequential Accuracy

(c) Time decay Recall in Class 0

(d) Time decay Recall in Class 1

Figure 5.11: CDCMS.CIL$_{OS}^{Preq.\ Acc.}$ vs CDCMS.CIL$_{US}^{Preq.\ Acc.}$ vs CDCMS.CIL$^{Preq.\ Acc.}$: Sine2 gradual (0.9:0.1) stream

further dropped until around the end of this drift (around 21k time steps), indicating that the models recovered by all the approaches were not helpful in adapting to this drift. This also means that the later performance recovery at around 21k time steps was because of the simple strategy of periodic substitution in the main ensemble. The fact that CDCMS.CIL$_{OS}^{Preq.\ Acc.}$ and CDCMS.CIL$_{US}^{Preq.\ Acc.}$ recovered better than CDCMS.CIL$^{Preq.\ Acc.}$ confirms the necessity of resampling method.

At the second drift (f2 → f3), CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ and CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ had several drift detections over the whole transitional period of this drift, whereas CDCMS.CIL$^{\text{Preq. Acc.}}$ only had one drift detection around the central point of this drift. CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ and CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ both recovered fourteen models that learnt from Sine-f1 briefly, while CDCMS.CIL$^{\text{Preq. Acc.}}$ only recovered three of them. This behaviour of recovering models that learnt from Sine-f1 is reasonable because the concept after this drift is Sine-f3, which is similar to Sine-f1 (26.8% differences) but different from Sine-f2 (73.2% differences). The fact that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ and CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ recovered more models of concept Sine-f1 than CDCMS.CIL$^{\text{Preq. Acc.}}$ indicates that the proposed clustering in the model space strategy for class imbalance learning had more confidence in recovering the knowledge of Sine-f1 to adapt to Sine-f3 when a resampling method is employed.

Figure 5.11(a) show that CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ had a significant drop in time decay G-Mean right after the second drift. Figures 5.11(c) and Figure 5.11(d) reveals that CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ sacrificed a lot of performance in class 0 to maintain its performance in class 1. The reason behind this behaviour can be explained as follows.

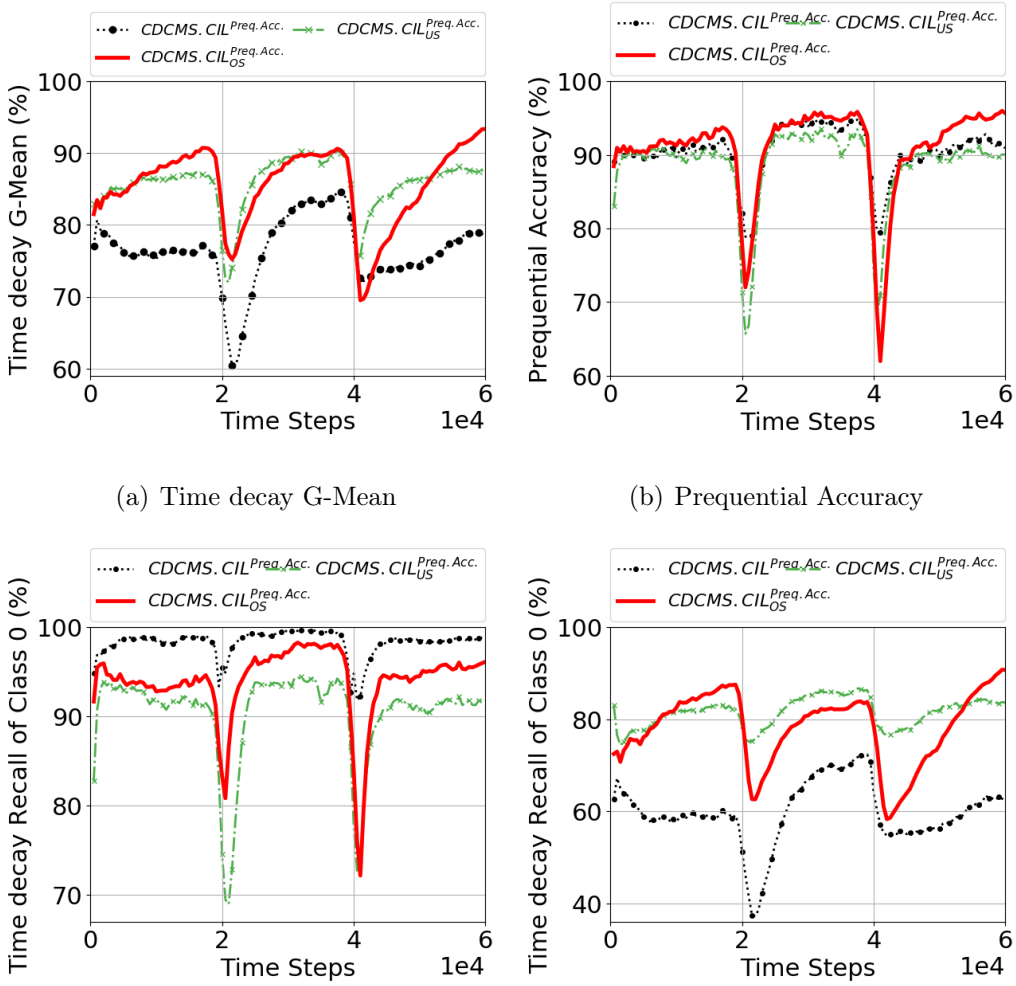
Given this data stream has a severe class imbalance ratio of 0.9:0.1, the undersampling rate used by CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ was likely aggressive because this rate is determined by the class imbalance ratio. Thus, lots of useful information in the majority class could be ignored in order to balance learning from both classes. Therefore, the base learners could not learn well in the majority class. When the base learners were then recovered to adapt to a similar concept, they could only help to prevent a significant drop in minority class performance but not the majority class. In contrast, CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ which uses oversampling did not have this issue because no information was ignored during learning. Although Figure 5.11(d) shows that the time decay class 1 recall of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ dropped slightly more than that of CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ right after the second drift, Figure 5.11(c) shows that it maintain the time decay class 0 recall better than CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$.

The behaviour of all three approaches at third and forth drifts can be described similar to their behaviour at the second drift.

5.3.2.1.2 Case where CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ performed worse: This section discuss the representative case where CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$, sometimes, performed worse in terms of reaction to and recovery from drift. Figure 5.12 presents the time decay G-Mean, class 0 and class 1 recalls and prequential accuracy of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$, CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ and CDCMS.CIL$^{\text{Preq. Acc.}}$ on the Sine1-gradual (0.8:0.2) stream. The concept sequence of this stream is: Sine-f3 $\rightarrow$ Sine-f4 $\rightarrow$ Sine-f3. The behaviours of the approaches are described as follows.

The first drift (f3 $\rightarrow$ f4) of Sine1 gradual (0.8:0.2) is extremely severe. According to Table A.1, the concepts of Sine-f3 and Sine-f4 have 100% differences, a distinctive intermediate concept was likely to form during the transitional period of this drift. During the drift, CDCMS.CIL$^{\text{Preq. Acc.}}$ and CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ had several drift detections since the beginning of this drift (around 1k time steps before the central drift point) until the end of this drift (around 1k time steps after the central drift point), while CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ had several drift detections since the beginning of this drift until the central drift point. At last, CDCMS.CIL$^{\text{Preq. Acc.}}$ recovered three models that learnt from the transitional period of this drift. CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ behaved similarly to CDCMS.CIL$^{\text{Preq. Acc.}}$, it just recovered one model that learnt from the transitional period of this drift. As for CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$, it recovered two models that were created and learnt right before the start of this drift, and a model that was created and learnt at the beginning of this drift.

As we can see from the behaviours of approaches, the proposed clustering in the model space strategy was able to recovered relevant past models based on the current concept. However, the proposed clustering in the model space strategy for class imbalanced learning considered the multiple drift detections to as the recurrence of the intermediate



(a) Time decay G-Mean

(b) Prequential Accuracy

(c) Time decay Recall in Class 0

(d) Time decay Recall in Class 1

Figure 5.12: CDCMS.CIL$_{OS}^{Preq.\ Acc.}$, CDCMS.CIL$_{US}^{Preq.\ Acc.}$ and CDCMS.CIL$^{Preq.\ Acc.}$: Sine1 gradual (0.8:0.2) stream

concept of Sine-f3 and Sine-f4, thus it kept recovering the models holding the knowledge of this intermediate concept. However, this intermediate concept may still be different from the the post-drift concept (Sine-f3), therefore, it was not very helpful in adapting to it. Optimistically, only a few models had been recovered, the main ensemble still had space for creating new base learners to learn from the the post-drift concept. This is why Figure 5.12 shows that all three of them were still able to adapt to this drift afterwards.

Figure 5.12 presents performance differences between using different resampling method. It shows that CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ adapted to this drift better than CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ and CDCMS.CIL$^{\text{Preq. Acc.}}$ (around 21k-30k time steps). This indicates that undersampling may be a better strategy than oversampling when the post-drift concept needs to learn from scratch.

At the second drift (f3 $\rightarrow$ f4), which is a severe recurrent gradual drift leading back to the concept of Sine-f4, all three approaches also had several drift detections. CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ and CDCMS.CIL$^{\text{Preq. Acc.}}$ had the last drift detection before the central point of this drift. Therefore, they both recovered lots of models holding the knowledge of Sine-f3. In addition, CDCMS.CIL$^{\text{Preq. Acc.}}$ also recovered a few models that learnt from the concept of Sine-f4 before 20k time steps. These behaviour of model recovery was, again, because there were not enough examples to determine whether this post-drift concept was novel, recurring or similar to a past concept. Thus, they both did not adapt to this drift well, especially CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$. In contrast, CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$ had the last drift detections near the end of this drift. Therefore, it recovered lots of models that learnt from the concept of Sine-f4 before 20k time steps and handled this drift the best, as shown in Figure 5.12.

5.3.2.2 Short Summary

This section first presents a quantitative analysis to compare the predictive performance of CDCMS.CIL using different resampling methods and weighting metrics. It shows that prequential accuracy with oversampling worked the best in CDCMS.CIL. In Section 5.3.2.1, we further analyse the impact of resampling method to the proposed memory strategies in dealing with concept drift in class imbalanced data streams. It shows that oversampling helps CDCMS.CIL to deal with class imbalanced learning without losing any useful information, whereas undersampling ignores lots of potentially useful information in the majority class.

Therefore, the base learners trained with oversampling can be more helpful in dealing with concept drift leading to recurring concepts or concepts that is similar to a previous one when they are correctly recovered. Yet, in the case where the proposed clustering in the model space strategy for class imbalanced learning does not manage to identify relevant past knowledge, using undersampling in CDCMS.CIL may be a better choice because of its aggressive nature to provide swift adaption in the minority class.

5.3.3 Comparing CDCMS.CIL against Existing Approaches

This section compares the performance of the proposed approach against existing approaches in artificial and real-world data streams and discusses the potential reason behind the results. We adopted CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ in this comparison because the analysis in Section 5.3.2 has shown that the effectiveness of proposed memory strategies for class imbalanced learning can mostly manifested when using prequential accuracy with oversampling.

Following the experiment procedure presented in Section 5.2 and 5.3.2, each approach ran for thirty times on each data stream, using different random seeds. Their performance were then grouped by class imbalance ratio, drift speed and stream type respectively. At last, Friedman test was performed to statistically compare their performance in each group. Nemenyi post-hoc test was also performed to detect statistical significance between each pair of approaches' performance in each group. Null hypothesis is rejected when the p-value ≤ 0.05. The performance rankings of the approaches grouped by factors in time decay G-Mean, class 0 and class 1 recalls are presented on Tables 5.11, 5.12 and 5.13 respectively. The smallest rank refers to the best performance. This analysis mainly based on their performance rankings in time decay G-Mean.

Table 5.11 shows that, CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ achieved the best overall ranking in time

Table 5.11: Statistical (Friedman) Ranking of Time Decay G-Mean on Streams Grouped by Factors (Comparison against Existing Approaches)

Groups	GH-VFDT$_d$	HD-VFDT$_d$	Online Bagging$_d$	OOB$_d$	UOB$_d$	CDCMS	CDCMS.CIL$_{OS}^{Preq.\ Acc.}$
Grouped by imbalance ratio (Artificial data streams)							
0.5:0.5	5.4033	5.5017	3.2483	3.9133	4.9067	3.055	1.9717
0.7:0.3	5.3258	5.4892	5.2333	2.5267	2.9817	4.7083	1.735
0.8:0.2	5.4342	5.7708	5.5817	2.1267	2.2417	4.9117	1.9333
0.9:0.1	5.5758	5.8742	4.9317	2.4217	1.9683	5.135	2.0933
Grouped by concept drift speed							
Abrupt	5.4267	5.6392	4.5592	2.8367	3.2908	4.5342	1.7133
Gradual	5.4429	5.6788	4.9383	2.6575	2.7583	4.3708	2.1533
Real-World	4.65	4.7405	5.4286	3.3071	4.5024	3.2929	2.0786
Grouped by streams							
Sine	6.3083	6.6667	4.45	2.1	3.1708	3.7417	1.5625
Agrawal	5.9083	5.8625	4.0958	3.6833	3.5333	3.8583	1.0583
SEA	5.1979	5.2521	5.825	1.9208	2.5542	4.85	2.4
STAGGER	5.2	6.1708	4.4458	2.5667	2.1542	5.025	2.4375
Airlines	5.0	6.0	7.0	2.7667	2.3	3.9333	1.0
NOAA	5.0333	5.0333	6.9667	1.7667	4.7	3.2667	1.2333
Covtype	3.7333	3.6095	5.181	3.8048	4.5238	4.1667	2.981
INSECTS	5.7867	6.0133	5.1533	3.0267	4.8733	1.9467	1.2
All streams	5.3179	5.5222	4.85	2.8305	3.2447	4.2798	1.955

- The p-values of Friedman tests are all $\leq$2.2E-16.
- Highlighted ranks denote significant superior performance.

decay G-Mean, followed by OOB$_d$ and then UOB$_d$. This shows that the proposed memory strategies to deal with concept drift in class imbalanced data stream learning is generally helpful when it is supported by a resampling method. In contrast, CDCMS did not perform as good as CDCMS.CIL$_{OS}^{Preq.\ Acc.}$, not only because it did not employ any resampling method but also its memory strategies were not designed for class imbalanced learning.

Besides, Table 5.11 shows that UOB$_d$ generally performed worse than OOB$_d$. This coheres to the analysis in Section 5.3.2, which shows the fact that undersampling would be detrimental in most cases as it discard potentially useful information in the majority class. Its advantage of swift adaptation to new concept could only be shown when the data stream is severely class imbalanced.

Table 5.12: Statistical (Friedman) Ranking of Time Decay Class 0 (Majority Class) Recall on Streams Grouped by Factors (Comparison against Existing Approaches)

Groups	GH-VFDT$_d$	HD-VFDT$_d$	Online Bagging$_d$	OOB$_d$	UOB$_d$	CDCMS	CDCMS.CIL$_{OS}^{Preq.\ Acc.}$
Grouped by imbalance ratio (Artificial data streams)							
0.5:0.5	5.2883	5.3483	3.3392	4.2192	4.5433	3.26	2.0017
0.7:0.3	3.8758	3.7375	1.52	5.5833	6.7583	2.1367	4.3883
0.8:0.2	3.765	3.5	1.3033	5.6617	6.9317	2.305	4.5333
0.9:0.1	3.4367	3.3033	1.52	5.8383	6.99	2.285	4.6267
Grouped by concept drift speed							
Abrupt	4.2133	4.1642	1.9712	5.2362	6.2475	2.6792	3.4883
Gradual	3.9696	3.7804	1.87	5.415	6.3642	2.3142	4.2867
Real-World	4.3021	4.6021	2.824	3.9719	6.2208	2.9615	3.1177
Grouped by streams							
Sine	4.2938	4.3396	1.9667	5.325	6.3875	1.7333	3.9542
Agrawal	4.3083	3.575	1.4	5.725	6.6125	2.3333	4.0458
SEA	3.5354	3.5812	2.5333	6.0667	6.475	1.6833	4.125
STAGGER	4.0167	4.575	2.575	4.2625	5.5583	3.7625	3.25
Airlines	3.0	4.0	1.0	5.1	7.0	2.0	5.9
NOAA	6.5	6.5	1.0	4.3	4.6667	2.0	3.0333
Covtype	4.4762	4.8048	4.1095	3.6	6.2952	2.9429	1.7714
INSECTS	4.0	4.3	1.4067	3.92	7.0	2.9667	4.4067
All streams	4.1266	4.0773	2.0712	5.1	6.2917	2.5741	3.7592

- The p-values of Friedman tests are all $\leq$2.2E-16.
- Highlighted ranks denote significant superior performance.

Table 5.11 also shows that GH-VFDT$_d$ and HD-VFDT$_d$ performed worse than Online Bagging$_d$ and CDCMS. This result is quite unexpected because GH-VFDT$_d$ and HD-VFDT$_d$ are two decision trees designed for class imbalanced data streamlearning, but Online Bagging$_d$ and CDCMS are not. This potentially because GH-VFDT$_d$ and HD-VFDT$_d$ are single learners, whereas Online Bagging$_d$ and CDCMS are ensemble learners which usually yield better performance. Also, the fact that Online Bagging$_d$ and CDCMS have implicit strategy to learn diversely from the data streams might help them in dealing with class imbalance.

Looking into the classwise rankings of the approaches, Tables 5.12 and 5.13 show that CDCMS.CIL$_{OS}^{Preq.\ Acc.}$ usually achieved better rankings in the minority class (class 1) than the majority class (class 0). In particular, it was the second best performing approach in the minority class. These indicate that CDCMS.CIL$_{OS}^{Preq.\ Acc.}$ could emphasise the minority better

Table 5.13: Statistical (Friedman) Ranking of Time Decay Class 1 (Minority Class) Recall on Streams Grouped by Factors (Comparison against Existing Approaches)

Groups	GH-VFDT$_d$	HD-VFDT$_d$	Online Bagging$_d$	OOB$_d$	UOB$_d$	CDCMS	CDCMS.CIL$_{OS}^{Preq.\ Acc.}$
Grouped by imbalance ratio (Artificial data streams)							
0.5:0.5	4.9567	4.9933	3.3342	3.7775	4.805	3.5183	2.615
0.7:0.3	5.415	5.6467	5.4425	2.6992	1.2417	5.1183	2.4367
0.8:0.2	5.4283	5.8183	5.6817	2.42	1.1217	4.995	2.535
0.9:0.1	5.625	5.835	5.0683	2.5367	1.3717	5.2267	2.3367
Grouped by concept drift speed							
Abrupt	5.3908	5.6433	4.7025	2.9283	2.2542	4.7908	2.29
Gradual	5.3217	5.5033	5.0608	2.7883	2.0158	4.6383	2.6717
Real-World	4.5708	4.5104	5.3969	3.5948	3.0479	3.8375	3.0417
Grouped by streams							
Sine	6.3208	6.6375	4.4167	2.6083	1.9792	3.7708	2.2667
Agrawal	5.8208	5.7083	4.75	3.55	2.4167	3.975	1.7792
SEA	5.3333	5.3417	5.1375	1.8917	1.8625	5.4625	2.9708
STAGGER	5.0042	6.0667	4.5771	2.7979	2.0625	5.05	2.4417
Airlines	6.0	4.0	7.0	3.0	1.0	5.0	2.0
NOAA	1.6	1.6	6.9667	3.3	4.7	5.9333	3.9
Covtype	3.7048	3.9333	5.1476	4.0286	3.881	4.3238	2.981
INSECTS	6.1133	6.0	5.8	3.58	1.0	3.1333	2.3733
All streams	5.2253	5.3962	4.9675	2.9811	2.2872	4.5684	2.5743

- The p-values of Friedman tests are all $\leq$2.2E-16.
- Highlighted ranks denote significant superior performance.

than most of the other approaches without sacrificing too much performance in the majority class in comparison to other approaches. Meanwhile, Tables 5.12 and 5.13 show that OOB$_d$ and UOB$_d$ generally achieved better rankings in the minority class (class 1) than in the majority class (class 0), but Table 5.11 show that their rankings in time decay G-Mean were generally not similar or better than CDCMS.CIL$_{OS}^{Preq.\ Acc.}$. These indicates that OOB$_d$ and UOB$_d$ may shifted the learning focus from the majority class to the minority class more than CDCMS.CIL$_{OS}^{Preq.\ Acc.}$. Another potential reason could be the fact that CDCMS.CIL$_{OS}^{Preq.\ Acc.}$ exploits past knowledge to deal with concept drift in class imbalanced data streams but OOB$_d$ and UOB$_d$ do not.

Further focusing on the time decay class 1 recall, Table 5.13 shows that UOB$_d$ performed the best because undersampling sacrifices the predictive performance in the majority

class to strive for good predictive performance in the minority class. This aggressiveness is manifested in UOB_d as it always needs to learn the post-drift concept from scratch. This nature of undersampling has been discussed in Section 5.3.2.

Table 5.12 shows that Online Bagging_d generally performed the best in the majority class (class 0) time decay recall, followed by CDCMS. This is because they were not designed for class imbalanced learning, thus have learning bias towards the majority class. Note that, Tables 5.12 and 5.13 show that the rankings of Online Bagging_d and CDCMS are better than that of GH-VFDT_d and HD-VFDT_d in both classes. This may further confirm that building a diverse ensemble could be more important than using a single learner with strategy to deal with class imbalance.

It is also worth to mention that experiments with the original version of GH-VFDT, HD-VFDT, Online Bagging, OOB and UOB (without drift detection) had also been performed to compare against the proposed approach. The results in time decay G-Mean, majority and minority class recalls are shown in Tables G.1, G.2 and G.3 in the Appendix G, respectively. However, these results show that employing a drift detector would generally improve their performance in all metrics.

In the next sections, we further discuss the performance of the approaches by different data stream types. In particular, Sine and Agrawal streams were chosen to be the representative artificial stream types because the performance of approaches in STAGGER and SEA can be explained similarly.

5.3.3.1 Sine Streams

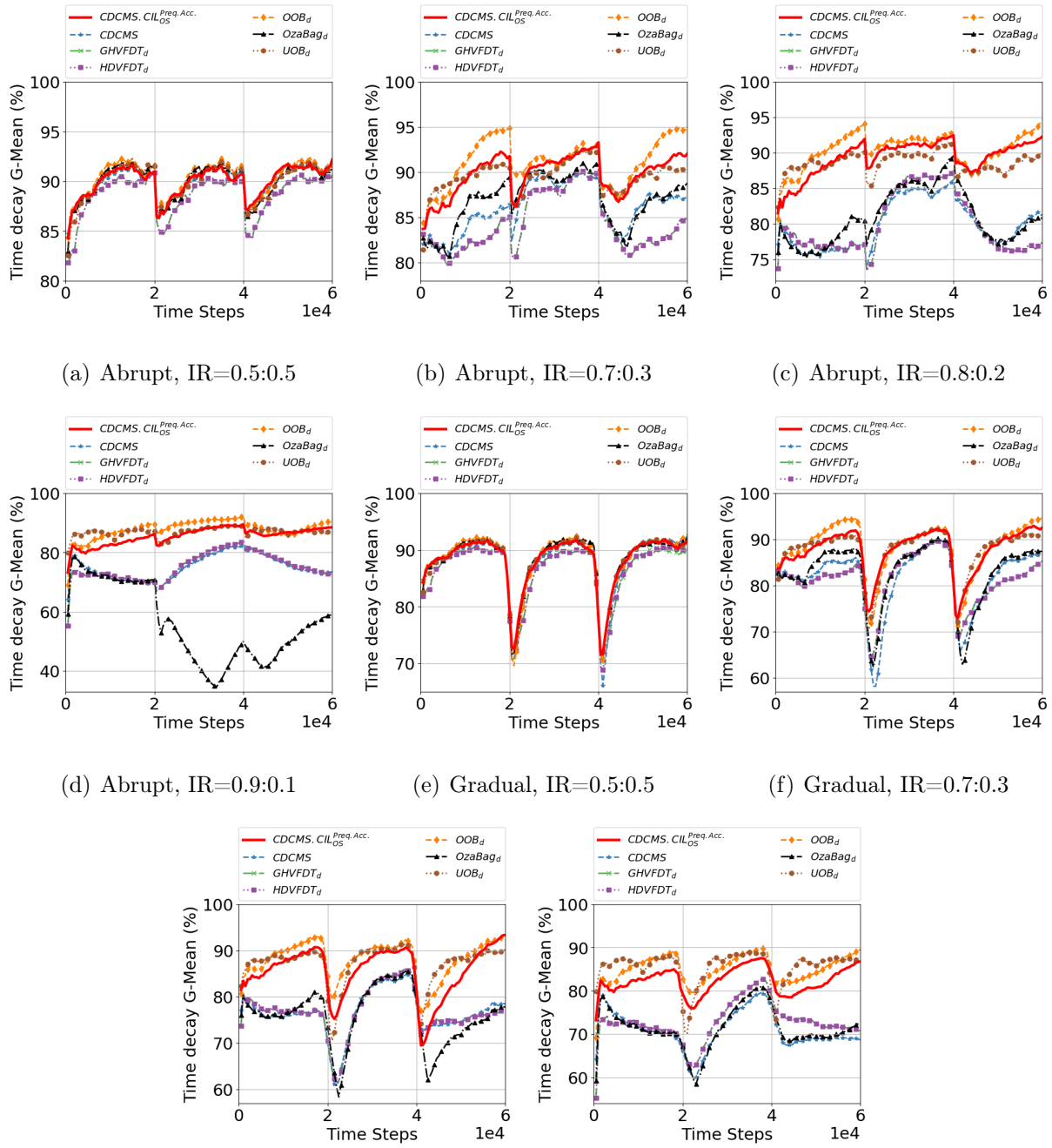
Table 5.11 shows that $\text{CDCMS.CIL}_{\text{OS}}^{\text{Preq. Acc.}}$ and OOB_d are the best performing approaches in Sine streams. Figure 5.13 shows that OOB_d usually performed the best in Sine1 streams

and Figure 5.14 shows CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ usually performed the best in Sine2 streams. Considering both Figures 5.13 and 5.14, they show that performance of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ and OOB$_{\text{d}}$ were generally similar because they both adopted oversampling to deal with class imbalance.

Figures 5.13 and 5.14 show that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ usually is the best one in recovering time decay G-Mean from concept drift because it managed to recover relevant past knowledge to deal with drifts. The detailed behaviour of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ is discussed in Section 5.3.2.1. OOB$_{\text{d}}$ and UOB$_{\text{d}}$ could also recover their time decay G-Mean from drifts, but they usually took longer period than CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$. Online Bagging$_{\text{d}}$ and CDCMS performed worse than CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ because they were not designed for class imbalanced learning. GH-VFDT$_{\text{d}}$ and HD-VFDT$_{\text{d}}$ performed the worst likely because they are not ensemble approaches. These results, again, show that, in learning drifting class imbalanced data streams, a diverse ensemble plays a more important role that solely applying strategy to deal with class imbalance in a single learner.

In the highly class imbalance cases (imbalance ratio of 0.9:0.1), Figures 5.13 and 5.14 show that OOB$_{\text{d}}$ usually performed slightly better than CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$. This could be the fact that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ suffered from early drift detection, which leads to recover irrelevant past models. This disadvantage of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ has been discussed in Section 5.3.2. Therefore, learning the post-drift concept from scratch will certainly perform better than trying to converge irrelevant past models.

Speaking of learning the post-drift concept from scratch, Figures 5.13(g), 5.13(h) and 5.14(h) shows that, in highly class imbalance Sine streams, UOB$_{\text{d}}$ was the best one in recovering time decay G-Mean. Yet, this behaviour was not presented in other Sine streams with less severe class imbalance ratio. These results further confirm that the strength of undersampling in learning class imbalanced concepts from scratch could only be shown when



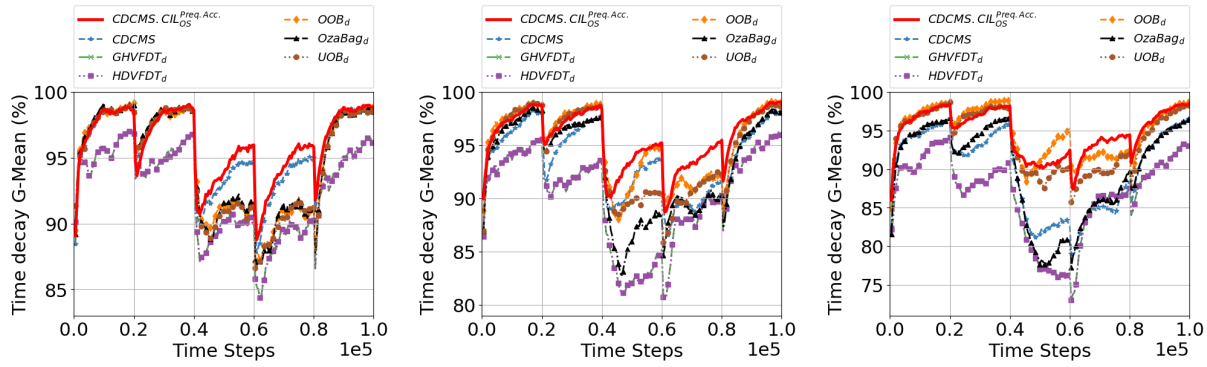
(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

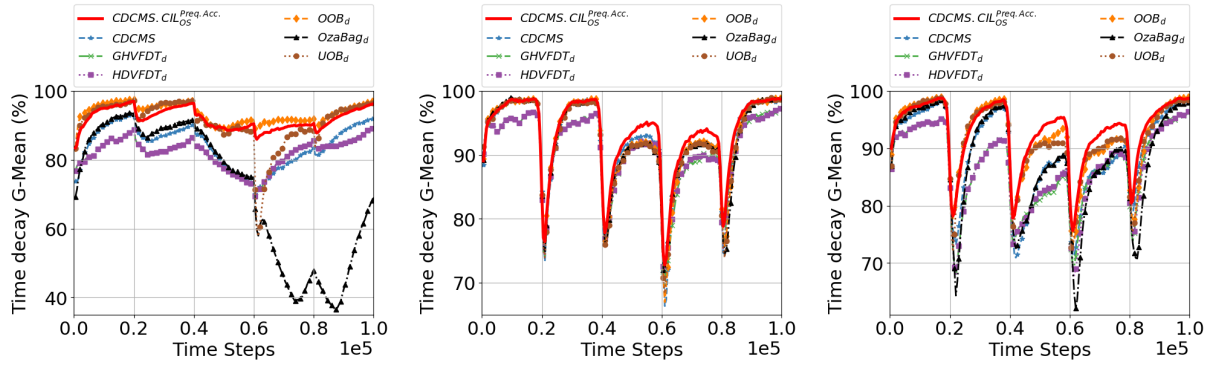
(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure 5.13: Comparing CDCMS.CIL against Existing Approaches: Sine1 (Time decay G-Mean)

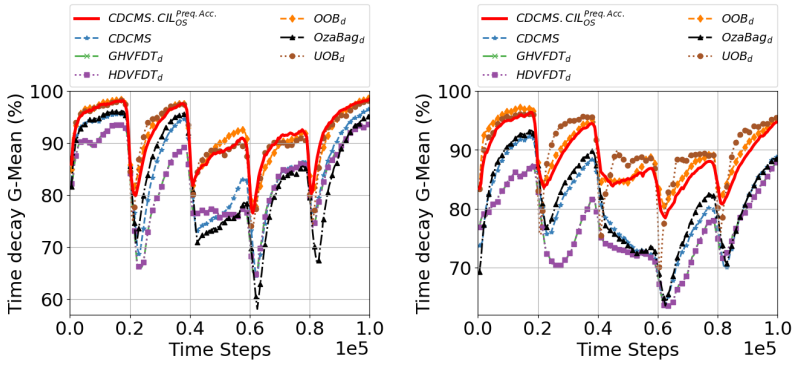
the data stream is severely class imbalanced, cohering the discussion in Section 5.3.3.



(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2



(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3



(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure 5.14: Comparing CDCMS.CIL against Existing Approaches: Sine2 (Time decay G-Mean)

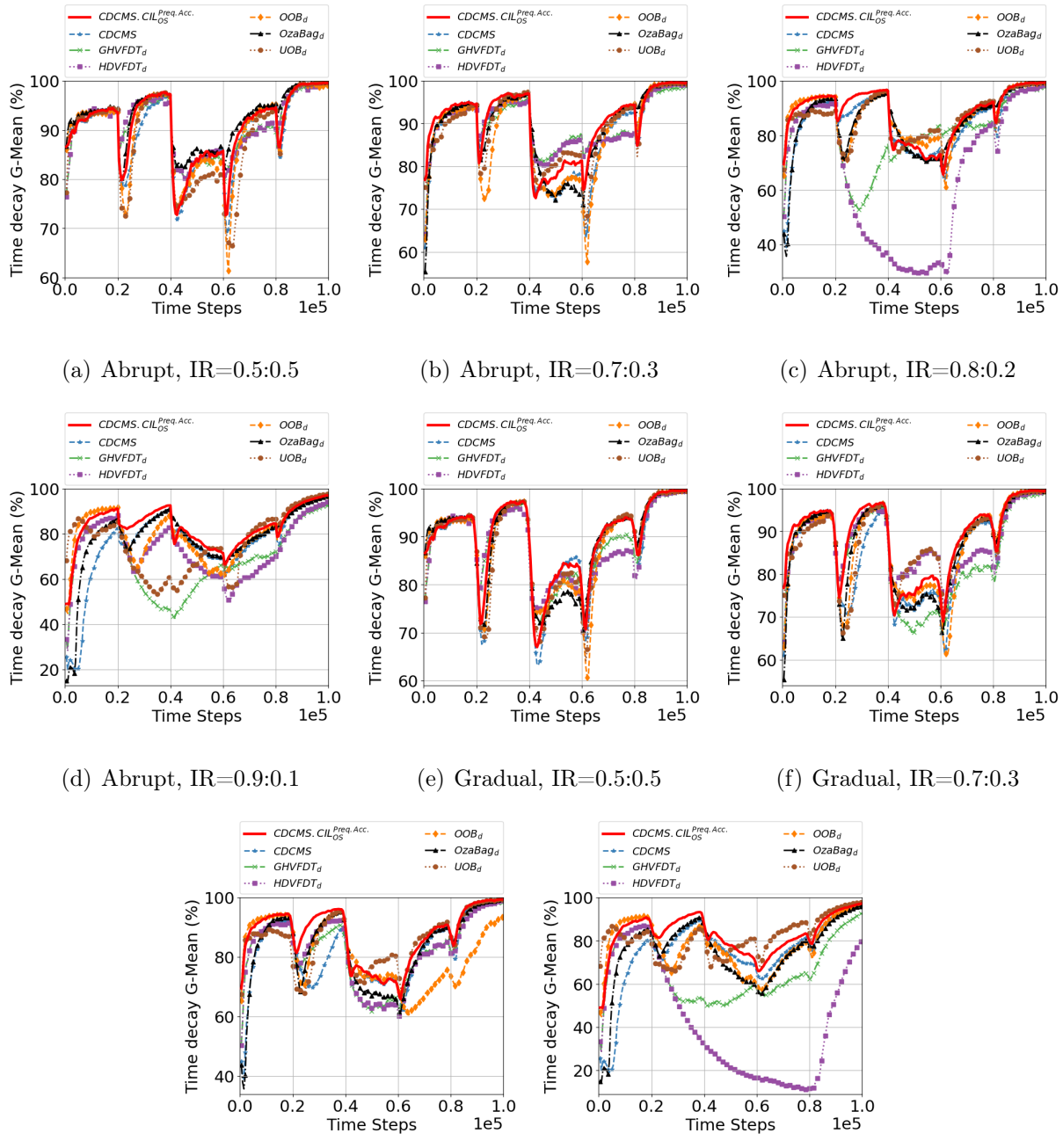
5.3.3.2 Agrawal Streams

Tables 5.11 and 5.13 show that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ performed the best in terms of both time decay G-Mean and minority class recall. Table 5.12 shows that Online Bagging$_{\text{d}}$ performed the best in the majority class. Note that, as shown on Tables 5.5 and 5.6, CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ and CDCMS did not employ any drift detector for Agrawal streams, whereas the other approaches did employ one. This means that the superior performance by CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ in Agrawal streams was attributed to the simple strategy of periodically introducing new base learners to the main ensemble and the resampling method. This is a surprising result. While CDCMS also relied on a similar training strategy, it has no strategy to deal with class imbalance. Therefore, it could not performed as good as CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$, cohering the findings in Section 5.3.1.

To further discuss the performance of the approaches, Agrawal1 streams were chosen to be example cases. Their performance in Agrawal2, Agrawal3, and Agrawal4 can be explained similarly. Figures 5.15 presents the time decay G-Mean of the approaches on different Agrawal1 streams respectively.

When comparing CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ against OOB$_{\text{d}}$ and UOB$_{\text{d}}$, Figure 5.15 shows that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ usually performed better in most stable concept periods. It also adapted to the concept drifts quite well in cases, such as Agrawal1 abrupt (0.9:0.1) stream (Figure 5.15(h)). The reason could be the drifts were not very severe or the underlying concept could be easy to learn. Considering these potential reasons and the hyper-parameter tuning results, they indicates that the simple strategy of periodically introducing new base learners to the main ensemble and the resampling method were sufficient to deal with the concept drift and class imbalance in Agrawal streams.

Nevertheless, we can still see the benefits of detecting concept drifts in Agrawal



(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure 5.15: Comparing CDCMS.CIL against Existing Approaches: Agrawal1 (Time decay G-Mean)

streams as done by OOB_d and UOB_d. For examples, Figure 5.15(h) shows that UOB_d adapted to the third drift of Agrawal1 gradual (0.9:0.1) stream quite well. Yet, in general,

resetting the system upon drift detection is detrimental to the predictive performance.

5.3.3.3 Real-World Data Streams

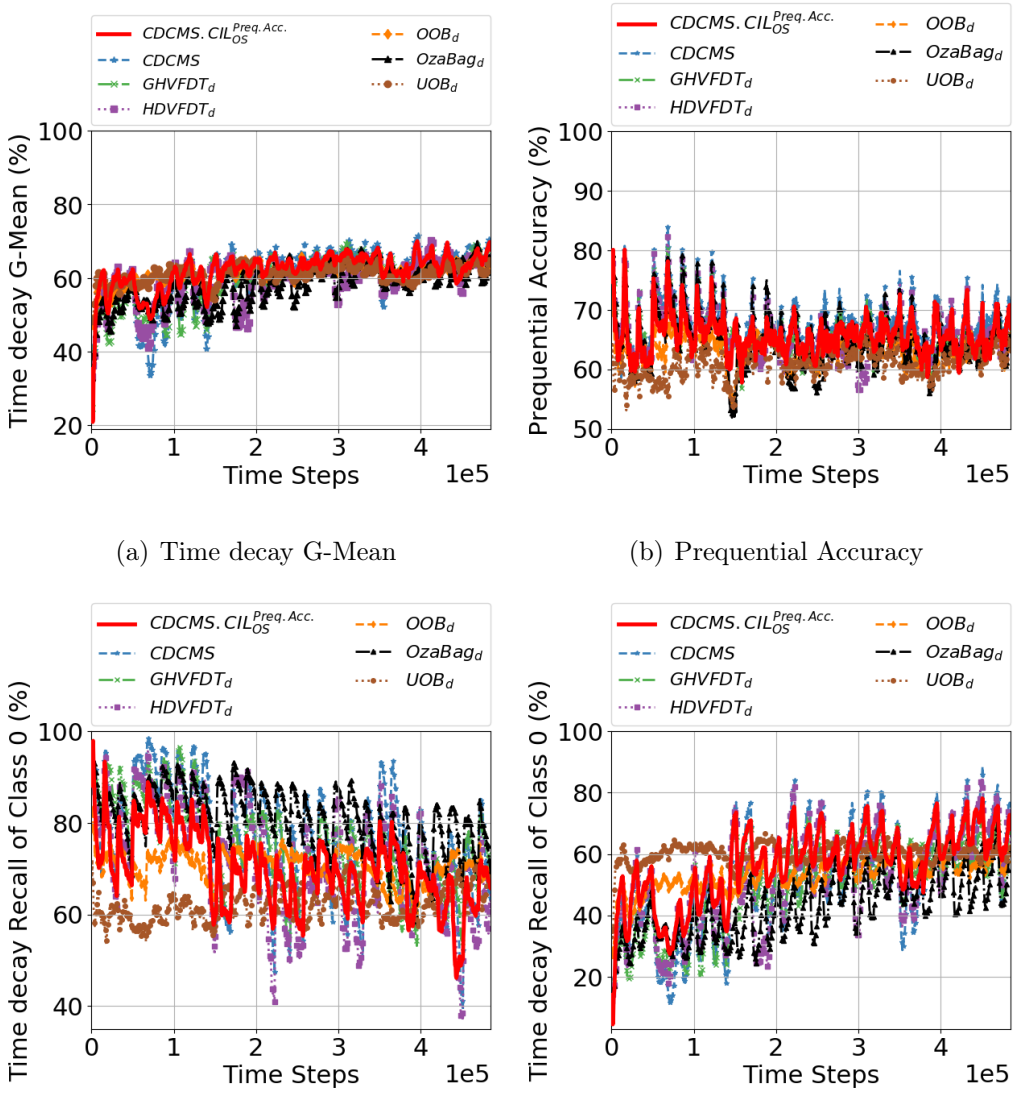
Tables 5.11 shows that, in terms of time decay G-Mean, CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ is one of the best performing approaches in all real-world data streams. Concerning the classwise performance, Table 5.13 shows that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ is one of the top-ranked approaches in the minority class (class 1). Table 5.12 shows that it is one of the top-ranked approaches in the majority class (class 0) in NOAA and Covtype streams. These results show that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ can usually performed better than compared approaches in most real-world applications.

In the next sections, we further discuss the performance of the approaches in Airlines and NOAA streams as the representative cases. Their performance in Covtype and INSECTS streams can explained similarly.

5.3.3.3.1 Airline Stream: Focusing on the time decay G-Mean in Airlines stream, Tables 5.11 shows that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ and UOB$_{\text{d}}$ performed the best. Figure 5.16(a) presents the time decay G-Mean, class 0 and class 1 recalls and prequential accuracy of the approaches on Airline stream.

First of all, Figure 5.16 show that the performance of the approaches fluctuated a lot throughout the Airline stream, meaning that there might be lots of concept drift. Figure 5.16(a) shows that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ generally performed the best in terms of time decay G-Mean, supporting its high ranking in this stream (ranked 1.0, as shown on Table 5.11).

Figure 5.16(a) also shows that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ and CDCMS both continuously strove to improve their performances throughout the stream because they have a similar



(a) Time decay G-Mean

(b) Prequential Accuracy

(c) Time decay Recall in Class 0

(d) Time decay Recall in Class 1

Figure 5.16: Comparing CDCMS.CIL against Existing Approaches: Airlines Stream

training strategy of periodically introducing new base learners to the main ensemble. For CDCMS, we can see that there were quite a number of "peak" in performance since 150k time steps but it also had several drastic drops in performance prior to 150k time steps. On the other hand, the performance of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ was more stable than CDCMS. Such differences may due to the fact that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ employed a resampling method. It also indicates that the proposed diversity-based strategies for class imbalanced learning

are more robust to false-positive drift detection than its predecessor in Airlines stream. In addition, Figures 5.16(c) and 5.16(d) shows that the time decay class 0 and class 1 recalls of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ gradually converge to a similar level. This shows the ability of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ in balancing the predictive performance in both classes over time.

As for the other approaches, Figure 5.16 shows that the performance of OOB$_{\text{d}}$ and UOB$_{\text{d}}$ were quite stable throughout the stream. OOB$_{\text{d}}$ performed better than UOB$_{\text{d}}$ in time decay class 0 recall but it performed worse than UOB$_{\text{d}}$ in time decay class 1 recall. These results also cohere the discussion on the nature of oversampling and undersampling in Section 5.3.2.

Lastly, Figure 5.16(a) shows that Online Bagging$_{\text{d}}$, GH-VFDT$_{\text{d}}$ and HD-VFDT$_{\text{d}}$ struggled the most in Airline stream. However, Figures 5.16(c) and 5.16(d) reveals that they might struggle with different reasons. These two figures shows that Online Bagging$_{\text{d}}$ maintained a high performance in the time decay class 0 recall while maintained a relatively low performance in the time decay class 1 recall, meaning that a strategy is still needed to deal with class imbalance despite the fact that the class imbalance ratio of Airline is very mild (0.544:0.456, as shown in Table 5.2). On the other hand, Figures 5.16(c) and 5.16(d) show that GH-VFDT$_{\text{d}}$ and HD-VFDT$_{\text{d}}$ put so much effort to balance their predictive performance in both classes over time but their predictive performance were less stable than other approaches. This is likely because they are single learner approaches, which are usually less stable than ensemble approaches. Again, this results cohere the discussion in Section 5.3.3.1.

5.3.3.3.2 NOAA Stream: Table 5.2 shows that the class imbalance ratio of NOAA stream is mild, around 0.7:0.3. Table 5.11 and 5.13 show that OOB$_{\text{d}}$ and CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ performed the best in both time decay G-Mean and minority class (class 1) recall. The behaviour log shows that CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ had several drift detection throughout the stream but we cannot see any drastic change in the time decay G-Mean in Figure 5.17(a).

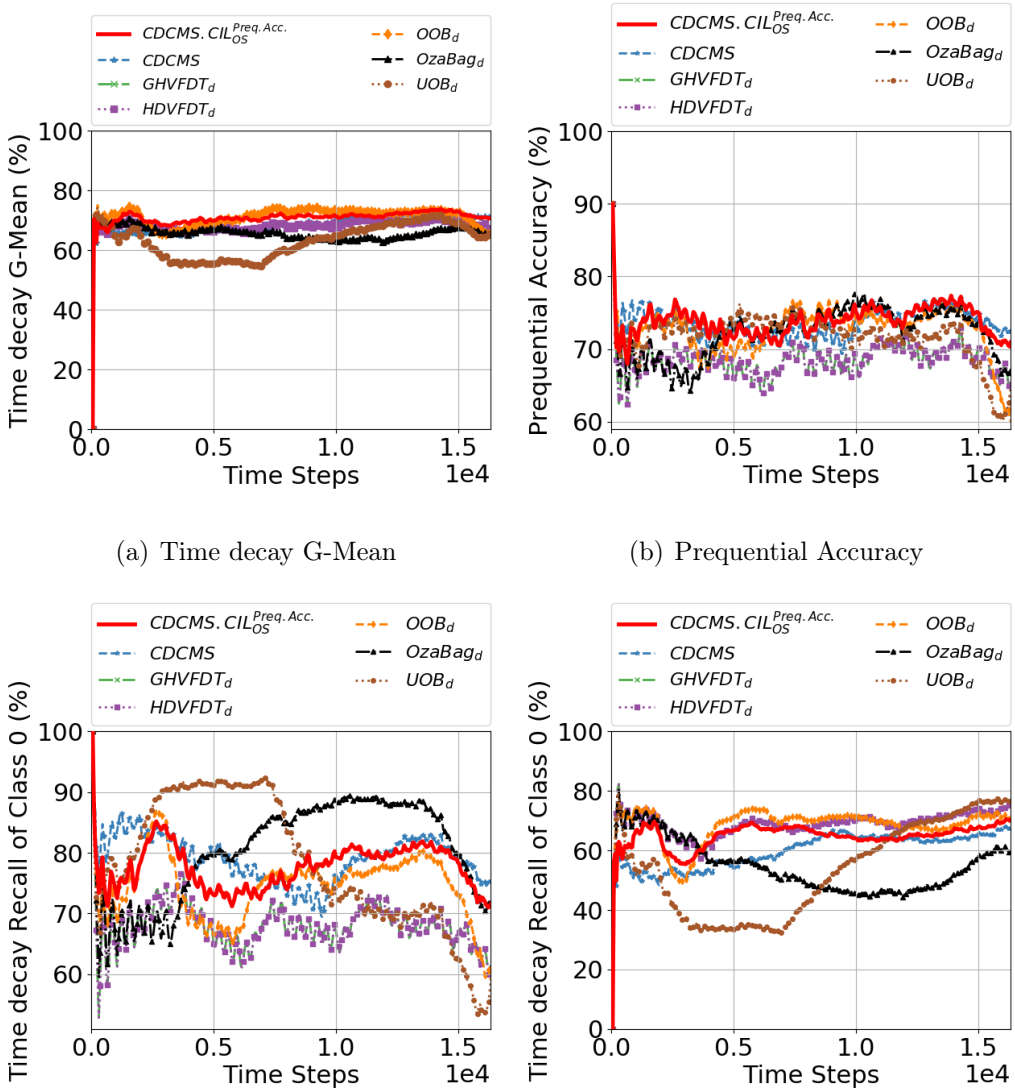
Therefore, most of these drift detections could be false-positives. This potentially shows the robustness of CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$ in handling false-positive drift detections.

For UOB$_{\text{d}}$, it only had drift detection at the beginning and the end of the data streams but we can see from Figure 5.17(a) that there was a period from 20k time steps to 70k time steps where it performed poorly. It then started to recover since 70k time steps. Figure 5.17(c) and 5.17(d) reveal that it performed very well in the majority class but poorly in the minority class during this period. As the class imbalance ratio of NOAA stream is mild, this indicates that the advantage of undersampling in learning the minority class only applies to learn severely class imbalanced data streams, cohering to the findings in Sections 5.3.2 and 5.3.3.1.

The performance of their approaches can be explained similarly in Section 5.3.3.3.1 on the case of Airline stream.

5.3.3.4 Short Summary

In this analysis, we quantitatively and qualitatively compared the performance of the proposed approach against six existing approaches in artificial and real-world data streams with different severity of concept drifts and class imbalance ratio. It has been shown that, the proposed approach is always one of the best performing approaches in terms of time decay G-Mean, considering different factors. This answers the RQ2.3 posed in the introduction of this chapter. In particular, the analysis has shown that the proposed memory strategies were usually effective in helping the system to recover from drifts and robust to false-positive drift detections. Meanwhile, in cases when the proposed approach had no drift detection or did not employ a drift detector, the simple strategies of periodically introducing new base learners to the main ensemble and training with a resampling method had already achieved a superior performance than existing approaches. Yet, in data streams which have very few



(a) Time decay G-Mean

(b) Prequential Accuracy

(c) Time decay Recall in Class 0

(d) Time decay Recall in Class 1

Figure 5.17: Predictive Performance on NOAA Stream

number of minority class examples or highly class imbalance, UOB_d recovered the predictive performance from the drifts better than the proposed approach.

5.4 Summary and Discussion

The main contribution of this chapter is the proposed diversity-based framework for class imbalanced learning. In particular, novel memory strategies are proposed to identify and recover relevant past knowledge, and manage a diverse memory, in spite of the class imbalance degree of the data stream (RQ2.1). This framework is also robust to false-positive drift detections. This study answers the research question 2 posed in Section 1.2.2: "How can memory strategies be used to tackle class imbalanced data streams with concept drifts? When and why are they helpful?"

CDCMS.CIL was evaluated through three experimental studies. In the first study, the raw version of CDCMS.CIL, which does not employ any resampling method, is compared with its predecessor CDCMS. Through experiments with artificial data streams, the proposed memory strategies have been shown to be effective in identifying recurring concepts and recovering relevant past knowledge to deal with concept drift in class imbalanced data streams (RQ2.1). However, the effectiveness of recovering predictive performance from the drift also depends on how well did the base learner learn from their corresponding concept. Specifically, the recovered base learners can only be helpful when they learnt from their corresponding concept without biasing towards either class. In other words, a resampling method is needed to enable CDCMS.CIL fully supports class imbalanced learning (RQ2.2).

With this in mind, the second study was performed to evaluate the impact of the resampling method to CDCMS.CIL. Through experiments with artificial and real-world data streams, it has been shown that CDCMS.CIL works with oversampling the best in most cases. The main reason is that oversampling helps to reduce the learning bias of the base learners without ignoring any potentially useful knowledge in the majority class (RQ2.2). However, CDCMS.CIL could, sometimes, suffer from early drift detection, especially in dealing with gradual drifts. In this case, CDCMS.CIL would recover past models that are relevant to the

pre-drift concept, as there are not enough examples from post-drift concept to determine which past knowledge to recover (RQ2.2).

In the last study, CDCMS.CIL was compared with five approaches from the literature, as well as its predecessor, CDCMS. In this study, CDCMS.CIL was equipped with oversampling as the strategy to deal with class imbalance. Experiments with artificial and real-world data streams shows that CDCMS.CIL performed similar or better than other compared approaches in most cases. In particular, CDCMS.CIL performed better in data streams presenting abrupt and abrupt recurrent drifts because no intermediate concept could form, thus CDCMS.CIL managed to recover relevant past knowledge to handle the drift. This validates the ability of CDCMS.CIL in deal with concept drift in class imbalanced data streams (RQ2.3).

Potential future work includes to improve the robustness of CDCMS.CIL to early drift detections. Investigation on the impact of other advanced resampling methods or class imbalance adaptable base learner to the predictive performance of CDCMS.CIL could also be done. Besides, CDCMS.CIL also inherits the issue from its predecessor, CDCMS, which the clustering in the model space strategy relies on a set of potentially high dimensional "clustering examples". This could weaken its ability in identifying relevant past knowledge. Therefore, alternatives could be devised in the future to overcome this weakness.

Chapter Six

Synthetic Minority Oversampling for Learning Drifting Class Imbalanced Data Streams

> *RQ3: How to produce synthetic examples that enable learning the decision boundaries better in class imbalanced stream learning with concept drifts? When and why are they helpful?*

Chapters 4 and 5 showed that using diversity to create a memory-based framework can help to exploit relevant past knowledge to handle multiple types of drift, not only in class balanced data streams but also class imbalanced data streams. However, relevant past knowledge might not always exist in the system's memory For examples, at the beginning of the data stream (when there is not yet enough data to obtain knowledge from), when the concept drift leads to a novel concept (as there is not enough data to obtain knowledge about a new concept right after the drift), and when relevant past knowledge has been deleted by the memory management strategy. In these situations, the learning system is not able to exploit relevant past knowledge. Thus, it struggles to learn the current underlying concept,

especially the minority class when the data stream is severely class imbalanced.

Oversampling and undersampling are two major categories of resampling method to handle class imbalance in the literature. As discussed in Chapter 5, undersampling might be preferable when the number of minority class is extremely low but oversampling is more preferable in most cases. OOB and UOB were proposed by S. Wang, Minku, and Yao (2015) which incorporate the above resampling methods into Online Bagging (Oza, 2005) to handle class imbalance. B. Wang and Pineau (2016) converts the approach of UnderOverBagging, which was proposed by S. Wang and Yao (2009) for offline learning, to learn from data stream. The converted approach is called OnlineUnderOverBagging (B. Wang and Pineau, 2016). The idea behind is to take the advantages of both resampling methods and further boost the diversity in the ensemble to handle class imbalance. However, this approach inherits the problem of its offline version, which takes a constant resampling rate as one of its hyper-parameters. Thus, it cannot adapt to data streams with changing class imbalance ratio.

Meanwhile, a special category of oversampling methods is very popular in the literature of offline machine learning. This category stems from an oversampling method called SMOTE (N. V. Chawla et al., 2002). SMOTE creates synthetic minority class examples for oversampling. It interpolates a point along the line between a randomly selected training example and one of its k-Nearest Neighbour. This method enriches the number of examples in the decision areas of the minority class, thus, achieves oversampling with less overfitting. In the same work by S. Wang and Yao (2009), they also proposed another version of UnderOverBagging by replacing the simple oversampling method by SMOTE. This approach is called SMOTEBagging. It is then converted by B. Wang and Pineau (2016) to be OnlineSMOTEBagging which is an online learning approach. Nevertheless, applying SMOTE directly to data stream learning may not be an ideal strategy because SMOTE requires to save all the examples in the memory. This will not be feasible when learning

from a data stream which has a potentially infinite sequence of examples. Similar to OnlineUnderOverBagging, OnlineSMOTEBagging also takes a constant resampling rate as one of their hyper-parameters, which means cannot adapt to data streams with changing class imbalance ratio.

A most recent approach proposed by Song, Minku, and Yao (2018) to handle class imbalanced data streams in Software Defect Prediction is also quite inspiring. This approach creates synthetic examples for oversampling by adding Gaussian noise to the most recent minority class example. This method is very efficient in terms of memory consumption because it does not require to store any past examples in the memory. Yet, as it always creates synthetic examples by adding Gaussian noise to the most recent minority class example, this means it always strengthens in learning the most recent minority class area, potentially overemphasising it. Also, the hyper-parameters for adding Gaussian noise are not adaptable. These designs may not be always ideal if there are a lot of noisy examples in the data stream.

On top of the above weaknesses, the above approaches from the literature are not able to adapt to concept drift, except OOB and UOB can handle drifts affecting $P(Y)$ only. To overcome these weaknesses, this chapter proposes a novel approach called Synthetic Minority Oversampling based on Stream Clustering (SMOClust) , which aims to create synthetic minority class examples for oversampling to learn drifting class imbalanced data streams. Together with the analysis to validate SMOClust, this chapter provides an answer to the RQ3, posed in Section 1.2.3: *"How to produce synthetic examples that enable learning the decision boundaries better in class imbalanced stream learning with concept drifts? When and why are they helpful?"* The following sub research questions are derived from RQ3 to guide the investigation done in this chapter, especially the analysis presented in Section 6.3:

- RQ3.1: How to produce minority class synthetic examples for oversampling that could explore the decision areas of the minority class while being able to adapt to concept

drift?

- RQ3.2: How does the proposed approach perform compared to existing approaches in different types of concept drift affecting the minority class? For which types does it perform the best and worst? Why?
- RQ3.3: How does the proposed approach perform compared to existing approaches in different real-world data streams?

This chapter focuses on concept drifts which affect the posterior probabilities of classes $P(Y|X)$ and / or affect the prior probabilities of the classes $P(Y)$, meaning that the class imbalance ratio of the data stream could change anytime.

To answer RQ3.1, a novel approach called Synthetic Minority Oversampling based on Stream Clustering (SMOClust) is proposed. This approach strategically selects the recent dense area of the minority class to create synthetic examples for oversampling. If this area does not exist, it adopt a strategy similar to Song, Minku, and Yao (2018), which create synthetic examples for oversampling by adding Gaussian noise to the most recent seen minority class examples. With these strategies, SMOClust can explore the decision areas of the minority class and consolidate its knowledge in the most recent minority class area. SMOClust also employs a drift detector to deal with concept drift explicitly. Meanwhile, it also requires the stream clustering method to be able to adapt to concept drift, such that the micro-clusters can reflect the current decision areas of the minority class.

SMOClust is evaluated and compared against nine existing approaches on artificial data streams concerning different types of concept drifts difficulties, answering RQ3.2. The experiment results shows that SMOClust performed particularly well in severely class imbalanced data streams with high proportions of safe and borderline minority class examples. It can also handle concept drifts of different movements of minority class clusters better than

other existing approaches. However, the experiment results also show that SMOClust is prone to handle rare and outlier minority class examples. To further analyse the characteristics of SMOClust, a few experiments with representative artificial data streams with lower dimensionality were performed. The learnt decision areas of SMOClust and the compared approaches at the time steps right before and after the drift, and at the end of the data stream were visualised and compared.

In additional to experiments with artificial data streams, we also performed experiments with different real-world data streams, answering RQ3.3. The experiment results present similar findings as obtained from the experiments with artificial data streams, further confirming the characteristics of SMOClust.

This chapter is organised as follows. Section 6.1 details the design of SMOClust. Section 6.2 presents the set up of the experiments to validate SMOClust and answer the sub research questions posed above. Section 6.4 summarises and discusses the finding in this chapter and the potential future works.

6.1 Description of Synthetic Minority Oversampling based on Stream Clustering (SMOClust)

This section details the design of the proposed approach – SMOClust, answering RQ3.1. The main idea of SMOClust is to exploit the power of stream clustering method to compress the information it has seen from each class. Most stream clustering methods represent these information in the form of micro-clusters. Based on micro-clusters of past examples, SMOClust creates synthetic minority class examples for oversampling. This method has the advantage of recording all the information of both classes without explicitly storing all the

examples to create synthetic examples for oversampling.

SMOClust is an explicit concept drift adaptation approach. It employs a concept drift detector to capture any potential drift by monitoring the predictive performance of the base learner. Upon drift detection, the base learner will always be replaced with a new one and SMOClust does not store any past base learner into its memory. As it was discussed in Chapters 4 and 5, this strategy is not ideal. However, this chapter focuses on investigating how the novel micro-cluster based synthetic minority oversampling strategy helps to learn drifting class imbalanced data streams. So, it is intended to keep other components of SMOClust in simple, such that they will not hinder us in analysing the characteristics of the proposed strategy. In the following paragraphs, the details of SMOClust is described and explained. The high-level pseudo-code of SMOClust is presented in Algorithm 10.

SMOClust has a base learner b to learn the data stream and make predictions to new examples. This base learner could be any single learner, such as Hoeffding Tree (Domingos and Hulten, 2000), or an ensemble learner, such as Online Bagging (Oza, 2005). To make predictions on new examples, it always delegates it to the current base learner b. SMOClust does not have a memory to store past models. Instead, it has stream clustering methods $SC[]$ to manage sets of micro-clusters representing the abstraction of past knowledge, one for each class of the problem (line 1, Algorithm 10). Any stream clustering methods can be used, for instance, CluStream (Aggarwal et al., 2003) and ClusTree (Kranen et al., 2011) were considered in hyper-parameter tuning of SMOClust in this study. These micro-clusters are used to create synthetic minority class examples for oversampling, which are explained in Section 6.1.1.

The most recent example s_t will be first used for concept drift detection (line 4, Algorithm 10). This concept drift detection method can be arbitrary, such as DDM (Gama, Medas, et al., 2004), DDM-OCI (S. Wang, Minku, Ghezzi, et al., 2013), PAUC (Brzezinski

Algorithm 10 Synthetic Minority Oversampling based on Stream Clustering - SMOClust

Hyper-parameters: Base Learner(B), Stream Clustering Method(sc), Class Size Fading Factor(θ), Gaussian Noise Variance(v), Categorical Change Probability(P_c), k-Nearest neighbour(k), Data Stream(S)

Variables: Base Learner(b), Stream Clustering Methods array($SC[]$)

$SC[] \leftarrow createStreamClusteringMethods(sc, 2)$ ▷ "2" refers to the number of SC to create, which corresponds to the number of classes of the problem.
for $s_t \in S$ **do**
  $drift_level \leftarrow DriftDetection(B, s_t)$
  if $drift_level == DRIFT$ **then**
    $B \leftarrow createNewBaseLearner(b); B.resetClassSize()$
  end if
  $B.trainOnInstance(s_t); B.updateClassSize(s_t, \theta)$
  $last_inst[s_t.classValue()] \leftarrow s_t$ ▷ Store the last seen example of each class
  $c_{maj} \leftarrow getMajorityClass(); c_{min} \leftarrow getMinorityClass()$
  while $(B.rawClassSize(c_{min}) < B.rawClassSize(c_{maj}))$ **and** $((SC[0].isReady()$ **and** $SC[1].isReady())$ **or** $last_inst[c_{min}] \neq NULL)$ **do**
    if $SC[0].isReady()$ **and** $SC[1].isReady()$ **then**
      $mCluster_{anchor} \leftarrow weightedRandomDrawByAvgTimeStamp(SC[c_{min}])$
      if $mCluster_{anchor}.surroundedBySameClass(SC)$ **then**
        $synthInst^{Bin} \leftarrow genSynthInstBykNN(SC[c_{min}], mCluster_{anchor}, c_{min}, k)$ ▷ Alg. 11
      else
        $synthInst^{Bin} \leftarrow genSynthInstByGauSampling(mCluster_{anchor})$
      end if
      $synthInst \leftarrow binaryToNominal(synthInst^{Bin}.copy())$
      $SC[c_{min}].trainOnInstance(synthInst^{Bin}.deleteClassAttribute())$
      $B.trainOnInstance(synthInst); B.updateClassSize(synthInst, \theta)$
    else ▷ **if** $last_inst[c_{min}] \neq NULL$
      $synthInst \leftarrow addGaussianNoiseToInstance(last_inst[c_{min}], v, P_c)$
      $synthInst^{Bin} \leftarrow nominalToBinary(synthInst.copy())$
      $SC[c_{min}].trainOnInstance(synthInst^{Bin}.deleteClassAttribute())$

25: $B.trainOnInstance(synthInst); B.updateClassSize(synthInst, \theta)$
26: **end if**
27: **end while**
28: $s_t^{Bin,noClass} \leftarrow nominalToBinary(s_t.copy())$
29: $SC[s_t.classValue()].trainOnInstance(s_t^{Bin,noClass}.deleteClassAttribute())$
30: **end for**

and Stefanowski, 2017), ADWIN (Bifet and Gavaldà, 2007) etc. When a concept drift detection is given, SMOClust will only reset the base learner b and the time decay class sizes but not the stream clustering methods $SC[]$ as they are already prepared to deal with concept drifts (line 5, Algorithm 10). That said, after concept drift detection, the stream clustering methods will still retain some knowledge belonging to the concept before the drift detection. This has two advantages: 1) knowledge of the pre-drift concept could be potentially helpful in learning the post-drift concept, especially when the drift has low severity (Minku and Yao, 2012) 2) to be robust to false positive drift detection, SMOClust can then exploit the knowledge stored in the micro-clusters to train the new base learner.

After that, SMOClust uses the most recent example s_t to train the base learner b and to update the time decay class sizes (line 7, Algorithm 10). These time decay class sizes are used to estimate the current minority class and to determine the oversampling rate. The time decay function to estimate the normalised class size of class c_m at current time step t follows the one used by S. Wang, Minku, and Yao (2015):

$$classSize(c_m)^{(t)} = \begin{cases} \frac{1}{|M|}, & \text{if } t = f \\ \frac{[c_{s_t}=c_m]+\theta \times classSize(c_m)^{(t-1)} \times (t-f)}{t-f+1}, & \text{otherwise} \end{cases} \tag{6.1}$$

where $m \in M$ and $M = \{0, 1\}$, considering binary classification tasks and θ $(0 < \theta < 1)$ is a predefined time decay factor. c_{s_t} is the true class of s_t. Thus, $[c_{s_t} = c_m] = 1$ if the true class of s_t is c_m, otherwise 0. f is the first time step used in the calculation. Note that unlike

OOB and UOB (S. Wang, Minku, and Yao, 2015) which estimate the class sizes of the data stream at the current time step t using the time decay function, SMOClust estimates the time decay class sizes seen by the base learner. The reason behind this difference will be discussed together with the mechanism of training the base learner with synthetic example.

After updating the time decay class sizes, SMOClust stores the most recent examples from each class (line 8, Algorithm 10). The one that belongs to the minority class is used to create synthetic examples for oversampling by adding Gaussian noise when the micro-clusters are not ready to use. After storing the most recent examples from each class, SMOClust checks if the class size of the minority class is smaller than that of the majority class (from the base learner's perspective) (line 10, Algorithm 10). If this condition is satisfied, SMOClust will also check if the stream clustering methods are ready for retrieving the micro-clustering results. If this condition is also satisfied, SMOClust will generate synthetic minority class examples for oversampling using the micro-clusters (line 13-17, Algorithm 10), which is be detailed in Section 6.1.1.

If the condition about the micro-clustering results is not satisfied but the system has seen and stored the most recent "real" example of the minority class (denoted as $s_{last_minority}$), SMOClust will generate synthetic minority class by adding Gaussian noise to $s_{last_minority}$ for oversampling (line 22, Algorithm 10). This strategy follows the strategy proposed by Song, Minku, and Yao (2018), except SMOClust treats ordinal attributes as categorical attributes because of the limitation in MOA (Bifet, Holmes, et al., 2010). In short, SMOClust adds Gaussian noise to a numeric attribute of $s_{last_minority}$ by applying a Gaussian distribution centred at the current value of $s_{last_minority}$ at that numeric attribute with a predefined variance v. For nominal attributes (including categorical and ordinal attributes), SMOClust adds Gaussian noise by changing it to any other possible value with a predefined probability P_c.

No matter the synthetic minority class example is generated based on micro-clusters or Gaussian noise, SMOClust will use it to train the base learner and the corresponding stream clustering method, and to update the class sizes immediately (line 19-20, 24-25, Algorithm 10). The reason for updating the class sizes with synthetic examples is that SMOClust estimates the class sizes seen by the base learner rather than the class sizes of the data stream. In this way, when the base learner has seen more examples from one class than the other class, SMOClust will immediate create synthetic minority class examples to train the base learner. This strategy is to avoid that the base learner will have nothing from the minority class to train for a long period when the data stream is extremely class imbalanced.

In the case of none of the above conditions being satisfied, SMOClust will not perform any oversampling because this means either oversampling is not needed or there is no information about the minority class for SMOClust to generate synthetic examples (Conditions after the first **and** operator in line 10, Algorithm 10 are not satisfied). Last but not least, the most recent example s_t will be copied and converted to the format that is suitable to train the stream clustering method, corresponding to the class value of s_t (line 28-29, Algorithm 10).

6.1.1 Generating a synthetic minority class example for oversampling using the micro-clusters

This section presents the overview of generating a synthetic minority class example for oversampling based on micro-clusters. The general idea is to create synthetic minority class examples in one of the dense areas of the minority class. In this way, we can consolidate the knowledge learnt in the minority class areas seen so far without greatly affected by noise. Meanwhile, if this dense area does not exist, SMOClust will pick one of the past minority class area to explore the decision boundary around it.

Algorithm 11 presents the pseudo-code of this method. The details of generating a synthetic minority class example using micro-clusters can be described as follows.

Algorithm 11 Generate Synthetic Instance with k-NN Micro-Clusters

1: **function** GENSYNTHINSTBYKNN($SC[c_{min}]$, $mCluster_{anchor}$, c_{min}, k)
2: $kNNmClusters \leftarrow SC[c_{min}].getkNNmClusters(mCluster_{anchor})$
3: $sphere_cluster \leftarrow createSphereCluster(mCluster_{anchor}, kNNmClusters)$
4: $synthInst \leftarrow sphere_cluster.sample_around_target(anchor_mCluster.getCentre())$ ▷ Alg. 13
5: $synthInst.setClassValue(s_t.classValue())$
6: **return** $synthInst$
7: **end function**

First of all, SMOClust takes one of the micro-clusters of the minority class as an anchor (denoted as $mc_{anchor}^{minority}$) randomly (line 12, Algorithm 10). In particular, micro-clusters that are created recently or are updated frequently and recently have higher chance to be chosen as this anchor. After that, SMOClust checks if $mc_{anchor}^{minority}$ is surrounded by the micro-clusters from the same class (line 13, Algorithm 10). If this condition is satisfied, SMOClust can consider this area is dense enough to create synthetic minority examples for oversampling. It will then make a copy of $mc_{anchor}^{minority}$ and then combine it with its k-Nearest micro-clusters (based on hull distance) in class c_{min} to form a temporary micro-cluster mc_{temp} (line 2-3, Algorithm 11). We denote this set of k-Nearest micro-clusters as $MC^{kNN,minority}$, thus, $|MC^{kNN,minority}| = k$ and each k-Nearest micro-clusters is denoted as $mc_i^{kNN,minority} \in MC^{kNN,minority}$. The details of how to combine a set of micro-clusters into one are presented in Algorithm 12.

To combine a set of micro-clusters into one, we first need to calculate the new centre c_{new} of the resulting micro-cluster mc_{temp}. This can be achieved by getting the weighted average among the centres of the original set of micro-clusters, dimensionwisely (line 9-15, Algorithm 12). After that, we set the radius r_{new} of the resulting micro-cluster mc_{temp} to as the distance between the new centre to the farthest hull (boundary) among all the original

Algorithm 12 Combining a set of micro-clusters into one

```
1: function COMBINE(mClusters[])
2:     dimensions ← mClusters[0].numOfDimensionss()
3:     for i ∈ range(0..mClusters.length) do
4:         all_centres[i] ← mClusters[i].getCentre()
5:         all_weights[i] ← mClusters[i].getWeight()
6:         all_radius[i] ← mClusters[i].getRadius()
7:     end for
8:     newCentre ← createArrayWithSize(dimensions)
9:     for i ∈ range(0..dimenstions) do          ▷ Weighted sum of centres, by dimension.
10:        result_by_dim ← 0
11:        for j ∈ range(0..mClusters.length) do
12:            result_by_dim ← result_by_dim + all_centres[j][i] * all_weights[j]
13:        end for
14:        newCentre[i] = result_by_dim/sum(all_weights)
15:    end for
16:    r_n ← createArrayWithSize(all_radius.length)
17:    for i ∈ range(0..all_radius.length) do    ▷ Find the distance from newCentre to farthest hull.
18:        distance_to_newCentre ← euclidean_distance(all_centres[i], newCentre)
19:        r_n ← all_radius.length[i] + abs(distance_to_newCentre)
20:    end for
21:    r_n ← r_n.sort(ascending); new_radius ← r_n[r_n.length − 1]
22:    return createMicroCluster(newCentre, newRadius)
23: end function
```

micro-clusters (line 17-20, Algorithm 12). Figure 6.1 illustrates an example of combining $mc_{anchor}^{minority}$ with its 3-nearest neighbours into one micro-cluster.

A synthetic minority class example will then be generated by sampling from this resulting micro-cluster with the highest chance near the centre of $mc_{anchor}^{minority}$ (line 4, Algorithm 11). Figure 6.2 illustrates an example of sampling from a synthetic minority class example from mc_{temp}.

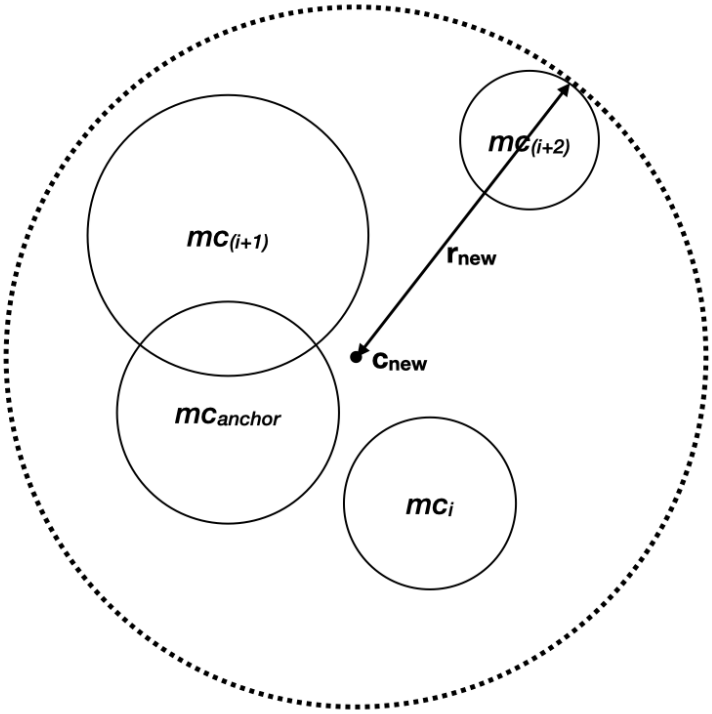


Figure 6.1: Illustration of Combining the $mc_{anchor}^{minority}$ with its 3-nearest neighbours into one micro-cluster

In Figure 6.2, the green circles are the micro-clusters belonging to the minority class while the blue circles are the micro-clusters belonging to the majority class.[1] The black circle line represents mc_{temp} and the red dashed lines are the contour of the probability density function to sampling a point. The closer to the centre of $mc_{anchor}^{minority}$, the higher the probability.

The reason why we sample a new synthetic minority class example close to $mc_{anchor}^{minority}$ is that, this mc_{temp} could possibly overlap with the micro-cluster from the other class. If we just sampling from mc_{temp} randomly or by a multivariate Gaussian distribution with mean at c_{new}, we will have high chance to sample a point that is close to the region or the majority

[1]Note that, the size and the number of micro-clusters in Figure 6.2 do not necessarily reflect the number of examples in each class. This figure just focuses on a particular region in the feature for explanation purposes.

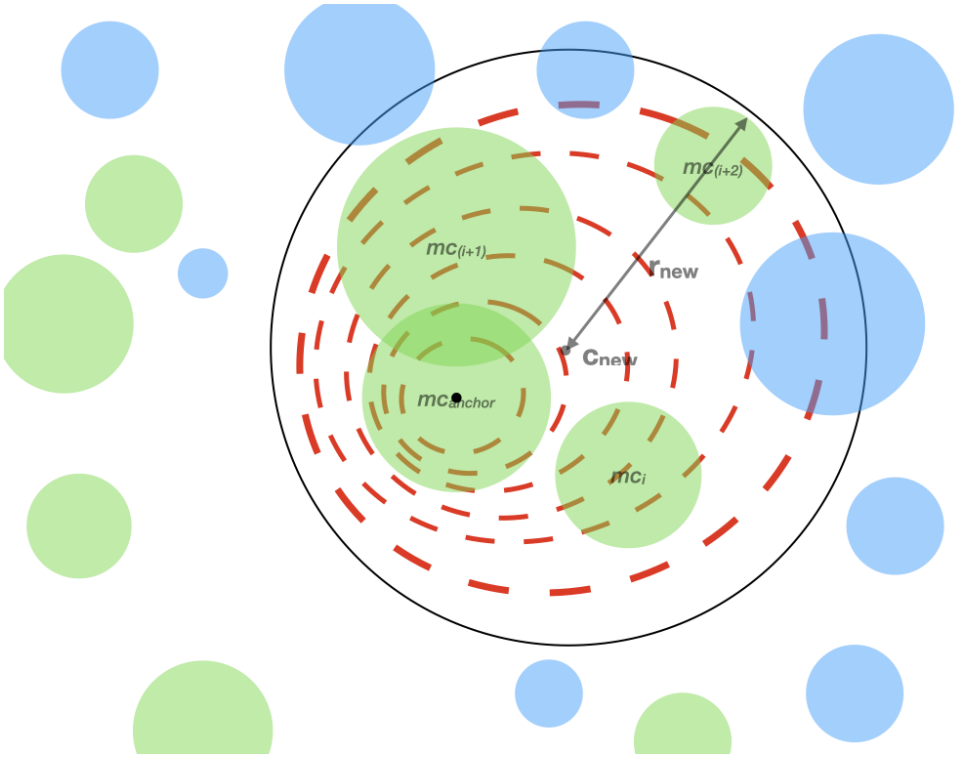


Figure 6.2: Illustration of Sampling a Synthetic Minority Class Example from mc_{temp}

class. Therefore, sampling points as synthetic minority class examples from mc_{temp} but close to the centre of $mc_{anchor}^{minority}$ can reduce the risk of generating noisy examples while maintaining the ability of exploring this dense region of minority class.

Although Figure 6.2 only illustrates an example in two dimensional feature space, this idea can be applied to any multi-dimensional space. This sampling strategy is further detailed in Section 6.1.2.

In the case that $mc_{anchor}^{minority}$ is not surrounded by the micro-clusters belonging to the same class, SMOClust will generate a synthetic minority class example by performing multivariate Gaussian sampling inside $mc_{anchor}^{minority}$ (line 16, Algorithm 10). For example, this will be the case when when $mc_{(i+2)}$ (top right green circle in Figure 6.2) is chosen to be the $mc_{anchor}^{minority}$. The the mean of the multivariate Gaussian distribution is the centre of $mc_{anchor}^{minority}$ and the standard deviation is set as a third of the radius of $mc_{anchor}^{minority}$ $(radius/3)$. In other

words, the boundary of $mc_{anchor}^{minority}$ is set at three units standard deviations (or standard score = 3) from the centre. Therefore, we have 99.9% of chance to sample a point within $mc_{anchor}^{minority}$. Gaussian distribution was chosen rather than uniform distribution in sampling $mc_{anchor}^{minority}$ because $mc_{anchor}^{minority}$ could partly overlap with the majority class region. Therefore, sampling a new point as synthetic minority class example close to the centre of $mc_{anchor}^{minority}$ is a safe strategy.

The reason for sampling from multivariate distribution rather than univariate distribution in each dimension is explained in Section 6.1.3.

6.1.2 Sampling from a Micro-cluster with the Highest Probability at a Designated Location

This section present the strategy to sampling points from the temporary micro-cluster mc_{temp} which is formed by combining $mc_{anchor}^{minority}$ and $mc_i^{kNN,minority} \in MC^{kNN,minority}$ with the highest probability at the centre of $mc_{anchor}^{minority}$. The general idea is to sample random points that are inside mc_{temp} and these points are likely to be close to the centre of $mc_{anchor}^{minority}$. The pseudo-code of this sampling strategy is presented in Algorithm 13. Figure 6.3 illustrates the steps of this sampling strategy and it can be explained as follows.

Let us first denote the micro-cluster mc_{temp} as HS_β which is a hyper-sphere with radius r and and centred at $\beta = (\beta_1, \beta_2, \beta_3, ..., \beta_n)$, where n is the number of dimensions of the input space of the problem, the equation of this hyper-sphere is:

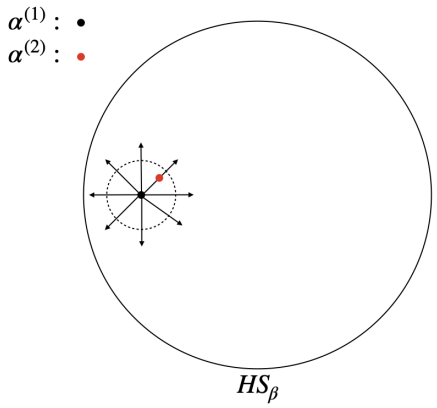
$$\sum_{i=0}^{n}(x_i - \beta_i)^2 = r^2 \tag{6.2}$$

Let us also denote the centre of $mc_{anchor}^{minority}$ to as $\alpha^{(1)} = (\alpha_1^{(1)}, \alpha_2^{(1)}, \alpha_3^{(1)}, ..., \alpha_n^{(1)})$ (the black dot

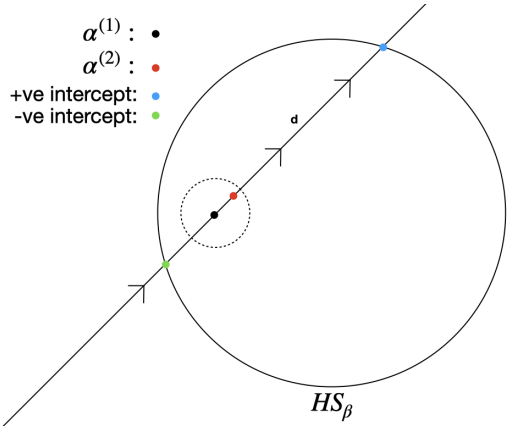
Algorithm 13 Sampling from a Hyper-Sphere by Skewed Gaussian with the Maximum of the Probability Density Function at a Designated Location

1: **function** SAMPLE_AROUND_TARGET($\alpha^{(1)}$, *sphere_cluster*)
2: $\beta \leftarrow sphere_cluster.getCentre()$
3: $r \leftarrow sphere_cluster.getRadius()$
4: $dimensions \leftarrow \beta.numOfDimensions()$
5: $\delta \leftarrow createArrayWithSize(dimensions)$
6: $\gamma \leftarrow createArrayWithSize(dimensions)$
7: $\alpha^{(2)} \leftarrow sample_random_hypersphere(\alpha^{(1)}, 1)$ ▷ Alg. 14
8: $A \leftarrow 0; B \leftarrow 0; C \leftarrow 0$
9: **for** $i \in range(0..dimensions)$ **do**
10: $\delta[i] \leftarrow \alpha^{(2)}[i] - \alpha^{(1)}[i]$
11: $\gamma[i] \leftarrow \beta[i] - \alpha^{(1)}[i]$
12: $A \leftarrow A + (\delta[i] * \delta[i])$ ▷ $A = \sum_{i=0}^{n} \delta_i^2$
13: $B \leftarrow B + (\delta[i] * \gamma[i])$ ▷ $\sum_{i=0}^{n} \delta_i \gamma_i$
14: $C \leftarrow C + (\gamma[i] * \gamma[i])$ ▷ $\sum_{i=0}^{n} \gamma_i^2$
15: **end for**
16: $B \leftarrow B * -2$ ▷ $B = -2(\sum_{i=0}^{n} \delta_i \gamma_i)$
17: $C \leftarrow C - (r * r)$ ▷ $C = (\sum_{i=0}^{n} \gamma_i^2) - r^2$
18: **return** $(-B + sqrt(B * B - 4 * A * C))/(2 * A)$ ▷ $\frac{-B+\sqrt{B^2-4AC}}{2A}$
19: **end function**

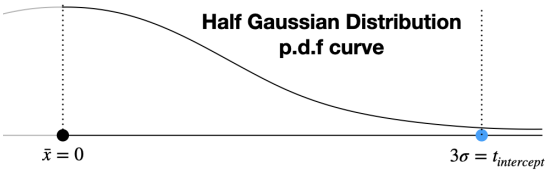
in Figure 6.3(a)), which should always be inside HS_β. First of all, we need to pick a random direction from $\alpha^{(1)}$ (Figure 6.3(a)). To achieve this, we need to randomly and uniformly pick a point in a unit hyper-sphere to indicate this direction, using the Muller (1959)'s method (Algorithm 14). The detail of the Muller (1959)'s method is explained in Section 6.1.3. We then denote this point to as $\alpha^{(2)} = (\alpha_1^{(2)}, \alpha_2^{(2)}, \alpha_3^{(2)}, ..., \alpha_n^{(2)})$ (the red dot in Figure 6.3(a)) (line 7, Algorithm 13). Points $\alpha^{(1)}$ and $\alpha^{(2)}$ can then form an n-dimensional infinite long straight line (the line d in Figure 6.3(b)), whose parameterised equation is:



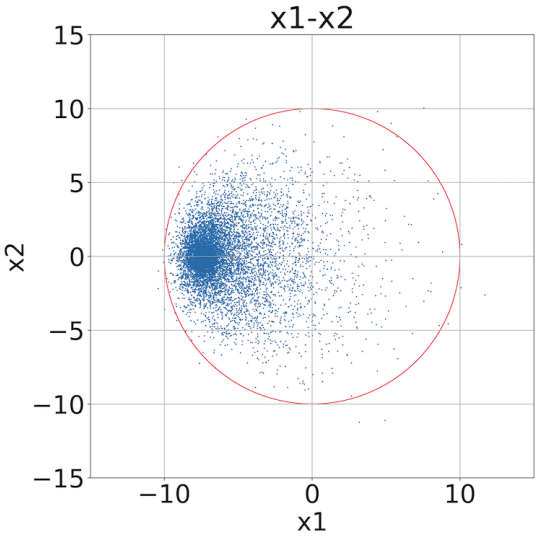
(a) Step 1



(b) Step 2



(c) Step 3



(d) Two Dimensional Example of the Sampling Result

Figure 6.3: Illustration of Sampling from mc_{temp}

$$x_i = \alpha_i^{(1)} + t(\alpha_i^{(2)} - \alpha_i^{(1)}) \tag{6.3}$$

where t is a scalar and $(\alpha_i^{(2)} - \alpha_i^{(1)})$ is the direction vector. As this line is infinite long, it will eventually intercept the hull of HS_β (the blue and green dots in Figure 6.3(b)).[2] To find the

[2]The idea of finding the intercepts of an n-dimensional infinite long straight line passing through a hyper-

exact locations of these intercepts, we can substitute Equation 6.3 into Equation 6.2:

$$\sum_{i=0}^{n}((\alpha_i^{(2)} - \alpha_i^{(1)})t + (\alpha_i^{(1)} - \beta_i))^2 = r^2 \tag{6.4}$$

Let us denote $\delta_i = \alpha_i^{(2)} - \alpha_i^{(1)}$ and $\gamma_i = \beta_i - \alpha_i^{(1)}$ (line 10 and 11, Algorithm 13), then Equation 6.4 becomes:

$$\sum_{i=0}^{n}(\delta_i t - \gamma_i)^2 = r^2$$

$$(\sum_{i=0}^{n}\delta_i^2)t^2 - 2(\sum_{i=0}^{n}\delta_i\gamma_i)t + (\sum_{i=0}^{n}\gamma_i^2) - r^2 = 0 \tag{6.5}$$

Let us denote $A = \sum_{i=0}^{n}\delta_i^2$ (line 12, Algorithm 13), $B = -2(\sum_{i=0}^{n}\delta_i\gamma_i)$ (line 13 and 16, Algorithm 13) and $C = (\sum_{i=0}^{n}\gamma_i^2) - r^2$ (line 14 and 17, Algorithm 13) to solve Equation 6.5 based on Bhaskara's equation:

$$t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

Here, we just take the positive root of t because it "follows" the direction vector, while the negative root "oppositely follows" the direction vector (the direction is denoted by the arrows on line d in Figure 6.3(b)). i.e.

$$t_{intercept} = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \tag{6.6}$$

sphere is inspired by the discussion on https://math.stackexchange.com/questions/151064/calculating-line-intersection-with-hypersphere-surface-in-mathbbrn?rq=1

Substituting $t_{intercept}$ into Equation 6.3 will obtain the intercept of the line and the hyper-sphere, following the direction vector (the blue dot in Figure 6.3(b)). Thus, to sample points within the HS_β, we can simply sample a scalar t_{sample} between 0 and $t_{intercept}$ (Figure 6.3(c)) and substitute it into Equation 6.3 to obtain the sampled point. As we want to sample this point with the highest chance at the target point $\alpha^{(1)}$, we can sample t_{sample} using Gaussian distribution with the mean = 0 and standard deviation = $\frac{t_{intercept}}{3}$. i.e.

$$g \sim N(0, (\frac{t_{intercept}}{3})^2)$$

$$t_{sample} = |g|$$

At last, we substitute t_{sample} into Equation 6.3 to obtain the sample point.

The reason for setting the standard deviation to be $\frac{t_{intercept}}{3}$ is that we want the sampled point to be within the micro-cluster. Yet, the probability density function of the Gaussian distribution has no bounds. Thus, we set the $t_{intercept}$ at 3 standard score (z-score = 3), such that 99.9% area under the probability density function curve of the Gaussian distribution is between $-t_{intercept}$ and $+t_{intercept}$. Also, we want t_{sample} to "follow" the direction vector (i.e. we only interested in line segment between the black and the blue dots on d in Figure 6.3(b)) ; thus, we only accept the positive value of t_{sample}.

Figure 6.3(d) presents a two dimensional example of using the aforementioned strategy to sample points in a hyper-sphere centred at (0,0) with a radius of 10. The points have the highest probability to be sampled at (-7,0)

6.1.3 Uniform Sampling inside a Hyper-Sphere

This section presents the Muller (1959)'s method to sample a random point from a hyper-sphere uniformly. The sampling method presented in Section 6.1.2 uses this method to pick a random direction uniformly to start off its sampling procedure. Note that the simple strategy of dimensionwise uniform sampling cannot be used to pick this random direction uniformly because it samples from a hyper-cube where all the distances from the centre to the hull are different. This means that the direction with a long distance has a higher chance to be chosen. That is why we need to leverage Muller (1959)'s method to pick this random direction uniformly. In the following paragraphs, Muller (1959)'s method is explained in details.

Given an n-dimensional hyper-sphere hs_{target} centred at the c with radius r_{target}, we first need to select a direction to uniformly sample a point g on the surface of a smaller concentric n-dimensional hyper-sphere hs_{small} with radius r_{small} using Muller's method (Muller, 1959) and then move it by a random distance between g and c.[3] Algorithm 14 presents the pseudo-code of Muller's method.

Muller (1959)'s method first samples an n-dimensional vector of numbers from a Gaussian distribution with mean = 0 and variance = 1 (line 5, Algorithm 14). It then divides each number in the vector by the L_2-norm of the vector (line 9, Algorithm 14). This vector now represents a point g sampled uniformly on the surface of a unit hypersphere hs_{unit}, which is also a direction from the centre c. Next, we need to sample a distance D, which is the distance from centre c to the final random point along this direction.

Consider the volume ratio of hs_{small} to hs_{target} is $(\frac{r_{small}}{r_{target}})^n$. This is also the probability

[3]This method of uniformly sampling inside a hyper-sphere is based on Muller's method (Muller, 1959) and the discussion on https://math.stackexchange.com/questions/87230/picking-random-points-in-the-volume-of-sphere-with-uniform-probability.

Algorithm 14 Uniform Sampling inside a Hyper-Sphere - Muller's method (Muller, 1959)

function SAMPLE_RANDOM_HYPERSPHERE($centre$, $radius$)
  $dimensions \leftarrow centre.numOfDimensions()$
  $g \leftarrow createArrayWithSize(dimensions)$
  for $i \in range(0..dimensions)$ **do**
    $g[i] \leftarrow random.nextGaussian(0, 1)$ ▷ random.nextGaussian(mean, variance)
  end for
  $L_2_norm \leftarrow calculate_L_2_Norm(g)$
  for $i \in range(0..dimensions)$ **do**
    $g[i] \leftarrow g[i]/L_2_norm$ ▷ g now is a point on the surface of the hyper-sphere and a direction vector
  end for
  $u \leftarrow pow(random.nextDouble(), 1/dimensions)$ ▷ $u = x^{\frac{1}{dimensions}}$, where $x \sim U(0, 1)$
  $res \leftarrow createArrayWithSize(dimensions)$
  for $i \in range(0..dimensions)$ **do**
    $res[i] \leftarrow g[i] * (radius * u) + centre[i]$ ▷ $D = radius * u$
  end for
  return res
end function

of sampling a random point inside hs_{target} that would lies inside hs_{small}. In other words, this is also the probability of sampling a distance D that would smaller or equals to r_{small}. i.e.:

$$P(D \leq r) = (\frac{r_{small}}{r_{target}})^n \tag{6.7}$$

To sample the distance D uniformly, we have to find a function f that transforms the random variable x to D, where x is sampled from a uniform distribution between 0 and 1 ($U(0, 1)$). i.e. $x \sim U(0, 1)$ and, thus, we have:

$$D = f(x) \tag{6.8}$$

Substituting Equation 6.8 into Equation 6.7, we have

$$P(f(x) \leq r_{small}) = (\frac{r_{small}}{r_{target}})^n$$

$$P(x \leq f^{-1}(r_{small})) = (\frac{r_{small}}{r_{target}})^n$$

$$f^{-1}(r_{small})\frac{1}{1-0} = (\frac{r_{small}}{r_{target}})^n$$

$$f^{-1}(r_{small}) = (\frac{r_{small}}{r_{target}})^n$$

therefore,

$$D = f(x) = (r_{target})(x^{\frac{1}{n}})$$

Finally, we just need to multiply each dimension of the point g, which represents the direction, with this distance D, then dimensionwisely add the centre c to the resulting vector to obtain the sampled point (line 14, Algorithm 14).

6.2 Design of Experiment

This section presents the design of the experiments to evaluate SMOClust. We first compared the performance of SMOClust against five existing approaches from the literature on artificial data streams of different types of drifts. This is to investigate for which types of drift SMOClust will be advantageous and disadvantageous. SMOClust is then compared against the same set of existing approaches on real-world data streams to obtain a general idea of the its predictive performance in real-world applications. Massive Online Analysis (MOA) (Bifet, Holmes, et al., 2010) was chosen to be the platform to perform the experiments. Section 6.2.1 summarises the artificial and real-world data streams used in the experiments. Section 6.2.2 presents the detailed setup of the experiments, including the procedure of hyper-parameter tuning and the evaluation method used in the experiments.

6.2.1 Data streams

To evaluate which types of drift would SMOClust be advantageous and disadvantages, we adopted the artificial data stream generator proposed by Brzezinski, Minku, et al. (2021) because it can simulate concept drifts that affect different factors, such as the class imbalance ratio, movement of minority class sub-clusters, and the proportion of safe, borderline and rare minority class examples. Thus, the generated artificial data streams can help us to understand when and why SMOClust is advantageous or disadvantageous.

Table 6.1 presents a summary of artificial data streams used in the experiments. Each of them has five numerical input attributes $\{x_i \in (-1, 1)\}_{i=1}^{5}$ and a class label $y_i \in \{0, 1\}$. They all consist of 200k examples where concept drift happens gradually during 70k to 100k time steps. Also, we created thirty artificial data streams of each type with different random seeds. Each of the thirty artificial data streams is used evaluate the data stream learning approaches in a single run. The evaluation method is detailed in Section 6.2.2

Following the naming convention by Brzezinski, Minku, et al. (2021), when the artificial data stream has no drift or no modifier specified, it is: 1) class balanced, 2) composed of a single cluster representing class 1, uniformly surrounded by the examples of class 0, and 3) examples only appear in safe regions. When the data stream is class imbalanced, class 1 is the minority class while class 0 is the majority class.

As shown Table 6.1, we considered four groups of drift from Brzezinski, Minku, et al. (2021)'s work in this study. The first group (Imbalance ratio drift) considers concept drift affecting the class imbalance ratio only. The second group (Single factor drift with static imbalance ratio) considers data streams with a static class imbalance ratio while the concept drift happens in the form of five factors, which are discussed in Section 1.1.2 and by Brzezinski, Minku, et al. (2021): splitting, moving, merging sub-clusters and decreasing

Table 6.1: Summary of Artificial Data Streams

Imbalance Ratio Drift	Single Factor Drift with Static Imbalance Ratio
StaticIm10_Im1, StaticIm1_Im10, Im1, StaticIm1_Im50	StaticIm{30/10/1}_ Split{3/7} / Move{3/7} / Merge{3/7} / Borderline{20/100} / Rare{20/100}
Double Factors Drift	**Complex Factor Drift**
Im1+Rare100, Im10+Rare60, Split5+Im10, Im1+Borderline100, Im10+Borderline20	StaticIm10_Split5+Im1+Rare100, StaticIm10_Split5+Im1+Borderline100, Split5+Im10+Borderline40+Rare40, Split5+Im10+Borderline80, Im10+Borderline20+Rare20

- All artificial data streams have 200k examples, where a single concept drift from 70k-th time step to 100k-th time step.
- "+" refers to the factors occurring simultaneously during the concept drift.
- StaticIm{N} refers to a static minority class ratio of N% throughout the entire stream.
- Im{N} refers to the minority class ratio of N% after the concept drift.
- Split{N}, Move{N}, Merge{N} refer to drifts which split, move and merge N clusters in the minority class respectively.
- Borderline{N}, Rare{N} refer to drifts changing N% of the minority class examples from appearing in a safe region of the clusters to being borderline region and rare cases respectively.

the ratio of safe examples while increasing the ratio of borderline or rare examples. In the third (Double factors drift) and the fourth (Complex factor drift) groups, we have chosen ten artificial data streams (five for each group) with concept drift affecting two factors and a group of factors, respectively. These artificial data streams were chosen evenly across the lists of data streams from Brzezinski, Minku, et al. (2021)'s work with double factors drift and complex factor drift in Brzezinski, Minku, et al. (2021)'s work respectively. These lists were sorted by the average performance of the compared data stream learning approaches in their work. Thus, picking data streams evenly from these list means that we are taking scenarios with different difficulty levels.

As the analysis which is presented in Section 6.3.1 shows that SMOClust performed well in severely imbalanced data streams, we performed additional experiments with the aforementioned single factor drift streams with more severe static class imbalance ratio to further evaluate SMOClust in extreme cases. These additional severely class imbalanced artificial data streams are summarised in Table 6.2. Note that, although the static imbalance ratio of 1% minority class examples was reused, another set of random seeds was used when performing these additional experiments.

Table 6.2: Summary of Single Factor Drift Artificial Data Streams with Severe Imbalance Ratio

Single Factor Drift with Severe Static Imbalance Ratio	
StaticIm{5/3/1/07/05/03}_	Split{3/7} / Move{3/7} / Merge{3/7} / Borderline{20/100} / Rare{20/100}

- All artificial data streams have 200k examples, where a single concept drift from 70k-th time step to 100k-th time step.
- StaticIm{N} refers to a static minority class ratio to be N% throughout the entire stream. StaticIm{$0N$} refers to a static minority class ratio to be $0.N$% throughout the entire stream.
- Split{N}, Move{N}, Merge{N} refer to drifts which split, move and merge N clusters in the minority class respectively.
- Borderline{N}, Rare{N} refer to drifts changing N% of the minority class examples from appearing in a safe region of the clusters to being borderline region and rare cases respectively.

Apart from experiments with artificial data streams, we also performed experiments with different real-world data streams to evaluate SMOClust in practical applications. These real-world data streams are summarised in Table 6.3 and their details are as follows.

Six real-world data streams have been used: Luxembourg (Žliobaitė, 2011), NOAA (Elwell and Polikar, 2011), Ozone (K. Zhang et al., 2006), PAKDD2009 (Theeramunkong et al., 2009), Forest Covertype (Covtype) (Blackard and Dean, 1999) and INSECTS (Souza

Table 6.3: Summary of Real-World Data Streams

Stream	#Examples (Pre)	#Examples (Actual)	#Nom. Attr.	#Num. Attr.	Imbalance Ratio (Pre)	Imbalance Ratio (Actual)
Luxembourg	190	1711	15	16	0.532:0.468	0.512:0.488
NOAA	1,815	16,344	0	8	0.698:0.303	0.685:0.315
Ozone	253	2,281	0	72	0.893:0.107	0.942:0.058
PAKDD2009	4999	44998	13	14	0.798:0.202	0.803:0.197
$\text{Covtype}_{(c_1=\{1\text{-}6\})}$	58,101	522,911	2	10	0.785:0.215	0.619:0.381
$\text{Covtype}_{(c_1=1)}$	58,101	522,911	2	10	0.595:0.405	0.524:0.476
$\text{Covtype}_{(c_1=2)}$	58,101	522,911	2	10	0.963:0.037	0.936:0.064
$\text{Covtype}_{(c_1=3)}$	58,101	522,911	2	10	0.963:0.037	0.999:0.001
$\text{Covtype}_{(c_1=4)}$	58,101	522,911	2	10	0.958:0.042	0.987:0.014
$\text{Covtype}_{(c_1=5)}$	58,101	522,911	2	10	0.963:0.037	0.971:0.029
$\text{Covtype}_{(c_1=6)}$	58,101	522,911	2	10	0.963:0.037	0.965:0.035
$\text{INSECTS}^{\text{inc.}}$	45,204	406,840	0	33	0.899:0.101	0.905:0.095
$\text{INSECTS}_{\text{abr.}}$	35,527	319,748	0	33	0.912:0.088	0.907:0.093
$\text{INSECTS}^{\text{inc.}}_{\text{grad.}}$	14,342	128,981	0	33	0.921:0.079	0.899:0.101
$\text{INSECTS}^{\text{inc.}}_{\text{abr. re.}}$	45,204	406,840	0	33	0.895:0.105	0.905:0.095
$\text{INSECTS}^{\text{inc.}}_{\text{re.}}$	45,204	406,840	0	33	0.895:0.105	0.905:0.095

- Total number of attributes = #Nominal attributes + #Numeric attributes + Class attribute.
- "Pre" refers to hyper-parameter tuning sets (i.e. the first 10% of the original data set). "Actual" refers to actual experiment sets (i.e. the remaining 90% of the original data set.
- $\text{Covtype}_{(c_1=x)}$: "c_1=x" refers to the class 1 is the class x in the original data set while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream. "$c_1=\{x_0 - x_n\}$" refers to the class 1 is the class x_0-x_n in the original data set combined while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream.
- For all INSECTS data streams, "ae-albopictus" is the class 1. "inc." refers to incremental, "abr." refers to abrupt, "grad" refers to gradual, and "re." refers to recurring.

et al., 2020). Their background are as follows.

The Luxembourg stream (Žliobaitė, 2011) was constructed from the European Social Survey from 2002 to 2007 and learning task of classify whether the internet usage is high or low based on a questionnaire. The potential concept drift is that the internet usage may change over time (Souza et al., 2020). The NOAA stream (Elwell and Polikar, 2011) contains weather records collected over five decades (1949-1999) at the Offutt Air Force Base in Bellevue, Nebraska. Daily weather measurements include a variety of features, such as temperature, pressure, wind speed, etc. Precipitation indicators and other weather-related events are also included. The learning task of NOAA stream is to predict whether the next day will rain or not, based on the weather measurements. The Ozone stream (K. Zhang et al., 2006) consists of air measurements collected from 1998 to 2004 and the learning task is to predict the ozone level eight hours ahead of time. The PAKDD2009 stream (Theeramunkong et al., 2009) is from a private label credit operation of a major Brazilian retail chain. The classification task is to decide whether a given credit card application should be approved, based on the applicant's financial status. This data set was first used in the data mining competition of the 13th Pacific-Asia Knowledge Discovery and Data Mining conference (PAKDD 2009) (Theeramunkong et al., 2009).

Forest Covertype (Covtype) stream (Blackard and Dean, 1999) contains the cartographic information about the forest cover type of 30 × 30-meter cells, which includes elevation, wilderness, slope, soil type etc. The actual forest cover type (the true label) for a given cell was determined from the US Forest Service (USFS) Region 2 Resource Information System (RIS) data. The learning task is to predict the forest cover type of a 30 × 30-meter cell based on the given cartographic information. This stream originally is a multi-classification problem, where it has seven forest cover types. To make it suitable for this study, it has been converted into seven binary classification streams. Each of them takes one of the forest cover types as one class while combining other forest cover types to be the other class. Therefore, there are seven Forest Covertype streams after the conversion, as shown on Table 6.3. They

allow us to evaluate how well a given data stream learning approach performs in learning each of the seven classes.

INSECTS streams (Souza et al., 2020) were constructed using a smart trap with optical sensor to collect the flying data of three different species of insects in a non-stationary environment for around three months. The temperature of the data collection environment was controlled to simulate concept drifts. The data collection were performed at the University of São Paulo, Brazil. These data streams consists of information extracted from the audio file of the sound produced by the insects, such as wing-beat frequency, harmonics positions etc. INSECTS streams originally have six classes: three species of mosquitoes with two genders. We converted them into binary classification tasks by combining classes belonging to species of ae-albopictus as the minority class while combining the rest of the classes as majority class. Also, it has to note that Souza et al. (2020) originally proposed seven INSECTS streams but we only adopted six of them which contain concept drifts and left the INSECT-out-of-control stream unused as it does not contain any concept drift.

To facilitate analysing the predictive performance of SMOClust, we also analysed the characteristics of the minority class of the real-world data streams, including the potential number of clusters, the ratios of safe, borderline, rare and outlier examples. Note that we only analysed the portion of the real-world data streams used in the actual experiments, which excludes the first 10% of each original real-world data streams that was used for hyper-parameter tuning (see Section 6.2.2 for details the hyper-parameter tuning procedure). The procedure of this analysis follows the methodology proposed by Brzezinski, Minku, et al. (2021) and it is described as follows.

The characteristics of each real-world data stream are estimated in successive batched of examples. We followed Brzezinski, Minku, et al. (2021) to use batch size of 2000 examples for all data streams except for Luxembourg and NOAA, where the batch size of 200 was

used as these data streams have only around 2000 examples. The class imbalance ratio and the ratios of each minority class type are estimated for each batch. It is worth noting that we only focused on analysing the class 1 because it is the global minority class of all the real-world data streams (see Table 6.3), even though this class could potentially become a majority during certain portions of the data stream, e.g., when there is potential concept drift affecting $P(Y)$ and changing the roles of majority and minority classes temporarily. As for types of minority class examples, they were estimated using the method proposed by Napierala and Stefanowski (2015). This method first finds the k-Nearest neighbours of each minority class example. Based on the class ratios among these k-Nearest neighbours, it then categorises each minority class example to as safe, borderline, rare, or outlier. Here, we followed Napierala and Stefanowski (2015) to adopt $k = 5$.

Following Brzezinski, Minku, et al. (2021)'s procedure, we also estimated the number of minority class clusters in each batch, using the affinity propagation algorithm (Frey and Dueck, 2007) and removing clusters with less than six minority class examples (Brzezinski, Minku, et al., 2021). The affinity propagation algorithm was run for thirty times with different random seeds for each batch. The average estimated number of minority class clusters is then recorded.

Lastly, we reported the ranges of the aforementioned characteristics across the different batched and their medians in Table 6.4. Note that we only performed analysis about types of minority class examples and the potential number of clusters on batched that contain at least six $(k + 1)$ minority class (class 1) examples. This is to prevent always categorising the minority class examples as rare cases or outliers when the total number of minority class examples in the batch is extremely low. The number of batched with less than size minority class examples is reported in brackets in the third column of Table 6.4.

As shown on Table 6.4, PAKDD2009 and NOAA streams usually present the most

Table 6.4: Characteristics of Real-World Data Streams (Values in the brackets are the median)

Stream	#Examples	#Batches (Uncounted)	Est. #Clust.	Minority Ratio	Safe	Borderline	Rare	Outlier
Luxembourg	1711	9 (0)	4-8 (6)	46%-53% (48%)	47%-62% (51%)	32%-48% (42%)	3%-9% (4%)	0%-3% (1%)
NOAA	16344	9 (0)	9-33 (28)	25%-37% (31%)	29%-45% (33%)	33%-50% (43%)	11%-18% (15%)	6%-13% (9%)
Ozone	2281	12 (5)	0-2 (0)	0%-18% (4%)	0%-22% (0%)	0%-61% (17%)	14%-58% (30%)	5%-70% (33%)
PAKDD2009	44998	23 (0)	19-32 (28)	17%-22% (20%)	0%-4% (2%)	27%-38% (32%)	33%-41% (36%)	22%-34% (30%)
$\text{Covtype}_{(c_1=\{1\text{-}6\})}$	522911	262 (8)	1-26 (1)	0%-76% (36%)	66%-99% (92%)	0%-26% (6%)	0%-8% (1%)	0%-5% (1%)
$\text{Covtype}_{(c_1=1)}$	522911	262 (2)	1-36 (1)	0%-91% (46%)	69%-100% (93%)	0%-25% (5%)	0%-5% (1%)	0%-4% (0%)
$\text{Covtype}_{(c_1=2)}$	522911	262 (149)	0-33 (5)	0%-89% (0%)	0%-100% (90%)	0%-78% (7%)	0%-14% (2%)	0%-11% (1%)
$\text{Covtype}_{(c_1=3)}$	522911	262 (240)	0-6 (1)	0%-3% (0%)	0%-62% (44%)	0%-67% (19%)	0%-38% (16%)	9%-67% (20%)
$\text{Covtype}_{(c_1=4)}$	522911	262 (108)	0-9 (3)	0%-10% (1%)	0%-100% (68%)	0%-79% (23%)	0%-56% (6%)	0%-50% (2%)
$\text{Covtype}_{(c_1=5)}$	522911	262 (159)	0-14 (3)	0%-28% (0%)	0%-99% (81%)	0%-61% (13%)	0%-38% (3%)	0%-29% (2%)
$\text{Covtype}_{(c_1=6)}$	522911	262 (110)	0-17 (2)	0%-21% (1%)	0%-100% (91%)	0%-93% (6%)	0%-33% (1%)	0%-22% (1%)
$\text{INSECTS}^{\text{inc.}}$	406840	204 (1)	0-27 (15)	0%-19% (9%)	0%-24% (5%)	0%-46% (32%)	0%-42% (29%)	14%-100% (33%)
$\text{INSECTS}_{\text{abr.}}$	319748	160 (0)	0-49 (13)	0%-39% (9%)	0%-50% (5%)	0%-53% (32%)	0%-46% (27%)	4%-100% (32%)
$\text{INSECTS}^{\text{inc.}}_{\text{grad.}}$	128981	65 (0)	7-27 (16)	5%-19% (10%)	0%-43% (8%)	17%-46% (33%)	5%-42% (27%)	8%-47% (28%)
$\text{INSECTS}^{\text{inc.}}_{\text{abr. re.}}$	406840	204 (1)	0-26 (15)	0%-18% (10%)	0%-29% (5%)	0%-57% (32%)	7%-41% (28%)	6%-93% (32%)
$\text{INSECTS}^{\text{inc.}}_{\text{re.}}$	406840	204 (1)	0-26 (15)	0%-17% (10%)	0%-27% (4%)	0%-58% (32%)	0%-41% (28%)	5%-100% (32%)

- $\text{Covtype}_{(c_1=x)}$: "$c_1=x$" refers to the class 1 is the class x in the original data set while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream. "$c_1=\{x_0\text{-}x_n\}$" refers to the class 1 is the class x_0-x_n in the original data set combined while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream.
- For all INSECTS data streams, "ae-albopictus" is the class 1. "inc." refers to incremental, "abr." refers to abrupt, "grad" refers to gradual, and "re." refers to recurring.

number of clusters of minority class examples, with medians of twenty eight clusters, meaning that the minority class is split into several clusters in this data stream. INSECTS streams usually present fewer clusters of the minority class than PAKDD2009 and NOAA streams, which have medians ranging from thirteen to sixteen clusters. Luxembourg, Ozone and Covtype usually present the least number of clusters of the minority class, having medians ranging from zero to six.

As for the types of minority class examples, Table 6.4 shows that the Ozone, PAKDD2009 and INSECTS streams mainly consist of borderline, rare, and outlier minority class examples. Luxembourg and NOAA streams mainly consist of safe and borderline minority class examples. Most Covtype streams mainly consist of safe minority class examples. Regarding the minority ratios, most of them have a small range, indicating that the potential concept drifts only affect $P(Y)$ with mild severity. In contrast, Covtype$_{(c_1=\{1\text{-}6\})}$, Covtype$_{(c_1=1)}$ and Covtype$_{(c_1=2)}$ streams have a very large range, indicating that that they potentially present severe concept drifts affecting $P(Y)$. In particular, Covtype$_{(c_1=2)}$ presents a large range of minority class ratio with a very small median (1%). This may indicate that the severe concept drifts affecting $P(Y)$ could potentially be abrupt.

6.2.2 Experiment Setup

This section presents the procedure of hyper-parameter tuning and experiments. The following are the approaches from the literature that were considered in this study and the reason behind the choice. All of these approaches are strict online approaches, which do not require storage of any past data, so that the comparisons are fair.

- OOB$_{(d)}$ and UOB$_{(d)}$ (S. Wang, Minku, and Yao, 2015): Baseline approaches able to handle class imbalanced data streams based on resampling, but that do not create

synthetic examples of the minority class.

- OnlineUnderOverBagging$_{(d)}$ (oUnderOverB$_{(d)}$) (B. Wang and Pineau, 2016): A simple existing approach which incorporates both undersampling and oversampling to handle class imbalance. We have slightly modified it with using time decay class size to determine the resampling rate, such that it can adapt to changes in class imbalance ratio dynamically.

- OnlineOversampling$_{(d)}$ (oOS$_{(d)}$): A simple approach which always uses the most recently seen minority class example for oversampling rather than creating synthetic data. This approach is used as a baseline to support the investigation of when the proposed strategy of creating synthetic minority class examples for oversampling is advantageous / disadvantageous.

- SMOGauNoise: This variation of the proposed approach is inspired by Song, Minku, and Yao (2018), which they proposed a data augmentation method for offline learning in software effort estimation. It has the same learning and making prediction strategies as the proposed approach but it always creates synthetic minority class examples for oversampling by adding Gaussian noise to the most recent seen minority class example, as done by Song, Minku, and Yao (2018). It is used to compare the proposed strategy against another method to generate synthetic examples in data stream learning setting. Note that this is the first time to investigate this data augmentation method proposed by Song, Minku, and Yao (2018) in the context of classification problems.

Approaches followed by "(d)" refers to these approaches were not prepared to handle concept drift originally[4]. We used a wrapper to create a version of them which can adopt a drift detector. Yet, their system will be reset upon concept drift detection. Besides, OnlineSMOTEBagging (B. Wang and Pineau, 2016) was also included in our preliminary

[4]Except OOB and UOB can handle concept drift affecting $P(Y)$.

experiments. However, it consumed too much memory and took over fourteen hours to complete a single run of hyper-parameter tuning. Thus, we decided to exclude it from our comparison because of its run-time and memory consumption are not feasible in real-world application, regardless its predictive performance.

In terms of evaluation method, we modified the periodic holdout test and applied it to experiments with artificial data streams. The modification allows the periodic holdout test to take the position of the minority class clusters, class imbalance ratio, and the proportions of borderline and rare examples into the consideration. In practice, to evaluate a given data stream learning approach, the data stream learning approach is tested on a holdout test set B_t^{test} of m examples after training on every n examples. This holdout test set is class balanced and it follows the same underlying joint probability distribution (concept) at the evaluation time step t, where $t \mod n = 0$. i.e., $B_t^{test} \sim P_t(X, Y)$.

In this study, each artificial data stream has 200k examples as mentioned in Section 6.2.1 and the approaches were evaluated at every 500 time steps, which means there are 400 test sets corresponds to each artificial data stream. During the evaluation, the approaches were tested at every n time steps. Their performance in G-Mean will then be sampled based on their prediction to the corresponding test sets. At the end of the evaluation, we summarise their performance across the stream by taking an average of their G-Mean on the test sets.

Table 6.5 presents the option considered in the hyper-parameter tuning test. Tables 6.6 and 6.6 present the chosen hyper-parameter values of the approaches. The hyper-parameters that are not mentioned in Tables 6.5, 6.6 and 6.7 were in MOA's default values.

To tune the hyper-parameters of the approaches, we have created an addition artificial data stream for this purpose. We denote this data stream as "hyper-parameter tuning stream". This stream was generated using the above procedure. It also consists of 200k examples with the concept drift happens from 70k to 100k time steps. The class imbalance

Table 6.5: Options for Hyper-parameters

Hyper-parameter	$OOB_{(d)}$ $UOB_{(d)}$	Online-Oversampling$_{(d)}$	Online-UnderOverBagging$_{(d)}$	SMOGauNoise	SMOClust
Base Learner	Hoeffding Tree with Naïve Bayes at leaves (HTNB)				
Ensemble Size (n)	15				
Drift Detector (ddm)[a]	{DDM, EDDM, ADWIN, DDM-OCI{α_{ddm}:{0.9}[b] θ_{ddm}:{0.9,0.99,0.999}}[b], DDM-GMean{α_{ddm}:{0.9}}[b], EWAUC{aucWindowSize:{50,200,500,1000}}}				
Class size fading factor (θ)	{0.9,0.99,0.999}				
Gaussian Noise Variance (σ^2)	-			{0.0001,0.0005,0.001}	
Categorical Change Probability (p)	-			{0,0.2,0.4}	
k-Nearest Neighbour (k)	-				{3,5,7}
Stream Clustering Method (sc)	-				{CluStream, ClusTree}

- Grey rows denote pre-chosen hyper-parameter values.

[a] Only applies to approaches employing a drift detector.

[b] α_{ddm} and θ_{ddm} refer to error fading factor and class size fading factor used in the corresponding concept drift detector respectively.

Table 6.6: Chosen Hyper-parameters of Approaches without Drift Detector

Stream Type	OOB	UOB	OnlineOversampling	OnlineUnderOverBagging
Artificial Stream	θ=0.999	θ=0.9	θ=0.9	θ=0.999
Luxembourg	θ=0.999	θ=0.9	θ=0.99	θ=0.99
NOAA	θ=0.99	θ=0.99	θ=0.99	θ=0.99
Ozone	θ=0.999	θ=0.99	θ=0.9	θ=0.999
PAKDD2009	θ=0.9	θ=0.99	θ=0.99	θ=0.9
$\text{Covtype}_{(c_1=\{1\text{-}6\})}$	θ=0.999	θ=0.999	θ=0.99	θ=0.999
$\text{Covtype}_{(c_1=1)}$	θ=0.99	θ=0.999	θ=0.99	θ=0.99
$\text{Covtype}_{(c_1=2)}$	θ=0.999	θ=0.999	θ=0.9	θ=0.999
$\text{Covtype}_{(c_1=3)}$	θ=0.999	θ=0.999	θ=0.9	θ=0.999
$\text{Covtype}_{(c_1=4)}$	θ=0.999	θ=0.99	θ=0.99	θ=0.999
$\text{Covtype}_{(c_1=5)}$	θ=0.999	θ=0.999	θ=0.9	θ=0.999
$\text{Covtype}_{(c_1=6)}$	θ=0.999	θ=0.999	θ=0.9	θ=0.999
$\text{INSECTS}^{\text{inc.}}$	θ=0.999	θ=0.99	θ=0.99	θ=0.999
$\text{INSECTS}_{\text{abr.}}$	θ=0.999	θ=0.999	θ=0.99	θ=0.99
$\text{INSECTS}^{\text{inc.}}_{\text{grad.}}$	θ=0.999	θ=0.999	θ=0.9	θ=0.99
$\text{INSECTS}^{\text{inc.}}_{\text{abr. re.}}$	θ=0.999	θ=0.999	θ=0.9	θ=0.99
$\text{INSECTS}^{\text{inc.}}_{\text{re.}}$	θ=0.999	θ=0.999	θ=0.999	θ=0.999

- $\text{Covtype}_{(c_1=x)}$: "c_1=x" refers to the class 1 is the class x in the original data set while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream. "c_1=$\{x_0 - x_n\}$" refers to the class 1 is the class x_0-x_n in the original data set combined while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream.
- For all INSECTS data streams, "ae-albopictus" is the class 1. "inc." refers to incremental, "abr." refers to abrupt, "grad" refers to gradual, and "re." refers to recurring.

Table 6.7: Chosen Hyper-parameters of Approaches with Drift Detector

Stream Type	OOB_d	UOB_d	Online-Oversampling$_d$	Online-UnderOverBagging$_d$	SMOGauNoise	SMOClust
Artificial Stream	θ=0.999, ddm=DDM	θ=0.999, ddm=DDM	θ=0.9, ddm=EDDM	θ=0.999, ddm=DDM	θ=0.9, σ^2=0.001, p=0.2, ddm=DDM	θ=0.99, σ^2=0.0005, sc=ClusTree, p=0.4, k=3, ddm=DDM
Luxembourg	θ=0.999, ddm=ADWIN	θ=0.9, ddm=EWAUC, window size=200	θ=0.999, ddm=EWAUC, window size=500	θ=0.99, ddm=EWAUC, window size=200	θ=0.9, σ^2=0.001, p=0.2, ddm=DDM-OCI, α_{ddm}=0.999, θ_{ddm}=0.9	θ=0.999, σ^2=0.001, sc=Clustrean, p=0.2, k=7, ddm=EWAUC, window size=1000
NOAA	θ=0.99, ddm=DDM	θ=0.99, ddm=DDM	θ=0.999, ddm=DDM-OCI, α_{ddm}=0.999, θ_{ddm}=0.9	θ=0.99, ddm=DDM	θ=0.99, σ^2=0.0005, p=0.2, ddm=DDM-OCI, α_{ddm}=0.9, θ_{ddm}=0.9	θ=0.99, σ^2=0.0001, sc=ClusTree, p=0.4, k=3, ddm=EWAUC, window size=200
Ozone	θ=0.999, ddm=DDM-GMean, α_{ddm}=0.9	θ=0.99, ddm=EWAUC, window size=50	θ=0.9, ddm=EWAUC, window size=500	θ=0.999, ddm=DDM-GMean, α_{ddm}=0.9	θ=0.9, σ^2=0.001, p=0.2, ddm=ADWIN	θ=0.9, σ^2=0.001, sc=CluStream, p=0.2, k=3, ddm=DDM-GMean, α_{ddm}=0.9
PAKDD2009	θ=0.99, ddm=ADWIN	θ=0.99, ddm=ADWIN	θ=0.99, ddm=EWAUC, window size=1000	θ=0.9, ddm=EWAUC, window size=50	θ=0.9, σ^2=0.0001, p=0.2, ddm=DDM-OCI, α_{ddm}=0.9, θ_{ddm}=0.9	θ=0.99, σ^2=0.0005, sc=ClusTree, p=0.2, k=3, ddm=DDM-OCI, α_{ddm}=0.9, θ_{ddm}=0.9
$Covtype_{(c_1=\{1\text{-}6\})}$	θ=0.999, ddm=EWAUC, window size=50	θ=0.999, ddm=DDM	θ=0.99, ddm=EWAUC, window size=1000	θ=0.99, ddm=EWAUC, window size=500	θ=0.9, σ^2=0.0005, p=0.2, ddm=EWAUC, window size=500	θ=0.9, σ^2=0.0001, sc=CluStream, p=0.2, k=3, ddm=EWAUC, window size=500
$Covtype_{(c_1=1)}$	θ=0.999, ddm=DDM-GMean, α_{ddm}=0.9	θ=0.9, ddm=DDM-GMean, α_{ddm}=0.9	θ=0.999, ddm=EWAUC, window size=500	θ=0.999, ddm=DDM-GMean, α_{ddm}=0.9	θ=0.999, σ^2=0.0001, p=0.2, ddm=DDM	θ=0.9, σ^2=0.0005, sc=CluStream, p=0.2, k=3, ddm=EWAUC, window size=50
$Covtype_{(c_1=2)}$	θ=0.999, ddm=EWAUC, window size=200	θ=0.999, ddm=EWAUC, window size=500	θ=0.99, ddm=EWAUC, window size=500	θ=0.999, ddm=EWAUC, window size=500	θ=0.999, σ^2=0.0005, p=0.4, ddm=EWAUC, window size=1000	θ=0.999, σ^2=0.0001, sc=CluStream, p=0.4, k=7, ddm=DDM-OCI, α_{ddm}=0.999, θ_{ddm}=0.9
$Covtype_{(c_1=3)}$	θ=0.9, ddm=DDM	θ=0.999, ddm=EWAUC, window size=1000	θ=0.9, ddm=EWAUC, window size=500	θ=0.999, ddm=EWAUC, window size=200	θ=0.99, σ^2=0.001, p=0.4, ddm=EWAUC, window size=1000	θ=0.99, σ^2=0.0005, sc=CluStream, p=0.4, k=7, ddm=ADWIN
$Covtype_{(c_1=4)}$	θ=0.999, ddm=EWAUC, window size=50	θ=0.9, ddm=EDDM	θ=0.99, ddm=DDM-OCI, α_{ddm}=0.999, θ_{ddm}=0.9	θ=0.999, ddm=EWAUC, window size=200	θ=0.9, σ^2=0.0005, p=0.4, ddm=EWAUC, window size=1000	θ=0.999, σ^2=0.0001, sc=CluStream, p=0.2, k=7, ddm=DDM-OCI, α_{ddm}=0.999, θ_{ddm}=0.9
$Covtype_{(c_1=5)}$	θ=0.999, ddm=EWAUC, window size=500	θ=0.999, ddm=EWAUC, window size=1000	θ=0.9, ddm=DDM-OCI, α_{ddm}=0.99, θ_{ddm}=0.9	θ=0.999, ddm=EWAUC, window size=500	θ=0.9, σ^2=0.001, p=0.2, ddm=DDM-OCI, α_{ddm}=0.99, θ_{ddm}=0.9	θ=0.99, σ^2=0.001, sc=CluStream, p=0.2, k=7, ddm=DDM-OCI, α_{ddm}=0.999, θ_{ddm}=0.9

$\text{Covtype}_{(c_1=6)}$	θ=0.999, ddm=EWAUC, window size=50	θ=0.999, ddm=EWAUC, window size=50	θ=0.99, ddm=EWAUC, window size=500	θ=0.999, ddm=EWAUC, window size=50	θ=0.99, σ^2=0.0001, p=0.4, ddm=EWAUC, window size=200	θ=0.999, σ^2=0.001, sc=CluStream, p=0.4, k=5, ddm=DDM
$\text{INSECTS}^{\text{inc.}}$	θ=0.99, ddm=ADWIN	θ=0.9, ddm=ADWIN	θ=0.9, ddm=ADWIN	θ=0.99, ddm=ADWIN	θ=0.99, σ^2=0.0001, p=0.2, ddm=ADWIN	θ=0.9, σ^2=0.001, sc=CluStream, p=0.2, k=3, ddm=DDM-OCI, α_{ddm}=0.99, θ_{ddm}=0.9
$\text{INSECTS}_{\text{abr.}}$	θ=0.999, ddm=EWAUC, window size=50	θ=0.999, ddm=DDM-OCI, α_{ddm}=0.9, θ_{ddm}=0.9	θ=0.999, ddm=EWAUC, window size=200	θ=0.999, ddm=EWAUC, window size=1000	θ=0.99, σ^2=0.0001, p=0.4, ddm=EWAUC, window size=1000	θ=0.99, σ^2=0.001, sc=CluStream, p=0.2, k=7, ddm=EWAUC, window size=50
$\text{INSECTS}^{\text{inc.}}_{\text{grad.}}$	θ=0.999, ddm=EWAUC, window size=500	θ=0.999, ddm=EWAUC, window size=500	θ=0.9, ddm=DDM	θ=0.99, ddm=EWAUC, window size=1000	θ=0.9, σ^2=0.0001, p=0.4, ddm=DDM	θ=0.9, σ^2=0.0001, sc=CluStream, p=0.2, k=3, ddm=DDM-GMean, α_{ddm}=0.9
$\text{INSECTS}^{\text{inc.}}_{\text{abr. re.}}$	θ=0.99, ddm=EWAUC, window size=500	θ=0.999, ddm=EWAUC, window size=200	θ=0.999, ddm=DDM	θ=0.99, ddm=EWAUC, window size=500	θ=0.99, σ^2=0.0001, p=0.4, ddm=EWAUC, window size=200	θ=0.99, σ^2=0.0001, sc=CluStream, p=0.2, k=3, ddm=DDM-OCI, α_{ddm}=0.9, θ_{ddm}=0.9
$\text{INSECTS}^{\text{inc.}}_{\text{re.}}$	θ=0.999, ddm=EWAUC, window size=1000	θ=0.9, ddm=DDM	θ=0.999, ddm=EWAUC, window size=200	θ=0.999, ddm=EWAUC, window size=1000	θ=0.999, σ^2=0.0001, p=0.4, ddm=EWAUC, window size=1000	θ=0.99, σ^2=0.001, sc=CluStream, p=0.4, k=3, ddm=DDM-GMean, α_{ddm}=0.9

- $\text{Covtype}_{(c_1=x)}$: "c_1=x" refers to the class 1 is the class x in the original data set while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream. "c_1=$\{x_0 - x_n\}$" refers to the class 1 is the class x_0-x_n in the original data set combined while the rest of the classes are combined to be the class 0 in the "Actual" experiment stream.
- For all INSECTS data streams, "ae-albopictus" is the class 1. "inc." refers to incremental, "abr." refers to abrupt, "grad" refers to gradual, and "re." refers to recurring.

ratio and the drift behaviour were randomly selected from the set of all combinations of drift factors. Following the convention used by Brzezinski, Minku, et al. (2021), the details of the hyper-parameter tuning stream can be expressed as "StaticIm42.24_Im1+Borderline40+Rare40", which was randomly chosen. The set of hyper-parameter values of each approach that leads to the best ten runs average of G-Mean across this hyper-parameter tuning stream was then be used in the experiments. In the experiments, we evaluated and compared their G-Mean by thirty runs average across each data stream.

As for the experiments with real-world data streams, the first 10% of each real-world data streams was used for the hyper-parameter tuning purposes. Traditional prequential evaluation was used because the ground truth of their underlying concepts are unknown in advance. The hyper-parameters of the approaches were then chosen based on the set of values leading to the best ten runs average of time decay G-Mean across the first 10% of each real-world data streams. The remaining 90% of the real-world data streams were then be used in the experiments. We evaluated and compared their time decay G-Mean by thirty runs average across each real-world data stream. The time decay G-Mean of the approaches were sampled at every 500 time steps, except they were sampled at every fifty and ten time steps for NOAA, Ozone and Luxembourg stream respectively. Also, all the data streams were normalised before hyper-parameter tuning and running experiments as required by the proposed approach and explained in Section 6.1.

The prequential performance is calculated using Equation 6.9, which is the same equation used in Sections 4.1.2.2, 4.2.2.2 and 5.2.2.

$$preq_perform_t(D) = \begin{cases} preq_perform^{ex}_t(D), & \text{if } t = f \\ \frac{preq_perform^{ex}_t(D) + \alpha \times preq_perform_{t-1}(D) \times (t-f)}{t-f+1}, & \text{otherwise} \end{cases} \tag{6.9}$$

where t is the current time step, $preq_perform_t^{ex}$ is 0 if the prediction to the current training example ex before learning is wrong and 1 if it is correct, f is the first time step used in the calculation, and α is the time decaying factor. In this work, we adopted $\alpha = 0.999$

In the experiments, each data stream learning approach ran for thirty times on each data stream with different random seeds. After running the experiments, their performance are group by different factor and their rankings in the groups are presented. Friedman test with a level of significance of 0.05 is applied to each group of factor to confirm if there is any statistical significance between the predictive performance of the data stream learning approaches. If there is, Nememyi post-hoc test is used to determine which approach is significantly different from the top-ranked approach. In the statistical tests, each group corresponds to a data stream learning approach while each observation within a group corresponds the average prequential predictive performance across the a given data stream in a single run.

Besides, their thirty run average predictive performance on each data stream is also presented to facilitate us in analysing the margin of the performance different. A12 effect size (Vargha and Delaney, 2000) is used to see how likely does the proposed approach performed better / worse than the existing approaches with such a margin of performance difference.

6.3 Analysis

This section presents the analysis done to evaluate the proposed approach – SMOClust. In Section 6.3.1, we first compare the predictive performance of SMOClust against existing approaches on artificial data streams considering different types of concept drift difficulty in the minority class and analyse when and why does the SMOClust performs better / worse than existing approaches, answering RQ3.2. In Section 6.3.2, we then compare their

predictive performance on different real-world data streams, answering RQ3.3.

6.3.1 Results with Artificial Data Streams

This section presents the analysis done to compare the predictive performance of SMOClust against nine existing approaches on artificial data streams that consider different drift difficulties in the minority class and different class imbalance ratio throughout the stream. This analysis allows us to have an idea of when SMOClust performed better and worse than existing approaches. Based on the general comparisons, representative cases were then chosen to investigate the behaviour of SMOClust when it performed better and worse than existing approaches in Sections 6.3.1.1 and 6.3.1.2 respectively.

Table 6.9 presents the Friedman rankings of average G-Mean of the approaches grouped by different concept drift factors on artificial data streams (presented in Table 6.1). It shows that SMOClust was one of the top-ranked approaches only when the minority class ratio throughout the stream is the lowest (1%). This preliminarily shows that SMOClust can handle severely class imbalanced data stream better than most other approaches, except it performed similarly to UOB and OnlineUnderOverBagging. In the same table, the group of rare cases shows SMOClust performed similar to approaches with low ranking. This may indicate that SMOClust is one of the approaches that do not handle rare cases very well. For other groups, although SMOClust was not one of the top ranked approaches, it usually performed similar to mid-ranked approaches.

However, Friedman rankings only show the relative position of approaches' predictive performance but they do not provide any information about the margin of difference. To investigate how much does SMOClust performed better in severely class imbalanced stream and worse in other groups of factor, we further compared their thirty runs average G-Mean

Table 6.9: Statistical (Friedman) Ranking of G-Mean on Artificial Streams Grouped by Factors

Groups	OOB	UOB	oOS	oUnder-OverB	OOB_d	UOB_d	oOS_d	oUnder-$OverB_d$	SMO-GauNoise	SMOClust
Imbalance Ratio Drift	3.7708	4.2167	4.6917	3.2708	6.8792	7.2417	7.6583	5.0125	6.7583	5.5
Double Factors	5.9733	6.7067	5.6667	6.58	4.0867	7.0067	5.7133	5.26	2.58	5.4267
Complex Factor	5.14	7.8933	5.2733	6.3533	4.2733	6.0333	6.5867	4.9933	2.94	5.5133
Single Factor Drift with Static Imbalance Ratio										
StaticIm{*}[a]_Split	6.8611	6.6278	7.45	5.5417	4.7222	6.6333	5.7167	3.7194	2.9611	4.7667
StaticIm{*}[a]_Move	5.1333	6.6556	4.7472	4.65	5.8167	7.7444	6.3028	5.3444	3.1889	5.4167
StaticIm{*}[a]_Merge	5.3806	6.6111	4.6861	3.8361	6.1861	7.7778	6.2417	4.7417	3.3	6.2389
StaticIm{*}[a]_Borderline	5.8111	5.9056	5.7833	4.2333	5.0944	7.3111	7.2556	4.0667	3.4222	6.1167
StaticIm{*}[a]_Rare	2.5167	6.6778	3.0417	4.3444	6.2889	7.2056	7.9917	5.5444	3.9667	7.4222
StaticIm30_{*}[b]	5.815	8.92	3.3883	6.9267	3.8117	7.8	3.7217	5.4967	2.19	6.93
StaticIm10_{*}[b]	4.75	8.23	4.8083	3.805	4.0667	8.42	6.925	3.5017	2.1633	8.33
StaticIm1_{*}[b]	4.8567	2.3367	7.2283	2.8317	8.9867	5.7833	9.4583	5.0517	5.75	2.7167
All	5.1106	6.4712	5.1754	4.8496	5.4083	7.1409	6.6633	4.814	3.5379	5.8288

[a] "StaticIm{*}[a]" refers to StaticIm{30/10/1}, which means the group includes all artificial data streams of that type in static minority class ratio of 30%, 10%, and 1% respectively.

[b] "{*}" refers to Split/Move/Merge/Borderline/Rare, which means the group includes all artificial data streams of the above five types of drifts with the same static minority class ratio.

- The p-values of Friedman tests are all $\leq$2.2E-16.
- Highlighted ranks denote significant superior performance.
- Underlined ranks denote the corresponding approach's performance have no statistical significance with SMOClust.

on each artificial data stream, presented in Table 6.10. A12 effect size (Vargha and Delaney, 2000) is adopted to compare how likely does SMOClust perform better or worse than existing approaches on each artificial data stream.

Although Table 6.9 shows that SMOClust usually achieved lower rankings than other approaches in less severe class imbalanced data streams, Table 6.10 shows that the margin of the under-performance was usually small as we can rarely see vivid orange cells in the table. In contrast, we can see a lot of vivid lime cells in sector StaticIm1 of Table 6.10. This reveals that the high ranking achieved by SMOClust in the group concerning minority class ratio

Table 6.10: 30 Runs Average G-Mean on Artificial Data Streams (A12 SMOClust vs Others)

Stream	OOB	UOB	oOS	oUnderOverB	OOB_d	UOB_d	oOS_d	$oUnderOverB_d$	SMO-GauNoise	SMOClust
StaticIm30										
StaticIm30_Split3	96.56%[-b]	96.33%[-b]	96.85%[-b]	96.6%[-b]	97.66%[b]	97.04%[*]	97.7%[b]	97.46%[b]	97.84%[b]	97.18%
StaticIm30_Move3	96.39%[-s]	95.7%[-b]	96.82%[m]	96.21%[-b]	96.8%[s]	95.92%[-b]	96.88%[m]	96.46%[-s]	97.19%[b]	96.61%
StaticIm30_Merge3	96.96%[-s]	96.41%[-b]	97.31%[m]	96.92%[-s]	97.32%[m]	96.5%[-b]	97.39%[b]	96.98%[*]	97.7%[b]	97.08%
StaticIm30_Split7	96.9%[-b]	96.62%[-b]	97.14%[-m]	96.83%[-b]	97.78%[b]	97.01%[-m]	97.75%[b]	97.45%[s]	97.99%[b]	97.36%
StaticIm30_Move7	96.41%[s]	95.7%[-b]	96.83%[b]	96.2%[*]	96.38%[s]	95.75%[-b]	96.71%[b]	96.17%[-s]	96.84%[b]	96.29%
StaticIm30_Merge7	97.34%[b]	96.86%[-s]	97.62%[b]	97.33%[m]	97.33%[b]	96.9%[-s]	97.53%[b]	97.33%[m]	97.6%[b]	97.09%
StaticIm30_Borderline20	97.17%[m]	97.1%[s]	97.26%[b]	97.2%[m]	97.35%[b]	97.16%[m]	97.27%[b]	97.28%[b]	97.42%[b]	96.9%
StaticIm30_Borderline100	95.7%[-s]	95.6%[-s]	95.96%[*]	95.73%[-s]	96.16%[s]	95.7%[-s]	96.11%[s]	95.97%[*]	96.3%[m]	95.89%
StaticIm30_Rare20	93.15%[b]	92.95%[b]	93.27%[b]	93.05%[b]	92.93%[b]	92.61%[b]	92.94%[b]	92.73%[b]	92.99%[b]	91.72%
StaticIm30_Rare100	71.6%[b]	69.1%[b]	72.11%[b]	67.03%[-s]	70.83%[b]	70.61%[b]	67.55%[s]	70.79%[b]	70.48%[b]	67.31%
StaticIm10										
StaticIm10_Split3	96.06%[s]	95.84%[*]	95.91%[*]	96.4%[b]	97.3%[b]	95.64%[-s]	96.71%[b]	97.43%[b]	97.57%[b]	95.81%
StaticIm10_Move3	96.16%[b]	94.73%[-b]	96.26%[b]	96.29%[b]	96.39%[b]	94.69%[-b]	95.59%[m]	96.37%[b]	96.82%[b]	95.36%
StaticIm10_Merge3	96.79%[b]	95.61%[-s]	96.96%[b]	97.02%[b]	97.02%[b]	95.67%[*]	96.35%[b]	97.04%[b]	97.43%[b]	95.87%
StaticIm10_Split7	96.36%[s]	96.12%[*]	96.08%[*]	96.59%[b]	97.43%[b]	95.55%[-b]	96.54%[b]	97.42%[b]	97.62%[b]	96.15%
StaticIm10_Move7	96.25%[b]	94.65%[-m]	96.33%[b]	96.34%[b]	96.15%[b]	94.57%[-m]	95.68%[b]	96.29%[b]	96.62%[b]	95.1%
StaticIm10_Merge7	97.21%[b]	96.16%[*]	97.36%[b]	97.47%[b]	97.13%[b]	96.18%[*]	96.76%[b]	97.43%[b]	97.52%[b]	96.11%
StaticIm10_Borderline20	97.21%[b]	96.98%[b]	97.12%[b]	97.32%[b]	97.2%[b]	96.85%[s]	96.69%[s]	97.37%[b]	97.37%[b]	96.64%
StaticIm10_Borderline100	95.73%[s]	95.45%[*]	95.64%[s]	95.86%[s]	96.02%[b]	94.69%[-b]	95.01%[-s]	96.09%[b]	96.18%[b]	95.48%
StaticIm10_Rare20	93.41%[b]	92.56%[b]	93.53%[b]	93.19%[b]	93.05%[b]	91.99%[m]	92.55%[b]	92.99%[b]	93.35%[b]	91.74%
StaticIm10_Rare100	72.91%[b]	62.94%[m]	70.19%[b]	68.8%[b]	68.27%[b]	68.76%[b]	51.42%[-b]	65.76%[b]	63.83%[b]	61.94%
StaticIm1										
StaticIm1_Split3	82.88%[-b]	89.88%[*]	74.12%[-b]	87.98%[-b]	58.16%[-b]	85.0%[-b]	50.0%[-b]	84.73%[-b]	82.01%[-b]	90.03%
StaticIm1_Move3	84.22%[-b]	91.28%[m]	74.16%[-b]	88.87%[-m]	63.31%[-b]	83.36%[-b]	49.59%[-b]	82.48%[-b]	77.1%[-b]	89.59%
StaticIm1_Merge3	88.14%[-b]	93.23%[b]	80.01%[-b]	91.68%[s]	62.83%[-b]	85.54%[-b]	53.14%[-b]	86.08%[-b]	82.98%[-b]	90.94%

StaticIm1_Split7	82.88%[-b]	90.08%[*]	74.62%[-b]	87.72%[-b]	60.18%[-b]	85.78%[-b]	53.32%[-b]	85.46%[-b]	79.56%[-b]	90.32%
StaticIm1_Move7	84.99%[-b]	92.21%[b]	74.64%[-b]	89.17%[-s]	62.19%[-b]	82.4%[-b]	46.07%[-b]	83.93%[-b]	75.46%[-b]	89.67%
StaticIm1_Merge7	89.39%[-b]	94.12%[b]	82.11%[-b]	92.6%[m]	67.24%[-b]	85.9%[-b]	62.51%[-b]	87.98%[-b]	84.47%[-b]	91.19%
StaticIm1_Borderline20	92.55%[-b]	96.34%[b]	89.44%[-b]	95.19%[-s]	64.7%[-b]	91.6%[-b]	68.54%[-b]	92.32%[-b]	89.66%[-b]	95.52%
StaticIm1_Borderline100	89.3%[-b]	93.52%[*]	84.25%[-b]	93.19%[*]	55.61%[-b]	84.27%[-b]	43.86%[-b]	80.26%[-b]	67.17%[-b]	93.29%
StaticIm1_Rare20	90.37%[-b]	91.24%[-b]	87.27%[-b]	92.27%[-s]	63.16%[-b]	77.91%[-b]	73.08%[-b]	91.46%[-b]	92.95%[s]	92.6%
StaticIm1_Rare100	68.79%[b]	58.12%[-b]	61.17%[s]	67.67%[b]	37.73%[-b]	41.75%[-b]	42.28%[-b]	62.02%[m]	71.92%[b]	60.8%
Imbalance Ratio Drift										
StaticIm10_Im1	98.11%[b]	96.27%[-s]	97.98%[b]	98.06%[b]	98.05%[b]	95.8%[-s]	97.51%[b]	98.06%[b]	98.01%[b]	96.66%
StaticIm1_Im10	93.08%[-b]	95.51%[s]	89.22%[-b]	95.78%[m]	78.05%[-b]	89.72%[-b]	73.33%[-b]	91.38%[-b]	85.05%[-b]	94.86%
Im1	97.98%[b]	97.88%[b]	98.25%[b]	97.75%[b]	92.58%[*]	93.59%[-b]	88.8%[*]	95.15%[s]	92.31%[-m]	96.97%
StaticIm1_Im50	93.27%[-b]	96.39%[b]	89.72%[-b]	95.78%[s]	79.14%[-b]	90.66%[-b]	75.44%[-b]	91.47%[-b]	85.77%[-b]	95.37%
Double Factors										
Im1+Rare100	51.35%[-b]	68.42%[b]	51.5%[-b]	46.6%[-b]	49.82%[-b]	65.26%[b]	45.57%[-b]	48.76%[-b]	52.74%[-b]	53.74%
Im10+Rare60	79.26%[b]	77.67%[-b]	79.8%[b]	78.3%[-b]	79.05%[b]	77.34%[-b]	79.04%[b]	78.13%[-b]	79.67%[b]	78.58%
Split5+Im10	96.58%[-m]	95.55%[-b]	96.66%[-m]	96.74%[-s]	97.58%[b]	96.17%[-b]	97.47%[b]	97.52%[b]	97.89%[b]	96.94%
Im1+Borderline100	93.93%[-b]	94.87%[-s]	93.1%[-b]	94.58%[-m]	95.4%[*]	92.61%[-b]	93.39%[-m]	94.87%[*]	88.1%[*]	95.28%
Im10+Borderline20	97.19%[b]	96.88%[s]	97.17%[b]	97.23%[b]	97.27%[b]	96.96%[s]	97.1%[m]	97.25%[b]	97.37%[b]	96.77%
Complex Factors										
StaticIm10_Split5+Im1+Rare100	61.46%[b]	48.75%[-b]	58.21%[b]	56.93%[*]	57.75%[b]	58.28%[*]	46.11%[-b]	54.66%[-b]	59.65%[b]	56.94%
StaticIm10_Split5+Im1+Borderline100	84.66%[-b]	87.89%[*]	79.73%[-b]	86.88%[-m]	81.97%[-b]	84.9%[-s]	54.87%[-b]	84.59%[-b]	82.97%[-b]	87.56%
Split5+Im10+Borderline40+Rare40	80.21%[b]	77.88%[-b]	80.43%[b]	79.4%[-m]	80.71%[b]	79.91%[s]	79.88%[b]	80.55%[b]	80.96%[b]	79.72%
Split5+Im10+Borderline80	90.84%[-b]	90.06%[-b]	91.11%[-s]	90.86%[-b]	91.32%[-s]	90.17%[-b]	90.58%[-m]	91.35%[*]	91.4%[*]	91.45%
Im10+Borderline20+Rare20	91.01%[b]	90.46%[*]	91.06%[b]	90.92%[b]	91.24%[b]	90.99%[b]	91.33%[b]	91.17%[b]	91.51%[b]	90.51%

- Based on the average G-Mean, cells are highlighted in lime / light grey when SMOClust performed better than the corresponding approach and cells are highlighted in orange / dark grey cells when SMOClust performed worse than the corresponding approach. The colour intensity scales with the absolute difference of average G-Mean between the SMOClust and the approach of the column and the intensity reaches the maximum when such a difference is $\geq 10\%$.
- Symbols [*], [s], [m] and [b] represent insignificant, small, medium and large A12 effect size against SMOClust. Presence/absence of the sign "-" in the effect size means that the corresponding approach was worse/better than SMOClust.

of 1% (StaticIm1_{*}) in Table 6.9 was mostly by a large margin of difference in average G-Mean. Besides, we can also see that the vivid orange cells mainly appear in artificial data streams that present concept drift leading to have 100% rare cases in the minority class (Rare100), especially in the cells comparing against OOB and OnlineUnderOverBagging. This confirms that SMOClust could not handle rare cases very well when compared against existing approaches.

Besides, Table 6.9 shows that UOB and OnlineUnderOverBagging performed similarly to SMOClust in the group of StaticIm1_{*}. The sector of StaticIm1 in Table 6.10 reveals more information about when do these three approaches perform differently. It shows that UOB performed better than SMOClust with medium-small margin (around 3% G-Mean) in StaticIm1 cases presenting concept drift of moving and merging minority class sub-clusters. It also shows that UOB performed worse than SMOClust with a small margin (around 1-2% G-Mean) in StaticIm1 cases presenting concept drift of increasing rare minority class ratio. It performed similarly to SMOClust in other StaticIm1 cases.

As for OnlineUnderOverBagging, the same sector shows that it performed better than SMOClust with small margin (around 1% G-Mean) in StaticIm1 cases presenting concept drift of merging minority class clusters and a large margin (around 7% G-Mean) in StaticIm1_Rare100 case. Also, it performed worse than SMOClust with medium-small margin (around 2-3% G-Mean) in StaticIm1 cases presenting concept drift of spliting minority class clusters. In general, UOB and OnlineUnderOverBagging both performed similarly to SMOClust in most StaticIm1 cases, except OnlineUnderOverBagging performed a lot better in StaticIm1_Rare100 case.

Based on the aforementioned results, we performed additional experiments with the same set of single factor drift artificial data stream but enforced extremely severe class imbalance ratios (presented in Table 6.2) to further evaluate SMOClust. These additional

experiments aim to check and confirm whether SMOClust achieves superior performance over most other approaches in severely class imbalanced data streams.

Table 6.12 presents the Friedman rankings of G-Mean of the approaches grouped by different concept drift factors on the severely class imbalanced artificial data streams. It shows that SMOClust can generally achieve higher rankings when the class imbalance ratio is very severe. As in the groups concerning minority class ratio of 5% and 3% (StaticIm5_{*} and StaticIm3_{*}), we can see that SMOClust still could not achieve high rankings. This may due to the fact that the artificial data streams are long enough to have quite a lot of minority class examples, despite the minority class ratios were extremely low. Therefore, the advantage of SMOClust was not manifested.

Table 6.13 further confirms this deduction. It presents the average G-Mean achieved by the approaches on the severely class imbalanced artificial data streams and the A12 effect size comparison results. In general, it shows that SMOClust performed a lot better than most other approaches when the minority class ratio throughout the stream being extremely severe ($\leq 1\%$). It also shows that SMOClust generally did not handle rare cases very well, especially when compared against OOB, OnlineUnderOverBagging and SMOGauNoise in data streams that presents concept drifts of Rare100. These results are consistent with previous results of experiments with artificial data streams considering less severe class imbalance ratio and different concept drift difficulties.

Further looking into Table 6.13, sectors of StaticIm5 and StaticIm3 show that SMOClust usually performed slightly worse than most existing approaches but it performed better than OnlineOversampling$_{\text{d}}$. The potential reason is that the drift detector may negatively affect the predictive performance of OnlineOversampling$_{\text{d}}$ because the comparison between the average G-Mean of SMOClust and OnlineOversampling in sectors of StaticIm5 and StaticIm3 does not yield the same observation.

Table 6.12: Statistical (Friedman) Ranking of G-Mean on Severely Class Imbalanced Artificial Streams Grouped by Factors

Groups	OOB	UOB	oOS	oUnder-OverB	OOB_d	UOB_d	oOS_d	oUnder-$OverB_d$	SMO-GauNoise	SMOClust
StaticIm{*}[a]_Split	5.7389	2.6028	8.0472	3.6139	8.2056	5.2444	9.2889	4.1694	4.8333	3.2556
StaticIm{*}[a]_Move	4.7875	3.1472	7.4958	2.8194	7.4319	6.1694	9.7153	4.7639	5.125	3.5444
StaticIm{*}[a]_Merge	4.7708	3.05	7.1736	2.6417	7.8292	6.0028	9.5875	4.6556	5.1361	4.1528
StaticIm{*}[a]_Borderline	4.65	2.6306	6.5611	2.6306	8.7111	6.2139	9.5139	4.8972	6.0417	3.15
StaticIm{*}[a]_Rare	3.1583	5.15	5.6444	2.8056	8.1361	7.5944	9.1861	5.2278	3.7944	4.3028
StaticIm5_{*}[b]	4.2067	7.16	6.18	2.525	5.15	7.7533	9.2167	3.0817	2.5033	7.2233
StaticIm3_{*}[b]	4.46	5.5333	7.2	2.2633	7.2033	6.75	9.6467	2.8533	3.27	5.82
StaticIm1_{*}[b]	4.9933	2.36	7.32	2.9233	8.53	5.58	9.59	5.3067	5.7467	2.65
StaticIm07_{*}[b]	4.8	1.9267	7.2133	3.25	9.02	5.3633	9.5133	5.49	6.1333	2.29
StaticIm05_{*}[b]	4.6167	1.61	6.9667	3.2683	9.3333	5.8633	9.37	5.755	6.1033	2.1133
StaticIm03_{*}[b]	4.65	1.3067	7.0267	3.1833	9.14	6.16	9.4133	5.97	6.16	1.99
All	4.6211	3.3161	6.9844	2.9022	8.0628	6.245	9.4583	4.7428	4.9861	3.6811

[a] "StaticIm{*}[a]" refers to StaticIm{5/3/1/07/05/03}, which means the group includes all artificial data streams of that type in static minority class ratio of 5%, 3%, 1%, 0.7%, 0.5%, and 0.3% respectively.

[b] "{*}" refers to Split/Move/Merge/Borderline/Rare, which means the group includes all artificial data streams of the above five types of drifts with the same static minority class ratio.

- The p-values of Friedman tests are all ≤2.2E-16.
- Highlighted ranks denote significant superior performance.
- Underlined ranks denote the corresponding approach's performance have no statistical significance with SMOClust.

As for the sectors concerning minority class ratios ≤ 1% in Table 6.13 (sectors of StaticIm{1/07/05/03}), we can see that there are lots of vivid lime cells, which indicate that SMOClust performed better than most other approaches when the data stream is extremely class imbalanced. Also, it is interesting to see that the case of StaticIm03_Rare100 presents a row of vivid lime cells, which does not follow the previous deduction of SMOClust handles rare minority class examples poorer than other approaches. As for UOB, it performed better than SMOClust in cases other than Rare100 drift. In particular, the margin of the superior performance of UOB over SMOClust increases as the minority class ratio throughout the stream decreases, case by case. As for OnlineUnderOverBagging, it generally performed

Table 6.13: 30 Runs Average Prequential G-Mean on Severely Class Imbalanced Artificial Data Streams (A12 SMOClust vs Others)

Stream	OOB	UOB	oOS	oUnderOverB	OOB_d	UOB_d	oOS_d	$oUnderOverB_d$	SMOGauNoise	SMOClust
StaticIm5										
StaticIm5_Split3	95.24%[*]	95.54%[s]	94.47%[-b]	96.1%[b]	96.2%[b]	95.26%[*]	94.09%[-s]	97.11%[b]	97.11%[b]	95.19%
StaticIm5_Move3	95.26%[b]	93.64%[-m]	94.74%[s]	95.93%[b]	95.46%[b]	93.48%[-b]	88.6%[-b]	95.88%[b]	96.11%[b]	94.23%
StaticIm5_Merge3	96.01%[b]	94.64%[*]	95.88%[b]	96.67%[b]	96.18%[b]	94.59%[*]	89.64%[-b]	96.65%[b]	96.87%[b]	94.7%
StaticIm5_Split7	95.07%[*]	95.23%[s]	94.05%[-b]	95.82%[b]	95.46%[m]	94.61%[-s]	92.68%[-s]	96.94%[b]	96.86%[b]	94.99%
StaticIm5_Move7	95.56%[b]	93.37%[-b]	94.83%[m]	95.93%[b]	95.12%[b]	93.02%[-b]	90.63%[-b]	95.74%[b]	95.93%[b]	94.31%
StaticIm5_Merge7	96.72%[b]	95.4%[s]	96.42%[b]	97.16%[b]	96.36%[b]	95.22%[*]	92.4%[-b]	97.0%[b]	97.15%[b]	95.21%
StaticIm5_Borderline20	97.1%[b]	96.92%[s]	96.76%[*]	97.42%[b]	96.39%[-s]	96.73%[*]	94.59%[-b]	97.26%[b]	97.25%[b]	96.69%
StaticIm5_Borderline100	95.63%[s]	94.99%[-s]	95.12%[-s]	95.99%[m]	95.18%[-s]	94.31%[-b]	86.67%[-b]	96.15%[b]	96.02%[m]	95.45%
StaticIm5_Rare20	93.68%[b]	92.29%[s]	93.51%[b]	93.46%[b]	92.73%[b]	91.55%[-b]	90.99%[-b]	93.04%[b]	93.66%[b]	92.15%
StaticIm5_Rare100	73.34%[b]	59.64%[-s]	68.17%[b]	69.93%[b]	66.84%[b]	66.64%[b]	47.73%[-b]	61.41%[s]	60.23%[*]	60.53%
StaticIm3										
StaticIm3_Split3	93.67%[-b]	95.28%[m]	91.81%[-b]	95.12%[m]	92.24%[-b]	94.84%[s]	87.39%[-b]	96.2%[b]	95.94%[b]	94.51%
StaticIm3_Move3	93.84%[s]	93.04%[-s]	91.87%[-b]	95.21%[b]	93.03%[*]	92.35%[-m]	80.77%[-b]	94.86%[b]	94.83%[b]	93.27%
StaticIm3_Merge3	95.04%[b]	94.03%[*]	93.86%[*]	96.19%[b]	93.78%[*]	93.55%[*]	84.34%[-b]	95.86%[b]	96.09%[b]	93.7%
StaticIm3_Split7	93.2%[-b]	95.05%[b]	90.86%[-b]	94.76%[m]	90.64%[-b]	93.94%[*]	84.29%[-b]	95.8%[b]	95.76%[b]	94.15%
StaticIm3_Move7	94.33%[b]	92.86%[-m]	92.31%[-b]	95.33%[b]	92.73%[-s]	92.23%[-b]	82.03%[-b]	94.74%[b]	94.4%[b]	93.5%
StaticIm3_Merge7	95.93%[b]	94.97%[*]	94.81%[*]	96.79%[b]	94.35%[*]	94.61%[-s]	85.91%[-b]	96.23%[b]	95.85%[b]	94.79%
StaticIm3_Borderline20	96.58%[*]	96.8%[s]	95.97%[-b]	97.23%[b]	92.88%[-b]	96.43%[-s]	91.48%[-b]	96.83%[s]	96.53%[*]	96.66%
StaticIm3_Borderline100	94.97%[-s]	94.59%[-m]	93.93%[-b]	95.77%[m]	91.38%[-b]	93.86%[-b]	70.99%[-b]	95.81%[m]	94.69%[*]	95.2%
StaticIm3_Rare20	93.43%[b]	92.09%[-m]	92.97%[b]	93.59%[b]	90.17%[-b]	90.87%[-b]	90.02%[-b]	93.33%[b]	94.44%[b]	92.34%
StaticIm3_Rare100	71.72%[b]	62.22%[b]	66.12%[b]	69.63%[b]	64.93%[b]	62.23%[b]	53.0%[-b]	62.63%[b]	61.83%[s]	59.92%
StaticIm1										
StaticIm1_Split3	84.29%[-b]	90.98%[*]	75.65%[-b]	88.67%[-b]	61.71%[-b]	85.62%[-b]	54.23%[-b]	85.79%[-b]	82.17%[-b]	90.91%
StaticIm1_Move3	82.76%[-b]	91.14%[m]	71.59%[-b]	87.92%[-b]	70.48%[-b]	85.17%[-b]	46.61%[-b]	81.17%[-b]	76.86%[-b]	89.55%
StaticIm1_Merge3	86.61%[-b]	92.27%[b]	77.68%[-b]	90.9%[s]	67.28%[-b]	85.85%[-m]	52.57%[-b]	84.94%[-b]	82.54%[-b]	90.22%

StaticIm1_Split7	82.03%[-b]	89.38%[*]	73.04%[-b]	87.2%[-b]	58.85%[-b]	85.31%[-b]	52.58%[-b]	84.55%[-b]	79.12%[-b]	90.25%
StaticIm1_Move7	83.3%[-b]	91.7%[b]	73.53%[-b]	88.48%[-m]	69.71%[-b]	81.62%[-b]	47.2%[-b]	81.25%[-b]	76.87%[-b]	89.36%
StaticIm1_Merge7	88.16%[-b]	93.49%[b]	81.61%[-b]	92.15%[m]	74.91%[-b]	85.81%[-b]	60.94%[-b]	86.48%[-b]	84.54%[-b]	91.15%
StaticIm1_Borderline20	92.42%[-b]	96.49%[m]	89.57%[-b]	95.32%[-m]	72.26%[-b]	91.37%[-b]	70.68%[-b]	92.25%[-b]	90.22%[-b]	95.9%
StaticIm1_Borderline100	89.38%[-b]	93.84%[*]	84.72%[-b]	93.39%[-s]	55.97%[-b]	86.42%[-b]	43.44%[-b]	77.92%[-b]	67.67%[-b]	93.66%
StaticIm1_Rare20	90.09%[-b]	91.35%[-b]	87.3%[-b]	92.24%[-m]	71.2%[-b]	80.35%[-b]	70.04%[-b]	90.97%[-b]	92.31%[*]	92.89%
StaticIm1_Rare100	68.79%[b]	58.9%[-b]	61.54%[*]	67.6%[b]	43.52%[-b]	45.74%[-b]	42.37%[-b]	62.08%[*]	71.26%[b]	61.5%
StaticIm07										
StaticIm07_Split3	77.51%[-b]	89.86%[*]	65.19%[-b]	83.34%[-b]	45.48%[-b]	80.25%[-b]	36.62%[-b]	74.97%[-b]	70.32%[-b]	89.45%
StaticIm07_Move3	74.0%[-b]	90.55%[b]	57.69%[-b]	81.13%[-b]	43.83%[-b]	76.82%[-b]	34.75%[-b]	70.38%[-b]	63.8%[-b]	87.26%
StaticIm07_Merge3	79.16%[-b]	91.48%[b]	66.0%[-b]	85.44%[-s]	47.07%[-b]	77.02%[-b]	37.31%[-b]	75.7%[-b]	70.53%[-b]	87.18%
StaticIm07_Split7	75.59%[-b]	88.17%[*]	63.64%[-b]	81.65%[-b]	41.83%[-b]	77.0%[-b]	37.01%[-b]	74.84%[-b]	70.73%[-b]	88.22%
StaticIm07_Move7	74.2%[-b]	91.4%[b]	59.56%[-b]	82.08%[-b]	45.21%[-b]	76.38%[-b]	36.29%[-b]	69.28%[-b]	62.5%[-b]	87.43%
StaticIm07_Merge7	81.7%[-b]	92.91%[b]	71.34%[-b]	87.48%[-m]	50.87%[-b]	81.99%[-b]	48.24%[-b]	78.85%[-b]	75.74%[-b]	88.97%
StaticIm07_Borderline20	89.33%[-b]	96.39%[b]	84.19%[-b]	93.29%[-b]	56.19%[-b]	88.59%[-b]	48.37%[-b]	89.35%[-b]	80.2%[-b]	95.1%
StaticIm07_Borderline100	84.99%[-b]	93.42%[s]	77.45%[-b]	90.82%[-b]	39.67%[-b]	76.94%[-b]	30.87%[-b]	66.34%[-b]	56.31%[-b]	92.59%
StaticIm07_Rare20	87.34%[-b]	90.98%[-b]	82.05%[-b]	90.52%[-b]	51.72%[-b]	71.26%[-b]	45.94%[-b]	87.73%[-b]	87.15%[-b]	92.51%
StaticIm07_Rare100	68.59%[b]	58.39%[-b]	58.71%[-b]	66.68%[b]	25.99%[-b]	40.18%[-b]	28.49%[-b]	62.65%[m]	70.08%[b]	61.97%
StaticIm05										
StaticIm05_Split3	67.41%[-b]	89.06%[m]	51.8%[-b]	75.26%[-b]	28.85%[-b]	70.03%[-b]	24.15%[-b]	61.4%[-b]	58.01%[-b]	86.92%
StaticIm05_Move3	60.73%[-b]	89.92%[b]	42.99%[-b]	71.27%[-b]	25.71%[-b]	60.37%[-b]	22.91%[-b]	52.3%[-b]	49.96%[-b]	82.97%
StaticIm05_Merge3	68.16%[-b]	90.63%[b]	52.0%[-b]	77.04%[-b]	26.29%[-b]	63.85%[-b]	25.45%[-b]	60.63%[-b]	53.63%[-b]	84.28%
StaticIm05_Split7	66.73%[-b]	87.47%[s]	52.25%[-b]	74.33%[-b]	29.97%[-b]	64.71%[-b]	26.36%[-b]	64.25%[-b]	59.18%[-b]	85.98%
StaticIm05_Move7	61.93%[-b]	91.08%[b]	44.91%[-b]	72.82%[-b]	32.82%[-b]	57.0%[-b]	25.03%[-b]	54.0%[-b]	49.63%[-b]	84.18%
StaticIm05_Merge7	72.74%[-b]	92.48%[b]	59.63%[-b]	80.73%[-b]	39.25%[-b]	69.5%[-b]	31.25%[-b]	70.66%[-b]	64.57%[-b]	86.78%
StaticIm05_Borderline20	84.19%[-b]	96.29%[b]	76.48%[-b]	90.35%[-b]	30.95%[-b]	79.13%[-b]	36.96%[-b]	77.36%[-b]	69.86%[-b]	93.92%
StaticIm05_Borderline100	77.54%[-b]	93.15%[b]	68.43%[-b]	87.03%[-b]	26.19%[-b]	57.71%[-b]	25.39%[-b]	52.65%[-b]	47.62%[-b]	91.05%
StaticIm05_Rare20	82.38%[-b]	90.85%[-m]	74.39%[-b]	87.71%[-b]	33.53%[-b]	60.14%[-b]	38.56%[-b]	81.84%[-b]	78.82%[-b]	91.51%
StaticIm05_Rare100	65.62%[m]	59.16%[-b]	55.02%[-b]	65.53%[b]	14.14%[-b]	32.84%[-b]	26.25%[-b]	56.47%[-b]	67.64%[m]	63.7%
StaticIm03										
StaticIm03_Split3	49.77%[-b]	88.35%[b]	33.95%[-b]	58.57%[-b]	14.32%[-b]	45.41%[-b]	13.85%[-b]	42.25%[-b]	39.95%[-b]	81.31%
StaticIm03_Move3	40.57%[-b]	88.81%[b]	22.43%[-b]	49.93%[-b]	18.07%[-b]	40.61%[-b]	10.55%[-b]	35.38%[-b]	35.13%[-b]	75.16%
StaticIm03_Merge3	49.77%[-b]	89.5%[b]	31.09%[-b]	59.18%[-b]	17.83%[-b]	46.84%[-b]	14.48%[-b]	41.79%[-b]	35.16%[-b]	76.22%

StaticIm03_Split7	50.45%[-b]	86.36%[b]	32.87%[-b]	57.4%[-b]	15.86%[-b]	48.09%[-b]	14.5%[-b]	41.15%[-b]	39.14%[-b]	80.4%
StaticIm03_Move7	41.14%[-b]	90.27%[b]	23.33%[-b]	53.4%[-b]	18.43%[-b]	33.73%[-b]	10.78%[-b]	33.98%[-b]	34.82%[-b]	75.72%
StaticIm03_Merge7	56.06%[-b]	91.5%[b]	38.89%[-b]	65.82%[-b]	23.33%[-b]	52.59%[-b]	16.89%[-b]	52.72%[-b]	49.98%[-b]	78.12%
StaticIm03_Borderline20	70.16%[-b]	96.09%[b]	55.01%[-b]	81.64%[-b]	23.08%[-b]	58.81%[-b]	19.34%[-b]	56.61%[-b]	43.26%[-b]	91.8%
StaticIm03_Borderline100	61.02%[-b]	92.72%[b]	49.14%[-b]	76.31%[-b]	18.34%[-b]	43.35%[-b]	15.68%[-b]	37.21%[-b]	37.93%[-b]	88.13%
StaticIm03_Rare20	69.26%[-b]	90.63%[s]	53.43%[-b]	79.4%[-b]	19.93%[-b]	44.64%[-b]	19.35%[-b]	59.5%[-b]	53.38%[-b]	89.69%
StaticIm03_Rare100	56.62%[-b]	58.64%[-b]	45.25%[-b]	60.56%[-b]	10.95%[-b]	24.78%[-b]	16.73%[-b]	47.49%[-b]	57.16%[-b]	64.62%

- Based on the average G-Mean, cells are highlighted in lime / light grey when SMOClust performed better than the corresponding approach and cells are highlighted in orange / dark grey cells when SMOClust performed worse than the corresponding approach. The colour intensity scales with the absolute difference of average G-Mean between the SMOClust and the approach of the column and the intensity reaches the maximum when such a difference is $\geq 10\%$.
- Symbols [*], [s], [m] and [b] represent insignificant, small, medium and large A12 effect size against SMOClust. Presence/absence of the sign "-" in the effect size means that the corresponding approach was worse/better than SMOClust.

worse than SMOClust in cases other than Rare100 drift, where the margin increase as the minority class ratio throughout the stream decreases, case by case.

Focusing on the cases of Rare100 drift in Table 6.13, we can see that the average G-Mean of SMOClust gradually and slightly increases as the minority class ratio of the stream decreases, case by case. In contrast, the average G-Mean of other approaches rapidly decreases as the minority class ratio of the stream decreases, case by case. This shows that the vivid lime cells in the case of StaticIm03_Rare100 were because SMOClust managed to perform similarly in handling concept drift leading to 100% rare minority class examples despite the minority class ratio of the stream, but other approaches were negatively affected by the severely low minority class ratio. In comparison, UOB performed similarly to SMOClust in cases of Rare100 drift when minority class ratios $> 1\%$. It then start to performed worse than SMOClust in cases of Rare100 when the the minority class ratio of the stream decreases, case by case. Meanwhile, OnlineUnderOverBagging performed better than SMOClust with a large margin in cases of Rare100 drift when the minority class ratios $> 1\%$. Yet, the margin decreases gradually as the minority class ratio of the stream decreases, case by case, until minority class ratio reaches 0.3%.

One reason could be the fact that the compared approaches all focus on learning the most recent decision areas of both classes by design, whereas SMOClust was designed to reinforce its knowledge in past minority class decision areas. When the minority class ratio throughout the stream was extreme (0.03%), the total number of minority class examples throughout the artificial data stream was very small ($200\text{k} \times 0.03\% = 60$). This also means that the number of locations spawning rare minority class examples after the drift is very small. SMOClust can then fully focus on learning these few locations, thus, it can perform better than the compared approaches when the minority class ratio throughout the stream being extreme.

Besides, we also discussed that UOB tends to perform better than SMOClust in most cases as the minority class ratio of the stream decreases, case by case. This potentially because UOB is an aggressive approach which usually tries to learn and to perform well in the minority class but sacrifices its predictive performance in the majority class. It is because it usually considers majority class areas in between minority class areas as minority class. This strategy may be advantageous especially when the data stream only presents concept drift of minority class clusters movements and the class imbalance ratio is extremely severe. In other words, the improvements in the recall on the minority class may occur in higher detriment of the majority class than for the proposed approach. In the detailed analysis (Section 6.3.1.1 and 6.3.1.2), we further look into this characteristic of UOB by inspecting its learnt decision area at different time steps of the chosen representative cases.

In Sections 6.3.1.1 and 6.3.1.2, representative cases were chosen to discuss why SMOClust performed better and worse than existing approaches respectively, providing a more detailed understanding of the results.

6.3.1.1 Cases where SMOClust performed better

This section discusses why SMOClust performed better than most other approaches in artificial data streams with severe class imbalance ratio when the class imbalance ratio is extremely severe (minority class ratio $\leq 1\%$ throughout the stream). StaticIm1-Move7 stream was chosen from Tables 6.10 as the representative case to discuss the behaviour of SMOClust in details.

As mentioned in Section 6.2.1, the artificial data streams have five input attributes and a class label. Therefore, it is difficult to visualise the learnt decision areas of the approaches and understand their behaviour in details. Because of this, we created a version of the representative streams with two input attributes and a class label while preserving

the characteristics which include the class imbalance ratio and the drift difficulty factors etc. Note that we only created a single copy of each two dimensional representative stream, such that we can compare the data stream learning approaches with their median predictive performance in thirty runs on the same data stream. Also, the hyper-parameters of the approaches were tuned based on a separated random two dimensional artificial data stream, following the procedure explained in Section 6.2.2. The chosen hyper-paramters of the approaches are presented in Tables 6.15.

Table 6.15: Chosen Hyper-parameters of Approaches for 2D Data Streams

OOB	UOB	Online-Oversampling	Online-UnderOverBagging	OOB_d	UOB_d
θ=0.999	θ=0.9	θ=0.9	θ=0.999	θ=0.999, *ddm*=DDM	θ=0.99, *ddm*=DDM
Online-$Oversampling_d$	Online-$UnderOverBagging_d$	SMOGauNoise			
θ=0.999, *ddm*=DDM	θ=0.999, *ddm*=DDM	θ=0.9, σ^2=0.001, p=0.2, *ddm*=DDM			
SMOClust					
θ=0.999, σ^2=0.0005, *sc*=CluStream, p=0.4, k=7, *ddm*=DDM					

Table 6.16 presents the their thirty runs average G-Mean on the two dimensional version of StaticIm1_Move7 stream. It shows that SMOClust performed the best. These results are slightly inconsistent with the results of the corresponding five dimensional stream in Tables 6.10, where SMOClust performed slightly worse than UOB but similarly to OnlineUnderOverBagging. Yet, in general, SMOClust still performed better than other approaches in both two dimensional and five dimensional versions of StaticIm1_Move7 stream. This may indicate that SMOClust tends to perform better in low dimensional data stream. Anyhow, the detailed analysis presented in the following paragraphs can still explain the characteristics

of SMOClust and why it performed better than most other approaches in this representative case.

Table 6.16: 30 Runs Average G-Mean on Two Dimensional Version of Representative Artificial Data Streams where SMOClust Performed Better (A12 SMOClust vs Others)

Stream	OOB	UOB	oOS	oUnderOverB	OOB_d
StaticIm1_Move7	82.11%[-b]	76.3%[-b]	79.46%[-b]	85.26%[-b]	53.45%[-b]
Stream	UOB_d	oOS_d	$oUnderOverB_d$	SMOGauNoise	SMOClust
StaticIm1_Move7	56.94%[-b]	76.88%[-b]	45.12%[-b]	82.94%[-b]	91.23%

- Based on the average G-Mean, cells are highlighted in lime / light grey when SMOClust performed better than the corresponding approach and cells are highlighted in orange / dark grey cells when SMOClust performed worse than the corresponding approach. The colour intensity scales with the absolute difference of average G-Mean between the SMOClust and the approach of the column and the intensity reaches the maximum when such a difference is $\geq 10\%$.
- Symbols [*], [s], [m] and [b] represent insignificant, small, medium and large A12 effect size against SMOClust. Presence/absence of the sign "-" in the effect size means that the corresponding approach was worse/better than SMOClust.

Figure 6.4 presents the approaches' predictive performance over time steps of their median run[5]. It shows that SMOClust performed the best in most time steps. In particular, SMOClust maintained the predictive performance to have at least 50% G-Mean on the class balanced holdout test sets during the concept drift (from 70k to 100k time steps) and recovered from the drift better than other approaches (the solid red line has a rapid recovery since 100k time steps). In contrast, other approaches usually dropped to around 0-20% G-Mean during the drift. This case showed the superior performance achieved by SMOClust in handling severely class imbalanced drifting data streams.

[5]Median run refers to the run that leads to the median of predictive performances averaged across time steps.

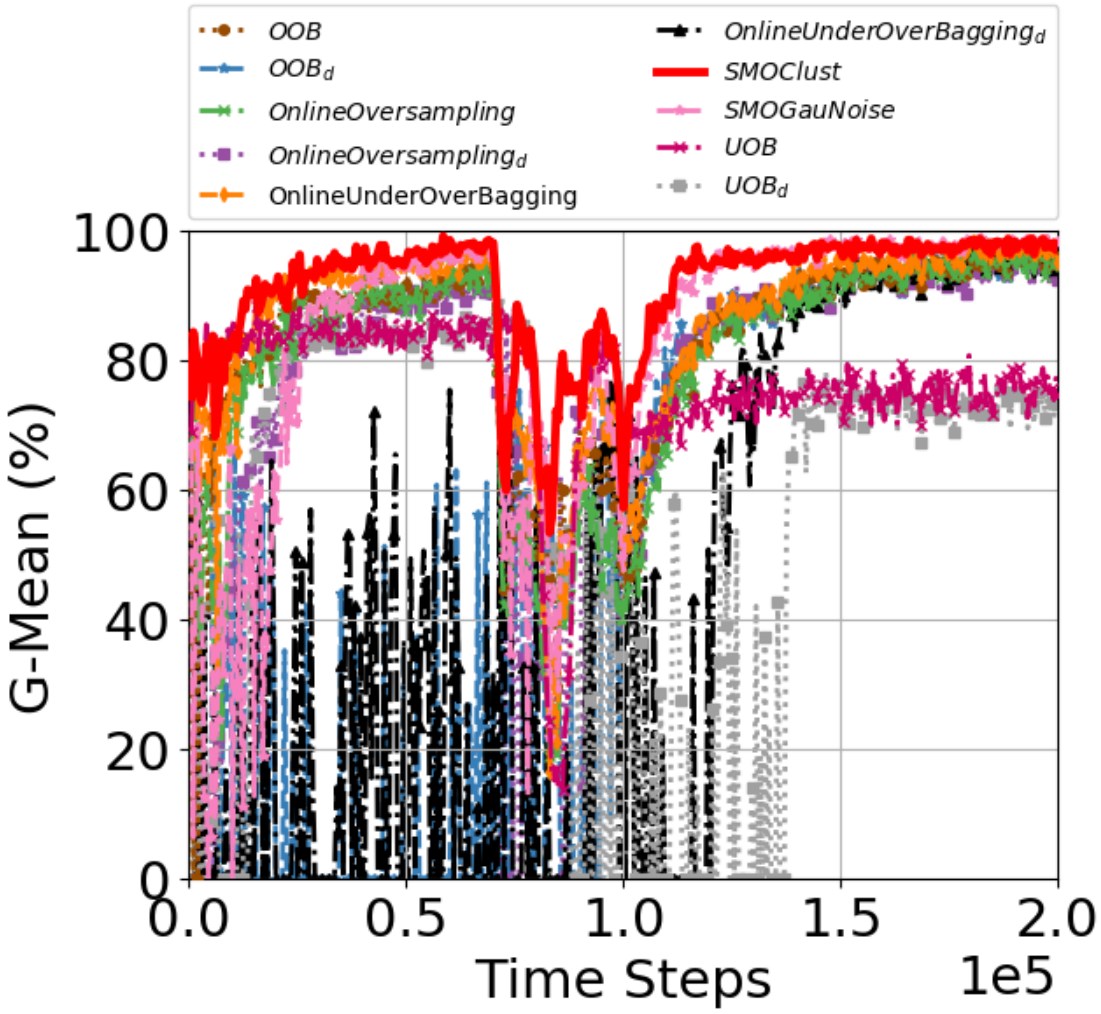
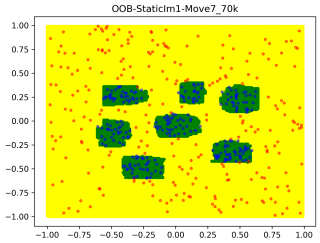


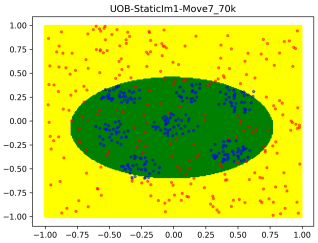
Figure 6.4: Periodic Class Balanced Holdout Test G-Mean Against Time Steps in Two Dimensional StaticIm1_Move7

Figures 6.5, 6.6 and 6.7 visualise the learnt decision areas of approaches at the time steps right before and after concept drift (70k and 100k time steps) and at the end (200k time steps) of the two dimensional StaticIm1_Move7 stream respectively. The yellow and green regions represent their learnt decision areas of class 0 (majority class) and class 1 (minority class) respectively, while the red and blue dots are the class 0 (majority class) and class 1 (minority class) examples in the class balanced test set, corresponding to the time steps.

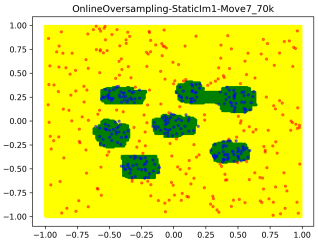
First of all, we compare the learnt decision areas of the approaches at the time steps



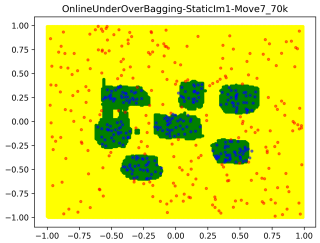
(a) OOB



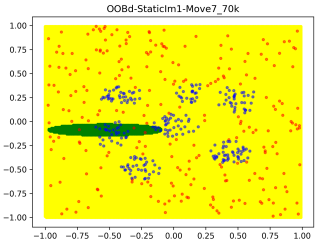
(b) UOB



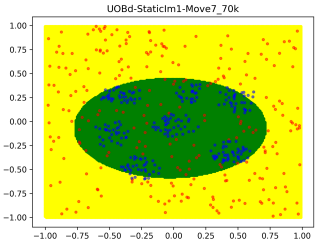
(c) OnlineOversampling



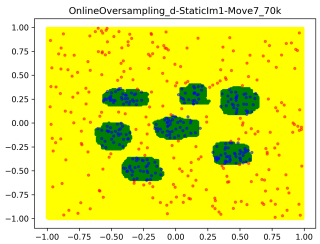
(d) OnlineUnderOverBagging



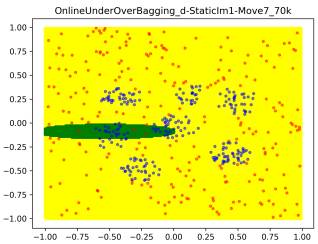
(e) OOB_d



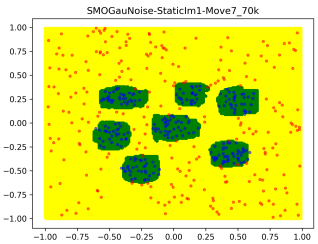
(f) UOB_d



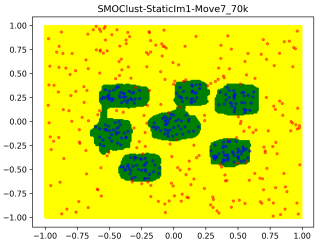
(g) $\text{OnlineOversampling}_\text{d}$



(h) $\text{OnlineUnderOverBagging}_\text{d}$



(i) SMOGauNoise



(j) SMOClust

Figure 6.5: Decision Areas Against Class Balanced Test Set at 70k Time Steps (Before Drift) of Two Dimensional StaticIm1_Move7

right before concept drift (at 70k time steps). Figure 6.5 shows that OOB, OnlineOversampling, OnlineUnderOverBagging, SMOGauNoise and SMOClust had learnt decision areas which match the corresponding class balanced test set. This explains why they performed the best before the drift (0-70k time steps, Figure 6.4). Figure 6.5(i) and Figure 6.5(j) show that the learnt decision areas of SMOClust were similar to SMOGauNoise because they both have strategies to explore the minority class decision boundaries. The expansion by SMOClust was slightly more aggressive than SMOGauNoise, with some sub-areas linked together. Although the proposed synthetic minority oversampling strategy prioritises "safe" areas to generate synthetic minority class examples, the strategy of using synthetic examples to train the stream clustering methods may contribute to the aggressiveness in exploring the minority class decision boundaries.

Figures 6.5(a) and 6.5(c) show that OOB and OnlineOversampling learnt the most compact minority class decision areas because they reuse the existing minority class examples for oversampling. Figure 6.5(d) shows that the minority class decision areas of OnlineUnderOverBagging were slightly larger than that of OOB and OnlineOversampling. Particularly, there were two green areas linked together. This may be the result of using oversampling and undersampling together, which managed to cover the true minority class clusters while preserving some aggressiveness from undersampling. In contrast, Figures 6.5(b) and 6.5(f) show that UOB and UOB_d learnt a single cluster to aggressively cover most minority class areas, considering the small majority class areas in between as part of the minority class. This is likely to cost some predictive performance in the majority class. Thus, we can see that UOB and UOB_d performed slightly worse than the other approaches before the concept drift (0-70k time steps, Figure 6.4). However, Table 6.13 shows that UOB performed slightly better than SMOClust in the five-dimensional StaticIm1_Move7 stream, indicating that the aggressive nature of undersampling may be an advantage in learning the minority class when the feature space is sparse and presents very few minority class exam-

ples. When the feature space is more compact, the proposed strategy in SMOClust is more advantageous.

Considering OOB_d and $\text{OnlineUnderOverBagging}_\text{d}$, Figure 6.4 shows that their predictive performance fluctuated considerably before the concept drift (0-70k time steps, Figure 6.4). Figures 6.5(e) and 6.5(h) show that their learnt minority decision areas were small which only covered a small proportion of the true minority class areas. Thus, it can be deduced that they were greatly affected by false-positive drift detections.

Over the next paragraphs, we compare the predictive performance and the decision boundaries of SMOClust against other approaches at the time steps right after concept drift (at 100k time steps) and at the end of the data stream (at 200k time steps), to understand how SMOClust handles concept drift of moving minority class sub-clusters when the data stream is severely class imbalanced.

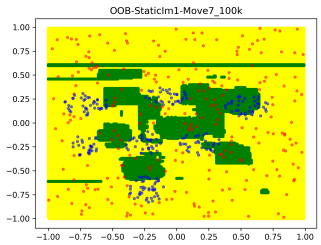
Figure 6.4 shows that the predictive performance of SMOClust fluctuated during the drift (70k-100k time steps, Figure 6.4). Thus, it is likely that its base learner had been reset several times due to drift detection. Yet, it was the fastest approach to recovering predictive performance from the drift. Figure 6.6 presents the learnt decision boundaries right after the drift. It shows that SMOClust and SMOGauNoise made the best attempt in adapting the drift. They were able to cover most minority class sub-clusters at the new position, especially SMOClust. The potential reason is that, although the base learner of SMOClust is reset upon drift detection, the stream clustering methods are not reset as they are expected to be drift adaptable. Therefore, SMOClust is more robust to incremental and gradual drifts than SMOGauNoise, explaining its rapid predictive performance recovery from the drift.

On the other hand, Figures 6.6(a), 6.6(b), 6.6(c), and 6.6(d) show that the learnt minority class decision areas of OOB, UOB, OnlineOversampling and OnlineUnderOverBagging mainly retained at the pre-drift position because they are not concept drift adaptable. Their

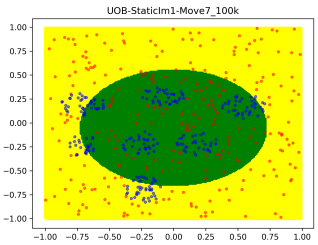
concept drift adaptable counterparts did not handle the drift very well either. Figures 6.6(e), 6.6(f), 6.6(g), and 6.6(h) show that their learnt minority class decision areas only covered a few minority class sub-clusters at the post-drift position, which is likely because their base learners had been reset for several times caused by drift detection and they do not have any strategy to deal with incremental and gradual drifts. As the result, they struggled to recover their predictive performance from the drift, as shown in Figure 6.4.

Lastly, we compare the learnt decision areas of the approaches at the end of the two-dimensional StaticIm1_Move7 stream. Figure 6.7 shows that OOB_d, SMOGauNoise and SMOClust are the best approaches in converging to the post-drift position of minority class sub-clusters. In particular, a few green areas of SMOGauNoise and SMOClust were slightly less compact than OOB_d, showing that hat SMOClust had slightly better generalisation than OOB_d.

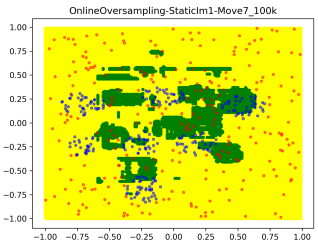
Figures 6.7(a), 6.7(c), and 6.7(d) show that OOB, OnlineOversampling and OnlineUnderOverBagging managed to converge to the new concept after the drift. However, they also retained a small portion of green areas which corresponds to the pre-drift position of the minority class. This shows that OOB, OnlineOversampling and OnlineUnderOverBagging can adapt to concept drift involving minority class sub-cluster movement. However, they required a longer period to adapt as they were hindered by the knowledge acquired pre-drift. Meanwhile, Figures 6.7(e), 6.7(g), and 6.7(h) show that their concept drift adaptable counterparts adapted better, except $\text{OnlineUnderOverBagging}_\text{d}$. While resetting base learners helps to adapt to concept drift, $\text{OnlineUnderOverBagging}_\text{d}$ partly uses undersampling in its strategy to deal with class imbalance led to some over-generalisation between the learnt minority class areas. UOB and UOB_d use undersampling to deal with class imbalance, thus Figures 6.7(a) and 6.7(f) show that they had the greatest over generalisation due to the aggressive nature of undersampling.



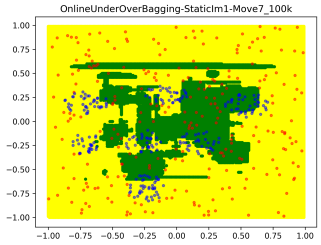
(a) OOB



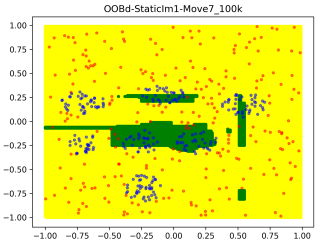
(b) UOB



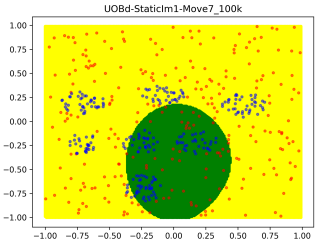
(c) OnlineOversampling



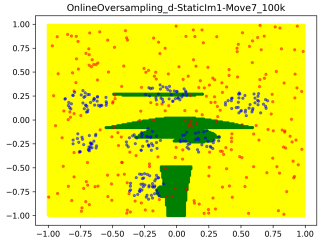
(d) OnlineUnderOverBagging



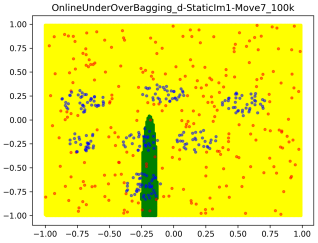
(e) OOB_d



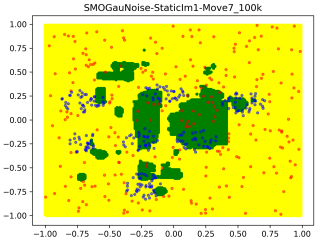
(f) UOB_d



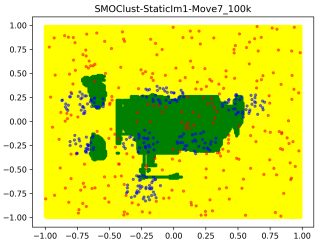
(g) $OnlineOversampling_d$



(h) $OnlineUnderOverBagging_d$

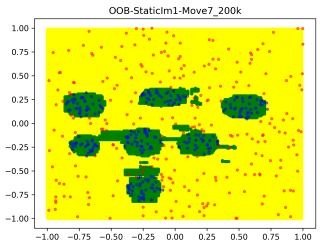


(i) SMOGauNoise

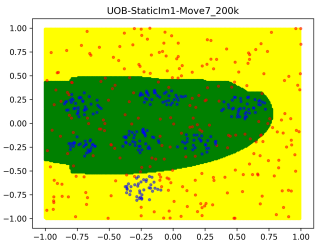


(j) SMOClust

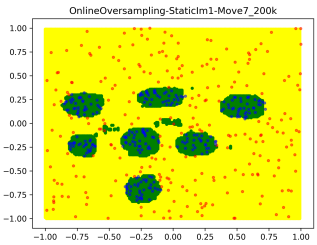
Figure 6.6: Decision Areas Against Class Balanced Test Set at 100k Time Steps (After Drift) of Two Dimensional StaticIm1_Move7



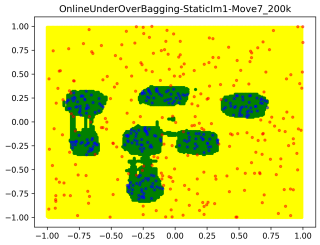
(a) OOB



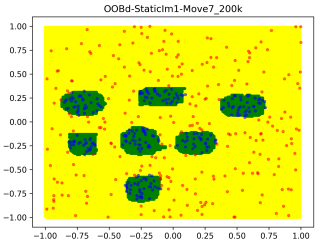
(b) UOB



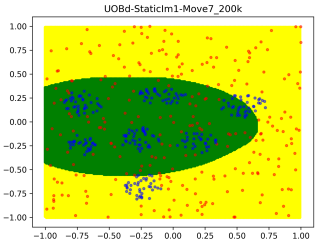
(c) OnlineOversampling



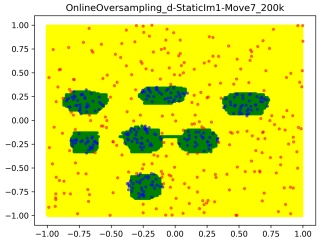
(d) OnlineUnderOverBagging



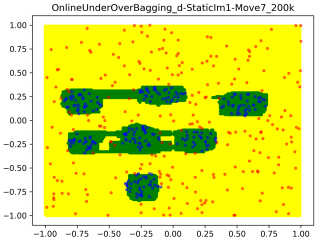
(e) OOB_d



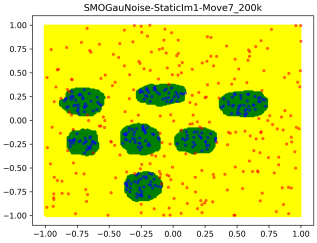
(f) UOB_d



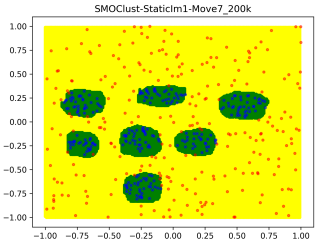
(g) $OnlineOversampling_d$



(h) $OnlineUnderOverBagging_d$



(i) SMOGauNoise



(j) SMOClust

Figure 6.7: Decision Areas Against Class Balanced Test Set at 200k Time Steps (End of Stream) of Two Dimensional StaticIm1_Move7

Short Summary: Through the pre-drift analysis, the ability of SMOClust in handling stationary severely class imbalanced data streams presenting several minority class sub-clusters is validated. In particular, it shows that SMOClust was able to learn and explore the true decision boundaries despite the data stream presenting very few minority class examples. The post-drift analysis shows that SMOClust was more robust in adapting incremental and gradual drift involving minority class sub-clusters movement than existing approaches. Although most of the approaches converged to the new concept at the end of the data stream, SMOClust was the best and the fastest approach in recovering predictive performance from the drift. The inconsistent results between two and five-dimensional versions of this representative case indicate that SMOClust may be more advantageous in lower-dimensional data streams.

6.3.1.2 Cases where SMOClust performed worse

This section discusses the situations where SMOClust performed worse than other approaches, particularly in cases with concept drift leading to 100% rare minority examples. StaticIm10_Rare100 stream was chosen from Tables 6.10 as the representative case to discuss the behaviour of SMOClust in detail. Following the method of analysis in Section 6.3.1.1, we also created a two-dimensional version of StaticIm10_Rare100 stream such that we can visualise and compare the learnt decision boundaries of the approaches to understand their behaviour.

Table 6.17 presents the approaches' thirty runs average G-Mean on the two dimensional StaticIm10_Rare100 stream. It shows that SMOClust performed better than most other approaches. Figure 6.8, showing the G-Mean of the approaches in their median run[6] throughout the two dimensional StaticIm10_Rare100 stream, also supports the results on

[6]Median run refers to the run that leads to the median of predictive performances averaged across time steps.

Table 6.17. As these results are not consistent with the results of the five-dimensional StaticIm10_Rare100 stream, shown in Table 6.9 and 6.12, we preliminary checked if using a different set of random seeds or picking another case that involves drift leading to 100% rare minority class examples would yield results that are consistent with Table 6.9 and 6.12. Yet, it still shows that SMOClust performed similar to or better than other approaches in two dimensional StaticIm10_Rare100 stream. Thus, in this analysis, we focus on why SMOClust can handle concept drift leading to 100% rare minority class examples than other approaches when the data stream has only two dimensions while attempting to deduce why it could not when the data stream has five dimensions.

Table 6.17: 30 Runs Average G-Mean on Two Dimensional Version of Representative Artificial Data Streams where SMOClust Performed Worse (A12 SMOClust vs Others)

Stream	OOB	UOB	oOS	oUnderOverB	OOB_d
StaticIm10_Rare100	70.61%[b]	63.65%[-b]	69.14%[-b]	68.19%[-b]	65.49%[-b]
Stream	UOB_d	oOS_d	$oUnderOverB_d$	SMOGauNoise	SMOClust
StaticIm10_Rare100	68.17%[-b]	65.04%[-b]	64.98%[-b]	64.56%[-b]	70.32%

- Based on the average G-Mean, cells are highlighted in lime / light grey when SMOClust performed better than the corresponding approach and cells are highlighted in orange / dark grey cells when SMOClust performed worse than the corresponding approach. The colour intensity scales with the absolute difference of average G-Mean between the SMOClust and the approach of the column and the intensity reaches the maximum when such a difference is $\geq 10\%$.
- Symbols [*], [s], [m] and [b] represent insignificant, small, medium and large A12 effect size against SMOClust. Presence/absence of the sign "-" in the effect size means that the corresponding approach was worse/better than SMOClust.

Figures 6.9, 6.10 and 6.11 visualise the learnt decision areas of the approaches at the time steps right before and after concept drift (70k and 100k time steps) and at the end (200k time steps) of the two dimensional StaticIm10_Rare100 stream respectively. The yellow and green regions represent their learnt decision areas of class 0 (majority class) and

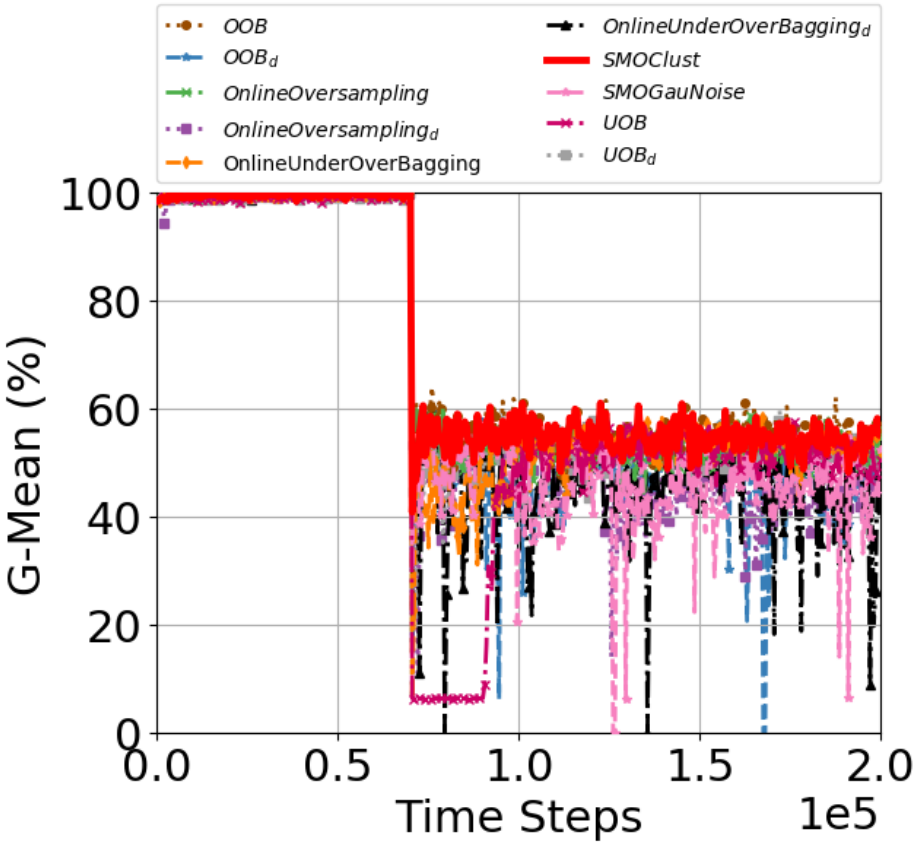
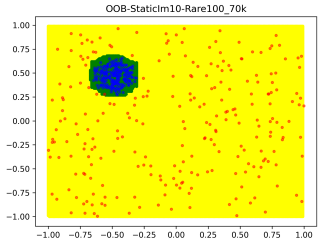


Figure 6.8: Periodic Class Balanced Holdout Test G-Mean Against Time Steps in Two Dimensional StaticIm10_Rare100

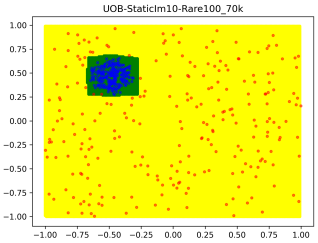
class 1 (minority class) respectively, while the red and blue dots are the class 0 (majority class) and class 1 (minority class) examples in the class balanced test set which corresponds to the time steps.

Figure 6.8 shows that all approaches performed very well during the pre-drift period (0-70k time steps). Figure 6.9 reveals that it is because they learnt the decision boundary of the pre-drift concept very well, as the minority class was just a single cluster. While most approaches learnt a oval shape decision boundary, UOB and UOB$_d$ learnt a rectangular shape, which could be due to the use of undersampling.

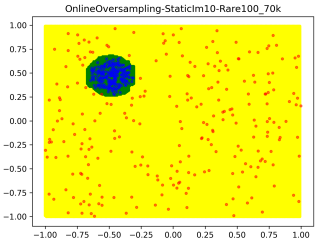
Figure 6.8 shows that the predictive performance of the approaches dropped to below



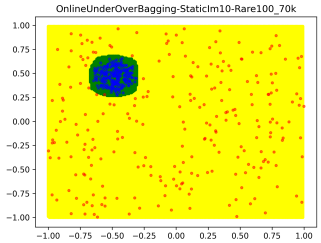
(a) OOB



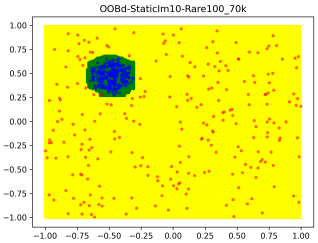
(b) UOB



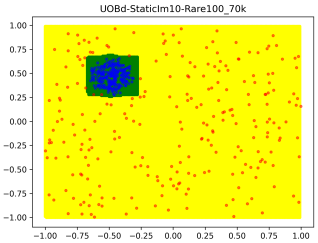
(c) OnlineOversampling



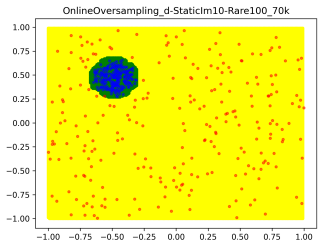
(d) OnlineUnderOverBagging



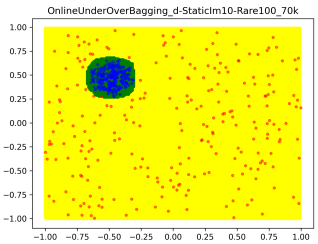
(e) OOB_d



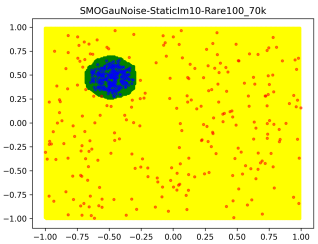
(f) UOB_d



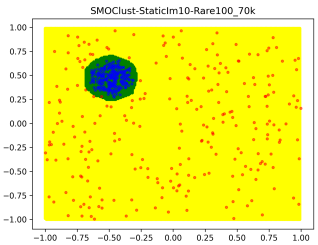
(g) $\text{OnlineOversampling}_\text{d}$



(h) $\text{OnlineUnderOverBagging}_\text{d}$



(i) SMOGauNoise



(j) SMOClust

Figure 6.9: Decision Areas Against Class Balanced Test Set at 70k Time Steps (Before Drift) of Two Dimensional StaticIm10_Rare100

60% G-Mean and started to differ since the concept drift began (70k time steps). While most approaches' predictive performance fluctuated with large magnitude, SMOClust's predictive performance was relatively steady, bouncing between 50%-60% G-Mean. UOB performed poorly since the drift began at 70k time steps until the drift was close to finishing at 100k time steps, indicating that undersampling struggled in dealing with this drift without the help of a concept drift detector.

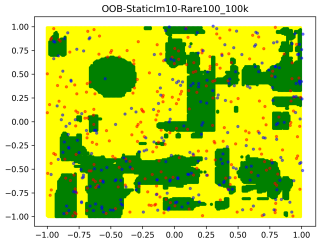
Figure 6.10 presents the learnt decision boundaries of the approaches right after the drift (100k time steps). It shows that OOB, OnlineOversampling, OnlineUnderOverBagging, $\text{OnlineOversampling}_{\text{d}}$, SMOGauNoise, and SMOClust learnt very complex decision areas, indicating that they made great efforts to learn all the areas that spawn rare minority class examples belonging to the post-drift concept. However, only approaches with a concept drift detector were able to forget the old area of the minority class at the top left corner. This shows that, although this drift was gradual, concept drift detection was important in helping the system to forget irrelevant past knowledge. In contrast, approaches without a drift detector retained the oval minority class cluster at the top left corner which belongs to the pre-drift concept. Most of them struggled to perform well since the drift started at 70k time steps, as shown in Figure 6.8. OOB was an exception in terms of predictive performance. However, the fact that it retained the knowledge about the pre-drift minority class areas makes it disadvantageous in dealing with other types of drift, as discussed in Section 6.3.1.1.

Comparing the learnt decision areas of SMOClust against other approaches with drift detector (OOB_{d}, UOB_{d}, $\text{OnlineOversampling}_{\text{d}}$, $\text{OnlineUnderOverBagging}_{\text{d}}$ and SMOGauNoise), it can be observed that the learnt minority class areas of SMOClust were complex and covered the feature space spawning minority class examples the most. While $\text{OnlineOversampling}_{\text{d}}$'s and SMOGauNoise's were also complex, they did not cover the feature space spawning minority class examples as much as SMOClust's did. $\text{OnlineOversampling}_{\text{d}}$

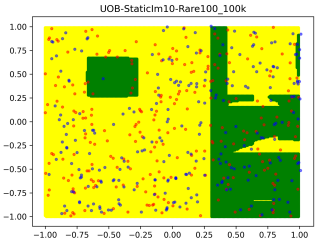
only reuses the recently seen minority class example for oversampling likely leads to overfitting to the most recent area. SMOGauNoise also has a strategy to explore the decision boundaries of the minority class, but it only explores the area around the recently seen minority class example. This could be disadvantageous when false-positive drift detections were triggered, resetting the base learner. SMOClust, on the other hand, does not have this disadvantage because the stream clustering methods are not reset upon drift detection. This makes it more robust to false-positive drift detections than other approaches. As the drift was gradual, OOB_d, UOB_d and $\text{OnlineUnderOverBagging}_\text{d}$ likely also suffered from multiple drift detection, as Figures 6.10(e), 6.10(f) and 6.10(h) show that the learnt a simple decision boundary right after the drift.

Figure 6.11 presents the learnt decision boundaries of the approaches at the end of the two dimensional StaticIm10_Rare100 stream (at 200k time steps). While most approaches continued to further improve their learnt decision boundaries since the drift had finished, Figures 6.11(h) and 6.11(j) show that $\text{OnlineUnderOverBagging}_\text{d}$ and SMOGauNoise did not improve as much as other approaches, meaning that they suffered from false-positive drift detection during the post-drift period. Also, we can see that, again, UOB and UOB_d developed a very large and mostly continuous learnt decision area for the minority class, confirming the aggressive nature of undersampling.

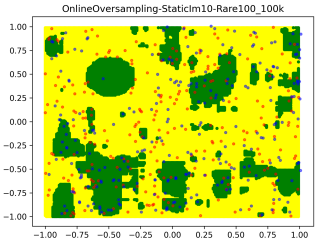
From this analysis, it has been shown that SMOClust managed to forget the pre-drift concept and adapt to drift leading to 100% rare minority class examples and more robust to false-positive drift detections than other approaches in the two dimensional StaticIm10_Rare100 stream. However, the experiment with the five-dimensional StaticIm10_Rare100 stream presents different results (Table 6.10). It shows that SMOClust only performed better than $\text{OnlineOversampling}_\text{d}$ but worse or similar to most other approaches. One potential reason is the fact that two-dimensional space is more compact than five-dimensional space, the rare minority class examples have a lot less space to randomly spawn, which means they



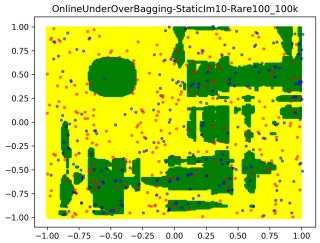
(a) OOB



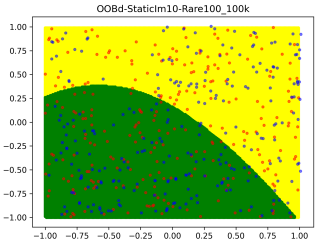
(b) UOB



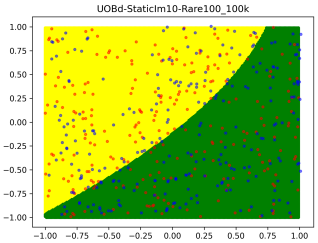
(c) OnlineOversampling



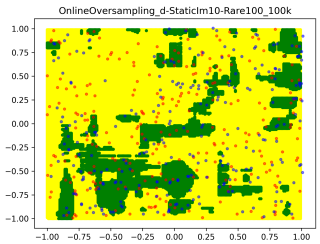
(d) OnlineUnderOverBagging



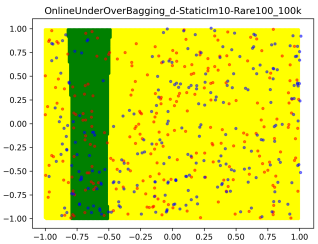
(e) OOB_d



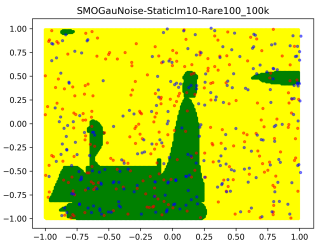
(f) UOB_d



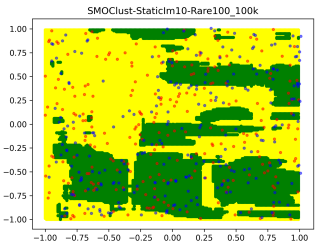
(g) $\text{OnlineOversampling}_\text{d}$



(h) $\text{OnlineUnderOverBagging}_\text{d}$

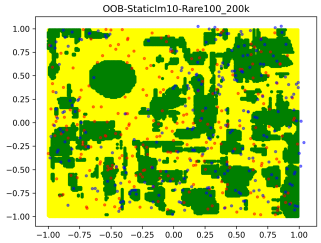


(i) SMOGauNoise

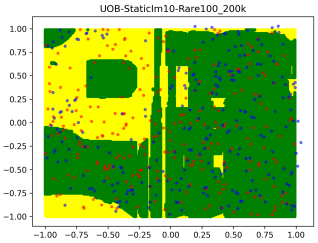


(j) SMOClust

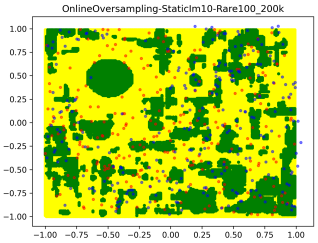
Figure 6.10: Decision Areas Against Class Balanced Test Set at 100k Time Steps (After Drift) of Two Dimensional StaticIm10_Rare100



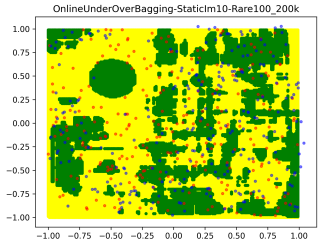
(a) OOB



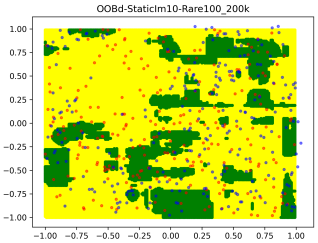
(b) UOB



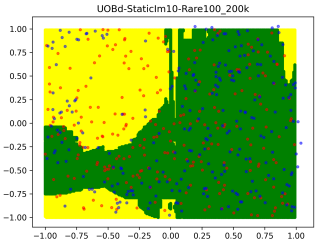
(c) OnlineOversampling



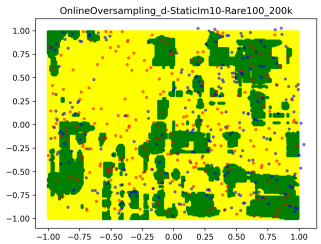
(d) OnlineUnderOverBagging



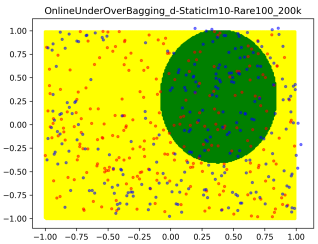
(e) OOB_d



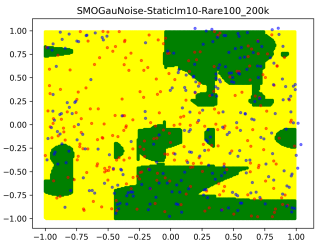
(f) UOB_d



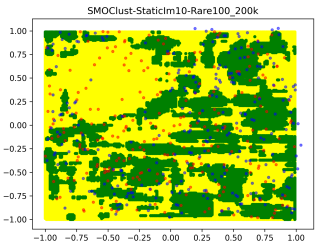
(g) $\text{OnlineOversampling}_\text{d}$



(h) $\text{OnlineUnderOverBagging}_\text{d}$



(i) SMOGauNoise



(j) SMOClust

Figure 6.11: Decision Areas Against Class Balanced Test Set at 200k Time Steps (End of Stream) of Two Dimensional StaticIm10_Rare100

are likely to spawn at the locations that had already been learnt and covered by SMOClust using micro-clusters. Therefore, SMOClust can predict their class label correctly. However, five-dimensional space is sparser than two-dimensional space, meaning that new rare minority class examples are less likely to spawn at previous locations. Therefore, SMOClust struggled to make correct predictions to new rare minority class examples. Another potential reason is that the stream clustering method may be less effective in data streams with more dimensions. For example, it may create some minority class micro-clusters that overlap with the majority class region because of the sparsity of the feature space. Therefore, the aforementioned advantage of SMOClust in dealing with drift could not be manifested. Anyhow, future work is needed to further confirm whether SMOClust tends to perform better in data streams with fewer dimensions.

Short Summary: This analysis shows that SMOClust managed to adapt to concept drift leading to 100% rare minority class examples and was robust to multiple drift detection during gradual drift as well as false-positive drift detections when the data stream has only two dimensions. However, the experiments with the corresponding five-dimensional stream present a different set of results, as the stream clustering methods used by SMOClust might not perform well when the data stream has more dimensions.

6.3.2 Results with Real-world Data Streams

This section presents the analysis done to compare the predictive performance of SMOClust against nine existing approaches in real-world data streams. Experiments with real-world data streams allow us to obtain a general idea of SMOClust's predictive performance in practical applications, where the class imbalance ratio, the position and the type of the concept drifts are unknown. Table 6.18 presents the Friedman rankings of approaches' G-Mean on real-world data streams group by factors.

Table 6.18 shows that the overall top ranked approaches on real-world data streams

Table 6.18: Statistical (Friedman) Ranking of prequential G-Mean on Real-World Streams Grouped by Factors

Groups	OOB	UOB	oOS	oUnder-OverB	OOB_d	UOB_d	oOS_d	oUnder-$OverB_d$	SMO-GauNoise	SMOClust
Luxumbourg	5.1	7.9	2.0333	6.6333	5.1	9.3	2.0333	6.6333	2.7667	7.5
NOAA	1.6667	4.9	4.1	1.6667	5.2	7.0333	8.7667	5.9667	10.0	5.7
Ozone	2.4333	2.2167	8.25	3.0333	5.1	2.3167	8.0167	6.5333	9.7	7.4
PAKDD2009	4.9	2.4333	1.2	5.4667	7.0	5.9667	2.3667	6.6667	9.0	10.0
Covtype	2.2524	5.6286	7.8524	2.7095	4.581	4.4429	9.7	5.3905	8.5	3.9429
INSECTS	4.9133	8.3933	6.6	5.5667	1.2333	7.58	3.4	2.4533	5.0	9.86
All	3.4021	6.176	6.4719	3.975	3.7896	5.851	6.6302	4.7375	7.2479	6.7188

- The p-values of Friedman tests are all $\leq$2.2E-16.
- Highlighted ranks denote significant superior performance.
- Underlined ranks denote the corresponding approach's performance have no statistical significance with SMOClust.

are OOB, OOB_d and OnlineUnderOverBagging whereas SMOClust usually achieved low rankings. SMOClust only achieved a relatively better ranking in Covtype streams than in other streams. Considering all real-world data streams, SMOClust performed similarly to UOB, OnlineOversampling, $OnlineOversampling_d$ and SMOGauNoise. Following the analysis method in Section 6.3.1, we also compared the thirty runs average prequential G-Mean of the approaches on each real-world data stream and reported the A12 effect size comparison results in Table 6.19 to further evaluate SMOClust's predictive performance in real-world data streams.

Table 6.19 shows that SMOClust usually performed similar or better than other approaches in NOAA and Covtype streams while it performed worse than other approaches in Ozone, PAKDD2009 and INSECT streams. Recalling the discussion in Section 6.2.1 on Table 6.4 about the estimated characteristics of the real-world streams, NOAA and Covtype streams mainly consist of safe and borderline minority class examples with different movements of minority class clusters. Also, the minority class ratios throughout Covtype streams

Table 6.19: 30 Runs Average Prequential G-Mean on Real-World Data Streams (A12 SMOClust vs Others)

Groups	OOB	UOB	oOS	oUnderOverB	OOB_d	UOB_d	oOS_d	$oUnderOverB_d$	SMOGauNoise	SMOClust
Luxembourg	93.29%[b]	91.58%[*]	95.15%[b]	92.49%[s]	93.29%[b]	91.09%[-m]	95.15%[b]	92.49%[s]	94.6%[b]	91.76%
NOAA	71.44%[b]	69.47%[s]	70.07%[b]	71.39%[b]	69.64%[*]	68.28%[-b]	65.85%[-b]	69.34%[*]	63.4%[-b]	69.52%
Ozone	65.7%[b]	66.49%[b]	54.03%[-m]	65.58%[b]	60.92%[b]	66.44%[b]	54.03%[-m]	57.12%[b]	49.17%[-b]	54.66%
PAKDD2009	50.84%[b]	56.8%[b]	57.91%[b]	49.64%[b]	47.56%[b]	49.46%[b]	57.2%[b]	47.8%[b]	20.84%[b]	9.36%
$Covtype_{(c_1=\{1\text{-}6\})}$	91.49%[b]	88.55%[-b]	88.93%[-b]	90.11%[-b]	90.81%[b]	88.91%[-b]	86.84%[-b]	89.19%[-b]	85.47%[-b]	90.52%
$Covtype_{(c_1=1)}$	90.59%[b]	88.1%[-b]	88.51%[-b]	88.91%[-b]	89.89%[-b]	90.82%[b]	86.2%[-b]	89.97%[*]	85.0%[-b]	90.01%
$Covtype_{(c_1=2)}$	67.14%[b]	65.32%[-b]	60.45%[-b]	67.08%[b]	64.99%[-b]	66.45%[b]	47.65%[-b]	64.16%[-b]	61.38%[-b]	65.56%
$Covtype_{(c_1=3)}$	56.88%[-m]	60.58%[b]	52.02%[-b]	57.66%[b]	45.26%[-b]	56.63%[-b]	37.73%[-b]	51.46%[-b]	55.49%[-b]	57.23%
$Covtype_{(c_1=4)}$	90.05%[-s]	84.73%[-b]	71.16%[-b]	90.95%[b]	91.31%[b]	84.67%[-b]	65.71%[-b]	88.47%[-b]	79.16%[-b]	90.12%
$Covtype_{(c_1=5)}$	65.98%[b]	63.22%[-b]	57.8%[-b]	65.74%[b]	62.93%[-b]	64.7%[m]	51.36%[-b]	61.95%[-b]	57.47%[-b]	64.64%
$Covtype_{(c_1=6)}$	68.76%[b]	67.44%[s]	64.53%[-b]	69.01%[b]	68.65%[b]	67.52%[b]	50.34%[-b]	68.84%[b]	62.78%[-b]	67.4%
$INSECTS^{inc.}$	74.91%[b]	73.22%[b]	74.8%[b]	74.95%[b]	75.33%[b]	71.2%[b]	74.87%[b]	74.77%[b]	63.31%[-b]	65.09%
$INSECTS_{abr.}$	73.21%[b]	70.56%[b]	71.65%[b]	72.91%[b]	75.98%[b]	72.91%[b]	74.76%[b]	75.68%[b]	73.54%[b]	52.0%
$INSECTS^{inc.}_{grad.}$	76.96%[b]	74.43%[b]	77.19%[b]	76.78%[b]	77.93%[b]	72.73%[b]	77.53%[b]	78.14%[b]	77.65%[b]	65.0%
$INSECTS^{inc.}_{abr.\ re.}$	72.11%[b]	70.31%[b]	71.07%[b]	71.81%[b]	74.78%[b]	71.23%[b]	73.35%[b]	74.01%[b]	73.29%[b]	64.09%
$INSECTS^{inc.}_{re.}$	72.75%[b]	70.64%[b]	71.78%[b]	72.49%[b]	75.0%[b]	71.67%[b]	73.88%[b]	74.4%[b]	73.49%[b]	64.51%

- Based on the average G-Mean, cells are highlighted in lime / light grey when SMOClust performed better than the corresponding approach and cells are highlighted in orange / dark grey cells when SMOClust performed worse than the corresponding approach. The colour intensity scales with the absolute difference of average G-Mean between the SMOClust and the approach of the column and the intensity reaches the maximum when such a difference is $\geq 10\%$.
- Symbols [*], [s], [m] and [b] represent insignificant, small, medium and large A12 effect size against SMOClust. Presence/absence of the sign "-" in the effect size means that the corresponding approach was worse/better than SMOClust.

are usually very low (except Covtype$_{(c_1=\{1\text{-}6\})}$ and Covtype$_{(c_1=1)}$). As discussed in Section 6.3.1, these are the characteristics of data stream that SMOClust is likely to perform similar or better than other approaches, especially when the class imbalance ratio is severe, such as Covtype$_{(c_1=3)}$ stream. Thus, we can see from 6.19 that the rows of NOAA and Covtype streams mainly consist of vivid lime cells and pale orange cells.

On the other hand, Table 6.4 shows that Ozone, PAKDD2009 and INSECT streams consist of large proportions of rare and outlier minority class examples. Based on the discussion in Section 6.3.1.2, SMOClust could not handle rare and outlier minority class examples very well, except when the dimensionality of the input attribute space was low or compact. Thus, it is not surprise to see a lot of orange cells on these data streams.

To summarise the result of experiments with real-world data streams, it has been shown that the advantage of the proposed synthetic minority class oversampling strategy in SMOClust is manifested in severely class imbalanced data streams with high proportions of safe and borderline minority class examples and concept drifts of different movements of minority class clusters. On the downside, SMOClust could not handle rare and outlier minority class examples very well. One potential reason is that the proposed method focuses on synthesising minority class examples in safe and borderline areas. Thus, when the proportions of rare and outlier become dominant, the synthetic examples generated by the proposed method become less reliable. These findings are consistent with result of experiments with artificial data streams.

6.4 Summary and Discussion

The main contribution of this chapter is the proposed approach called Synthetic Minority Oversampling based on Stream Clustering (SMOClust) . The main novelty of this approach

is the stream clustering based method to produce synthetic minority class examples for oversampling (RQ3.1). This method helps the learning system to strategically explore the decision areas of the minority class and to consolidate the knowledge in the most recent seen minority class area (RQ3.1). At the same time, SMOClust can also employ a drift detector to deal with concept drift explicitly. This study answers the RQ3 posed in Section 1.2.3: "How to produce synthetic examples that enable learning the decision boundaries better in class imbalanced stream learning with concept drifts? When and why are they helpful?"

To evaluate the performance and the characteristics of SMOClust, experiments with artificial data streams concerning different types of concept drift difficulties are performed. The results show that SMOClust performed particularly well in severely class imbalanced data streams with high proportions of safe and borderline minority class examples (RQ3.2). It also handles concept drifts of different movements of minority class clusters better than other existing approaches (RQ3.2). However, when the data stream presents high proportions of rare and outlier minority class examples, SMOClust becomes disadvantageous (RQ3.3). To further understand the reason behind the experiment results on artificial data streams, additionally experiments with a few artificial data stream with lower dimensionality are performed. However, it shows that SMOClust could handle rare minority class examples better than other approaches in these cases. This indicates that the reason for SMOClust could not handle rare cases very well on the previous set of experiments could partially and potentially because of the dimensionality of the data streams. In other words, SMOClust may be more advantageous when the dimensionality of the data stream is low. One potential reason related to the dimensionality is that, although the proposed method focuses on synthesising examples in the dense area of minority class micro-clusters, this area is still parse in higher dimensional space. Hence, the synthetic examples tends to be less similar to the actual minority class examples from the data streams than the synthetic examples generated in lower dimensional space.

Lastly, we validated the performance of SMOClust on different real-world data streams. To facilitate the analysis to the experiment results of this part of study, we also estimated the characteristics of the real-world data streams, following the procedure adopted by Brzezinski, Minku, et al. (2021). Based on the estimated characteristics and the experiment results, we concluded that the SMOClust behaved similarly to the experiments with artificial data streams (RQ3.3).

In the future, the idea of SMOClust could be further improved by adding strategies to handle rare and outlier minority class examples better. Also, investigation on how to improve the performance of SMOClust on less severe drifting class imbalanced data streams can also be considered as part of future work.

Chapter Seven

Conclusion and Future Work

This chapter summarises the contributions of this thesis and discusses the potential direction of the future work. The main contributions are the answers to the research questions 1, 2 and 3, which are proposed in Chapters 4, 5 and 6, respectively.

Overall, this thesis tackles the joint challenge of concept drift and class imbalance in data stream learning in two directions: exploiting past knowledge and create synthetic minority class examples. In the first direction, the idea is to exploit relevant past knowledge to help learning the concept after the drift, especially the new minority class concept in class imbalanced data stream learning. While past knowledge may not always exist in the system's memory, the second direction aims to complement these situations where the first direction could not address. The idea is to create synthetic minority class examples to help in learning the minority class concept, especially after the concept drift, without the aid from the past knowledge.

Due to the gaps in the literature of memory strategies for dealing with concept drifts, Chapter 4 first investigates using diversity as a memory framework to deal with concept drift in class balanced data streams. Chapter 5 then further investigates this idea to deal with concept drift in class imbalanced data streams. Lastly, Chapter 6 investigates the latter

direction of creating synthetic minority class examples for learning drifting class imbalanced data streams.

Chapter 4 presented two studies to answer research question 1: "Can diversity be used to create a memory-based framework able to tackle different types of drifts, including recurrent, abrupt, gradual, severe and mild-severe drifts more efficiently and effectively? How well does this framework perform?" The first preliminary study showed that diversity can be used as a memory management strategy to maintain a diverse set of past models in the system's memory. This optimises the chances of having relevant past knowledge to exploit, preparing the data stream learning approach in dealing with different types of concept drift, including recurrent, abrupt, gradual, severe and mild-severe drifts.

The second study in Chapter 4 brought the idea of diversity further to memory recovery. It proposed a novel diversity-memory based framework that can deal with recurrent, abrupt, gradual, severe and mild-severe drifts and is robust to false-positive drift detections, called Concept Drift Handling Based on Clustering in the Model Space (CDCMS). The analysis showed that CDCMS is good at dealing with abrupt drift and recurrent drift. Yet, it could be disadvantageous when the data stream presents several extremely severe gradual drifts where intermediate concepts could form. Experiments with several artificial and real-world data streams showed that CDCMS usually performed similar or better than existing approaches from the literature.

Chapter 5 proposed novel memory strategies for class imbalanced learning which are incorporated into CDCMS to become a diversity-based framework for class imbalanced learning, called Concept Drift Handling Based on Clustering in the Model Space for Class Imbalanced Learning (CDCMS.CIL). It answered the research question 2: "How can memory strategies be used to tackle class imbalanced data streams with concept drifts? When and why are they helpful?" The analysis showed that the proposed strategy is effective in recover-

ing relevant knowledge to deal with concept drifts, especially recurrent drift and drift leading to concept that is similar to one of previous concepts. Such effectiveness is unaffected by the class imbalance degree of the data stream because the proposed memory strategies rely on a class balanced synthetic data set that is created by using the micro-clusters associated to the base learners.

Further analysis showed that class imbalance strategy is still required to enable the proposed memory strategies to work well because their effectiveness in dealing with drift also depends on whether the past models have bias towards the majority class. Experiments with artificial and real-world data streams showed that CDCMS.CIL performed similar or better than five existing approach from the literature and its predecessor CDCMS because it managed to exploit relevant past knowledge to deal with concept drift and usually robust to false-positive drift detections despite the data stream is class imbalanced.

Finally, Chapter 6 proposed a novel approach called Synthetic Minority Oversampling based on Stream Clustering (SMOClust) to answer the research question 3: "How to produce synthetic examples that enable learning the decision boundaries better in class imbalanced stream learning with concept drifts? When and why are they helpful?" In the analysis, SMOClust was shown to be good at handling severely class imbalanced data streams with high proportions of safe and borderline minority class examples. It can also adapt to concept drift of different movements of minority class sub-clusters. These results thank to the proposed synthetic minority class oversampling strategy based on stream clustering, being robust to incremental and gradual drifts as well as false-positive drift detections. Experiments with artificial and real-world data streams showed that SMOClust usually performed similar or better than existing approaches from the literature when the data stream presents the aforementioned characteristics.

Over the next sections, the contributions of Chapters 4, 5 and 6 and their future work

are explained in more detail.

7.1 Diversity-based Memory Strategies for Concept Drift Adaptation in Class Balanced Data Streams

7.1.1 Contributions

Exploiting past knowledge is an intuitive strategy to help learning the concept after the drift. Some existing approaches, such as DDD (Minku and Yao, 2012) and RCD (Gonçalves Jr and Barros, 2013), explicitly apply this idea to adapt to drifts. However, these approaches only investigate how to utilise the current knowledge in the system to adapt to drifts. None of the existing work investigates how to maintain past knowledge in the system's memory in order to maximise the chance of having relevant knowledge to exploit.

Chapter 4 first presented a preliminary study to investigate the possibility of using diversity as a memory management strategy. It then further extended this idea and proposed a memory-based framework to deal with recurrent, abrupt, gradual, severe and mild-severe drifts in class balanced data stream learning, which is called Concept Drift Handling Based on Clustering in the Model Space (CDCMS). It answered the research question 1: "Can diversity be used to create a memory-based framework able to tackle different types of drifts, including recurrent, abrupt, gradual, severe and mild-severe drifts more efficiently and effectively? How well does this framework perform?" Note that the studies in Chapter 4 only concerned class balanced data stream learning, because we wanted to first investigate the proposed strategies in dealing with concept drift without being affected by class imbalance.

First, the preliminary study showed that, diversity-based memory management strat-

egy is more advantageous than other memory management strategies, such as First In First Out (FIFO), Least Recently Used (LRU), and Elitism because it can maintain various past knowledge, preparing the system to adapt to recurrent drift and drift leading to a concept that is similar to one of the previous concepts. However, its impact to the predictive performance of the data stream learning approach is not straightforward because the predictive performance also depends on the effectiveness of the memory recovery strategy.

In the second study, a novel diversity-memory based framework, called CDCMS, was proposed. This framework creates ensembles with different diversity levels through the proposed clustering in the model space strategy. The analysis shows that this strategy is effective in identifying recurring concepts and recovering relevant past knowledge to handle different types of concept drift, including recurrent, abrupt, gradual, severe and mild-severe drifts. Also, the highly diverse ensemble created based on this strategy helped to prevent significant drops in predictive performance after concept drifts. Experiments with artificial and real-world data streams showed that CDCMS performed similar or better than the compared approaches from the literature. It performed particularly well in dealing with abrupt and recurrent drifts, in terms of both overall predictive performance and recovery from drifts. However, the experiments also showed that CDCMS could be disadvantageous in data streams presenting multiple very severe gradual drifts because intermediate concepts could form during the transitional period. When this happens, the proposed clustering in the model space strategy, sometimes, considered that the concept drift leads to the recurrence of the intermediate concept, thus, recovered past knowledge that is irrelevant to the post-drift concept.

An additional contribution of Chapter 4 is that, we also incorporated a strategy which periodically replaces the least performing ensemble member with a new base learner. This strategy is to deal with drift that could not be captured by the drift detector. It also helps to fill up the memory quickly so that the approach holds a good number of past models for the

clustering in the model space strategy. This strategy is inspired from the OAUE (Brzeziński and Stefanowski, 2014), where the weighting metric was replaced by prequential accuracy to better reflect the performance of the ensemble members at the current time steps.

7.1.2 Future Work

Despite the study in Chapter 4 showing that CDCMS can handle recurrent, abrupt, gradual, severe and mild-severe drifts, it would sometimes suffer from very severe gradual concept drift where intermediate concept could form. Therefore, future work is needed to distinguish intermediate concepts from the "true" concepts of the data streams. For example, we could devise a drift detector that can estimate the starting and ending points of the drift, then notify CDCMS to perform the clustering in the model space strategy only when the drift is complete.

Besides, the clustering in the model space strategy, proposed in Section 4.2.1.1, is based on the prediction correctness of the base learners to the most window of recent examples, where the batch size is a hyper-parameter of CDCMS. When the batch size is large, the dimensionality of the "clustering examples" becomes large. This means that CDCMS will cluster the model space based on a very high dimensional "clustering examples". This potentially weakens the ability of the clustering in the model space strategy in grouping models with similar knowledge, thus weakening its effectiveness in identifying recurring concepts and recover relevant past knowledge. However, the dilemma here is that the size of the most recent window of examples could not be too small. Otherwise, this window may not contain enough examples to approximate the concept after the drift. As the result, the effectiveness of the clustering in the model space strategy in identifying recurring concepts and recover relevant past knowledge is also negatively affected. Therefore, a future work is to improve the clustering in the model space strategy, such that it does not need to rely on

a high dimensional "clustering examples".

Furthermore, CDCMS creates a highly diverse ensemble to prevent a significant drop in performance right after the drift. This ensemble is created by joining models with distinct concepts. However, this ensemble only participates in making prediction but not training. Meanwhile, in the work of DDD, Minku and Yao (2012) show that an ensemble trained with high diversity on the past concept could be able to converge to the concept after the drift. Therefore, a future work can further investigate if a highly diverse ensemble of models holding distinct concepts could also converge to a single concept, leading to further improvements in predictive performance.

Another direction is about the memory management strategy. Although the study in Chapter 4 shows that diversity is advantageous in most cases, the current method in deciding which model to discard between similar two models is too simple, in which it just discards the model trained with less examples. However, this model may have acquired some unique characteristics of the concept which are potentially useful for adapting future concept drifts. Therefore, as a future work, a new memory management strategy can be devised to not only consider the diversity in the memory, but also consider the uniqueness of the knowledge acquired by the base learners.

7.2 Diversity-based Framework for Concept Drift Adaptation in Class Imbalanced Data Streams

7.2.1 Contributions

Chapter 4 showed that diversity can be used to create a memory-based framework, dealing with recurrent, abrupt, gradual, severe and mild-severe drifts. In particular, it showed that exploiting past knowledge to help adapting to concept drift is effective. This idea could be particularly helpful in adapting to concept drift in class imbalanced data streams because there exists particularly few examples from the minority class after the drift. Therefore, Chapter 5 brought the idea of the diversity-memory based framework to deal with concept drift in class imbalanced data stream learning. However, the proposed memory strategies in Chapter 4 could not be applied to class imbalanced data stream learning directly because they rely on the most recent batch of examples, which is likely class imbalanced as well. To solve this issue, Chapter 5 proposed a new set of memory strategies which leverage the ability of stream clustering methods to track the decision areas learnt by each base learner. It answered the research question 2: "How can memory strategies be used to tackle class imbalanced data streams with concept drifts? When and why are they helpful?" A memory-based framework was proposed by incorporating the new memory strategies into CDCMS . It is called Concept Drift Handling Based on Clustering in the Model Space for Class Imbalanced Learning (CDCMS.CIL).

Three studies had been conducted to evaluate CDCMS.CIL. In the first study, the proposed memory strategies was verified that they can maintain and recover relevant past knowledge to deal with different types of concept drift, especially recurrent drift and drift leading to concept that is similar to one of previous concepts. They are unaffected by the class imbalance degree of the data stream because they rely on a class balanced synthetic data

set, created by using the micro-clusters associated to the base learners rather than the most recent batch of examples. However, the effectiveness of recovering predictive performance from drifts depends on whether the base learners had bias towards the majority class of their corresponding concept. Otherwise, the proposed strategies could not help to adapt future concept drift.

Due to this reason, the second study in Chapter 5 investigated the impact of using different resampling methods to the predictive performance of CDCMS.CIL. The analysis showed that CDCMS.CIL works the best with oversampling in most cases, because most of the useful knowledge is retained. On the other hand, CDCMS.CIL could only work well with undersampling when the concept drift leads to novel concepts because the aggressive nature of undersampling maybe more advantageous than oversampling in learning new concepts from scratch. However, CDCMS.CIL could sometimes suffer from early drift detection, especially when the drift is gradual. In such case, CDCMS.CIL tends to recover past models that are relevant to the concept before the drift, as there is insufficient amount of examples from the concept after the drift.

In the last study, we validated the performance of CDCMS.CIL through experiments with artificial and real-world data streams. The results showed that CDCMS.CIL performed similar or better than other existing approaches in most cases, especially in data streams with abrupt and abrupt recurrent drifts where no intermediate concept could form. Thus, CDCMS.CIL managed to recover relevant past knowledge to handle the drift.

7.2.2 Future Work

Chapter 5 showed that CDCMS.CIL could suffer from early drift detection which leads the clustering in the model space strategy for class imbalanced learning to recover past knowledge

that is similar to the concept prior to the drift. Therefore, a future work could be similar to the one mentioned in Section 7.1.2, which is to devise a drift detector that could estimate the starting and ending points of concept drifts in class imbalanced data stream. CDCMS.CIL can then cluster the model space only when a the drift is deemed to complete.

Another direction for future work is to exploit inter-class relevant knowledge. The existing memory recovery strategies, including the proposed clustering in the model space strategy for class imbalance learning, only exploit past knowledge that is relevant to the current concept by class-to-class. For example, a past model could only be considered as relevant to the current class 1 when it had acquired a similar class 1 concept. However, the decision areas of the current class 1 may have been occupied by the class 0 with similar size and shape. In this case, if we could identify them correctly, we could exploit them to aid in learning the current class 1.

Furthermore, the current method of using micro-clusters to compress the knowledge learnt by each base learner gives extra burden to the learning system in terms of computational resources and memory consumption. Despite it enables the proposed memory strategies to work properly in class imbalanced data streams, it does not solve the issue of clustering the model space based on high dimensional "clustering examples". Therefore, in the future work, we could investigate another light-weight method to compress the knowledge learnt by each base learner, avoiding the issue of clustering the model space based on high dimensional "clustering examples". For example, if we could devise a data structure to summarise the statistics of the examples seen by each base learner while taking the location in the feature space into the consideration, then we can cluster the model space based on these statistics.

Other directions of future work include an investigation of the impact of using other advanced resampling methods or class imbalance adaptable base learners to the predictive

performance of CDCMS.CIL.

7.3 Synthetic Minority Oversampling for Learning Drifting Class Imbalanced Data Streams

7.3.1 Contributions

The proposed memory strategies in Chapter 5 was not only intuitive but also effective in dealing with concept drifts in class imbalanced data streams. However, they could only be applied to situations where relevant past knowledge exists in the system's memory. When relevant past knowledge does not exist, strategies are still needed to help learning the new concept, especially the minority class in class imbalanced data stream learning. Therefore, Chapter 6 presented a study following this direction and proposed a novel strategy called Synthetic Minority Oversampling based on Stream Clustering (SMOClust) to answer the research question 3: "How to produce synthetic examples that enable learning the decision boundaries better in class imbalanced stream learning with concept drifts? When and why are they helpful?"

SMOClust employs a stream clustering method per class to maintain sets of micro-clusters by class, where the stream clustering method is required to be concept drift adaptable. SMOClust strategically creates synthetic minority class examples based on the micro-clusters in the same class, which are the abstraction of the minority class examples seen so far. This is the first work to investigate synthetic minority oversampling based on stream clustering. Besides, SMOClust also employs a drift detector as strategy to deal with abrupt drifts explicitly. Therefore, SMOClust is an explicit concept drift adaptation approach for class imbalanced data stream learning. SMOClust is also robust to false-positive drift detec-

tions because only the base learner is reset upon drift detection but not the stream clustering methods. Thus, it can still produce synthetic minority class examples following the current concept for oversampling when false-positive drift detection is given.

Through experiments with both artificial and real-world data streams, SMOClust had been shown to perform better than most compared approaches in extremely class imbalanced data streams presenting high proportions of safe and borderline minority class examples, and concept drifts involving different movements of minority class sub-clusters. Besides, the results also suggested that SMOClust may tend to perform better in data streams with low dimensionality. The main weakness of SMOClust is to handle rare and outlier minority class examples. SMOClust also performed similar or marginally worse than the existing approaches in data stream with less severe class imbalanced ratio.

Among all the compared approaches, UOB is the most competitive against SMOClust. It usually performed similar or better than SMOClust in extremely severe class imbalanced data streams. The main reason is the aggressive nature of undersampling may be advantageous in learning the minority class, especially when the class imbalance ratio is extreme.

7.3.2 Future Work

The results in Chapter 6 showed that SMOClust may perform better in low dimensional data streams than high dimensional data streams. Future work could further confirm this characteristics.

Another direction of future work is to improve SMOClust in handling data streams presenting high proportions of rare and outlier minority class examples, especially in data streams with more dimensions. Also, clustering methods are generally prone to high di-

mensional feature spaces. So, a study on alternatives to stream clustering method is also proposed as future work.

Besides, Chapter 6 also shows that UOB is the most competitive approach to SMOClust, especially in extremely class imbalanced data streams. This shows the potential of undersampling in dealing with extreme cases. Therefore, a study to devise a smart undersampling method for learning drifting class imbalanced data stream can also be a future work. Moreover, if such smart undersampling method could be accomplished, we could potentially combine it with SMOClust, taking the advantages from both worlds to deal with concept drift in class imbalanced data streams.

Finally, SMOClust only concerns how to better learn the minority class without exploiting any past knowledge. It does not store any past model to deal with recurrent drifts and drifts leading to a concept drift that is similar to a previous one neither. Therefore, a study of combining the idea of SMOClust and CDCMS.CIL, and investigating the interaction between different strategies could also be a future work.

Appendix One

Percentage Differences of Sine, Agrawal, SEA and STAGGER Concepts

Table A.1 presents the percentage difference between pairs of concepts used to compose artificial data streams in Chapters 4 and 5. Each percentage difference in Table A.1 is based on one million examples which are sampled uniformly. The calculation of the difference is as follow.

$$diff(f_a, f_b) = \frac{\sum_{i=1}^{n} |y_i^{f_a} - y_i^{f_b}|}{n} \tag{A.1}$$

where $y_i^{f_a}$ and $y_i^{f_b}$ are the class labels determined by the a-th and b-th functions of the generator, respectively, and n is the total number of examples generated uniformly at random to calculate the severity. We consider a concept drift to be severe when the concepts before and after the drift have at least around 50% difference, while around 25% difference is referred to as a mild severity drift.

Table A.1: Percentage Difference of Concepts

	Sine			SEA				STAGGER	
	f1	f2	f3	f1	f2	f3	f4	f1	f2
f2	100.0%	-	-	8.5%	-	-	-	59.3%	-
f3	26.8%	73.2%	-	7.4%	16.0%	-	-	77.8%	48.1%
f4	73.2%	26.8%	100.0%	13.1%	4.6%	20.6%	-	-	-
f5	-	-	-	23.9%	32.5%	16.5%	37.1%	-	-

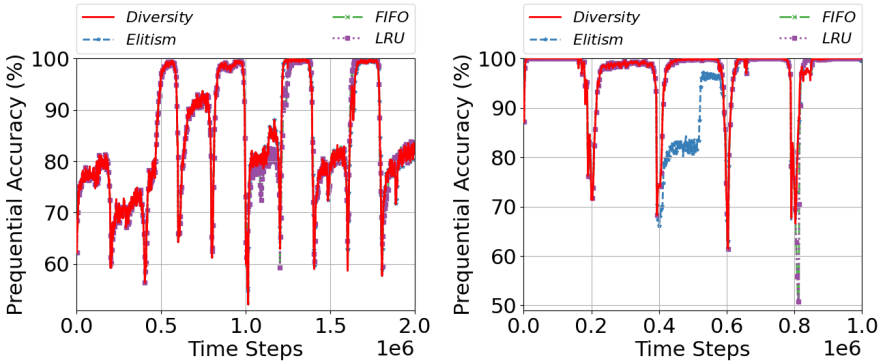
	Agrawal								
	f1	f2	f3	f4	f5	f6	f7	f8	f9
f2	53.9%	-	-	-	-	-	-	-	-
f3	53.1%	50.8%	-	-	-	-	-	-	-
f4	53.9%	20.5%	50.8%	-	-	-	-	-	-
f5	53.4%	47.6%	50.7%	47.7%	-	-	-	-	-
f6	69.9%	28.9%	51.2%	35.5%	48.1%	-	-	-	-
f7	50.5%	53.3%	50.1%	53.5%	60.1%	57.2%	-	-	-
f8	33.5%	60.4%	46.5%	59.6%	59.6%	59.8%	49.8%	-	-
f9	50.4%	53.3%	50.2%	53.5%	59.9%	57.3%	6.0%	49.5%	-
f10	32.9%	61.3%	46.5%	61.3%	60.0%	59.9%	51.1%	1.8%	51.1%

All percentage differences were calculated using Equation A.1 based on one million random generated examples.

Appendix Two

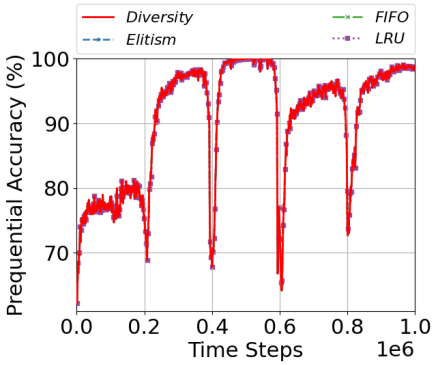
All Performance Plots of DP

This section presents all results regarding the discussion in Section 4.1.3. Figures B.1-B.2 presents the prequential accuracy of DP using different memory strategies (Diversity, FIFO, LRU, and Elitism) over the artificial data streams. Figures B.3-B.4 presents the prequential accuracy of DP and the existing approaches over the artificial data streams.

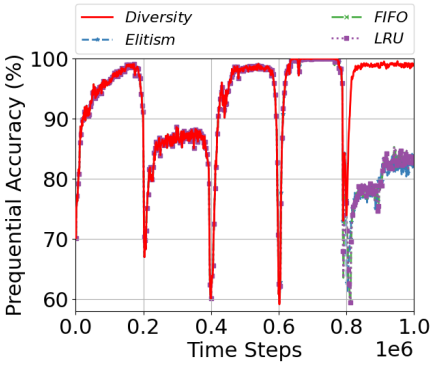


(a) Agrawal1

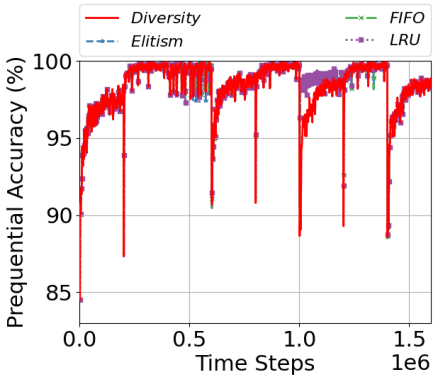
(b) Agrawal2



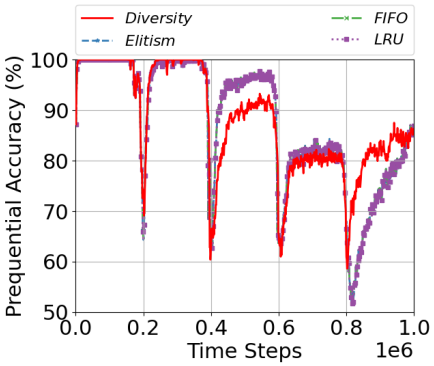
(c) Agrawal3



(d) Agrawal4

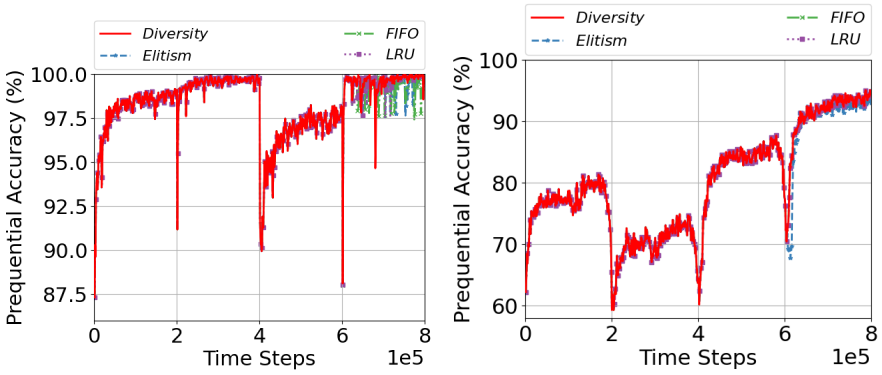


(e) Agrawal5



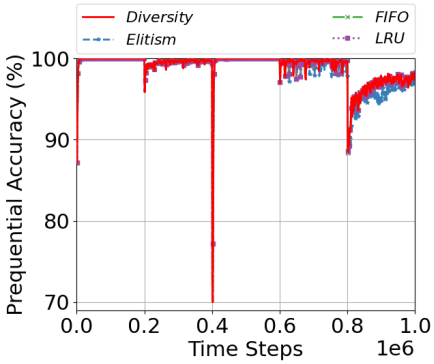
(f) Agrawal6

Figure B.1: Prequential Accuracy Using Different Memory Strategies over Artificial Data Streams (Agrawal1-6)

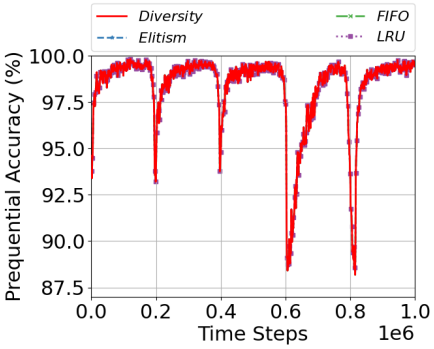


(a) Agrawal7

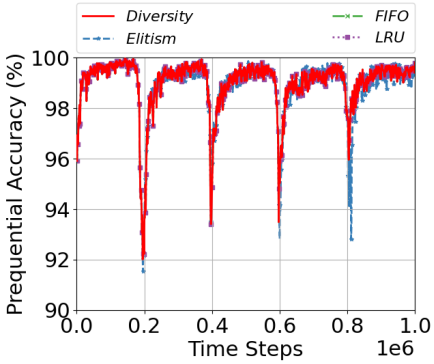
(b) Agrawal8



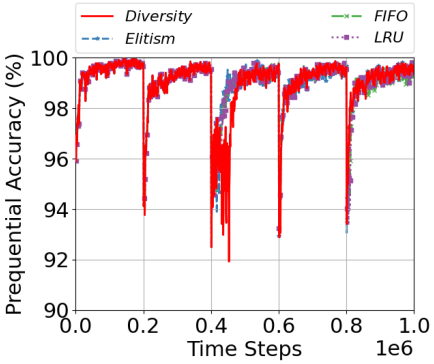
(c) Agrawal9



(d) SEA1

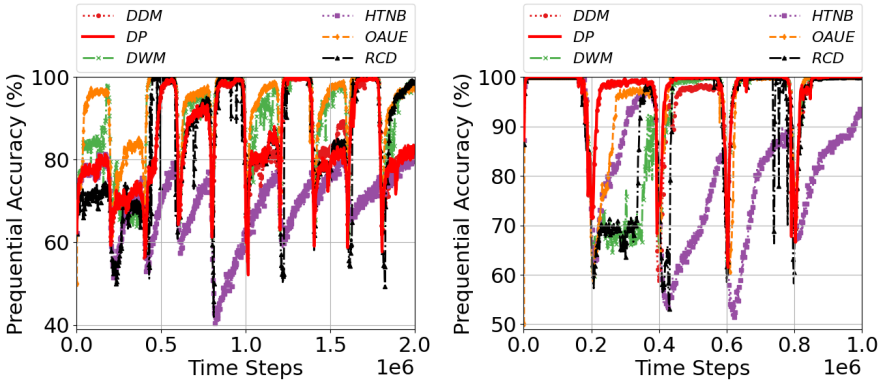


(e) SEA2



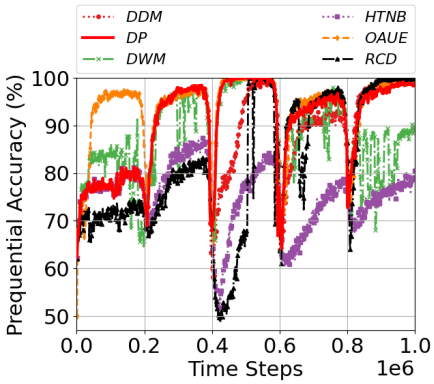
(f) SEA3

Figure B.2: Prequential Accuracy Using Different Memory Strategies over Artificial Data Streams (Agrawal7-9, SEA1-3)

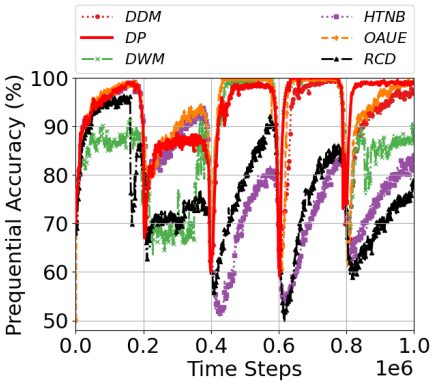


(a) Agrawal1

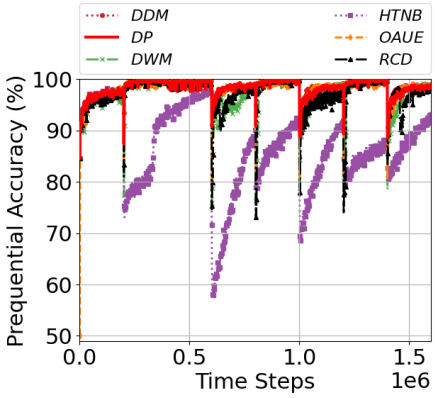
(b) Agrawal2



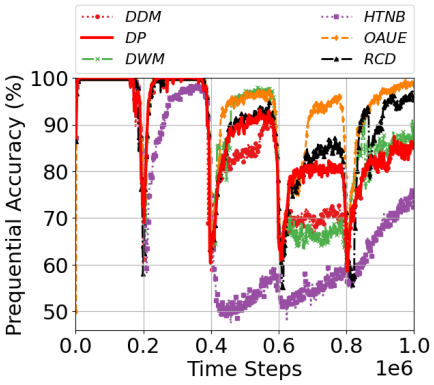
(c) Agrawal3



(d) Agrawal4

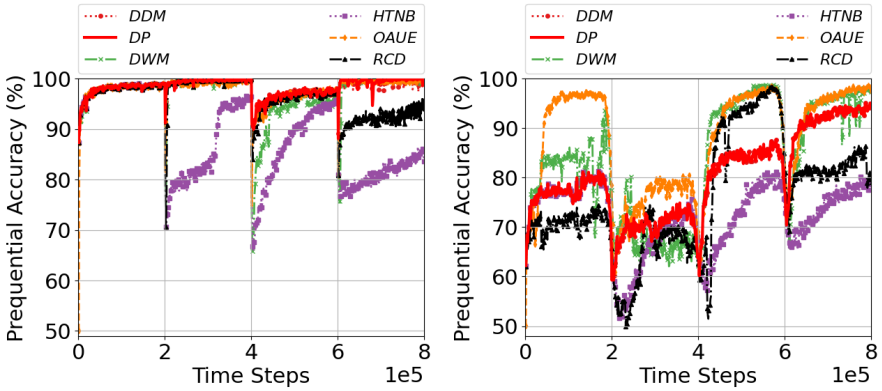


(e) Agrawal5



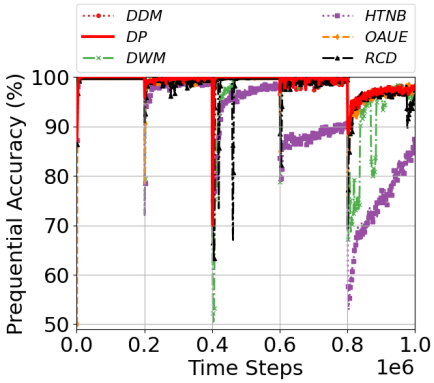
(f) Agrawal6

Figure B.3: Prequential Accuracy of DP and Existing Approaches over Artificial Data Streams (Agrawal1-6)

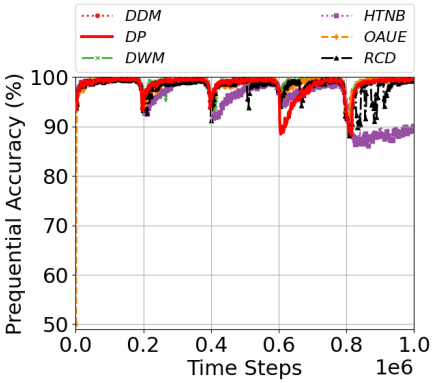


(a) Agrawal7

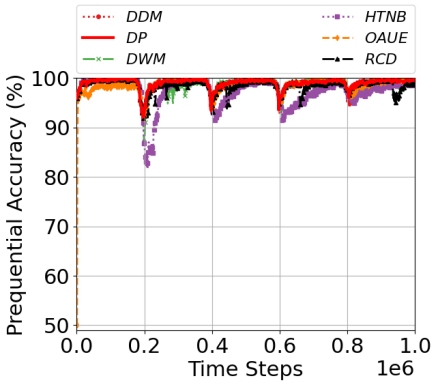
(b) Agrawal8



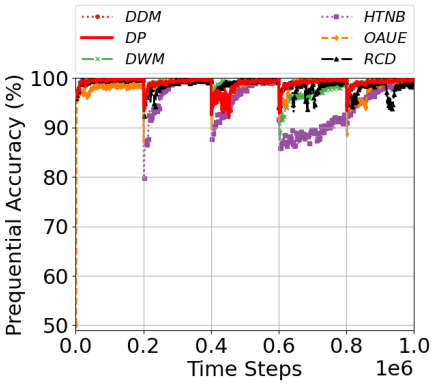
(c) Agrawal9



(d) SEA1



(e) SEA2



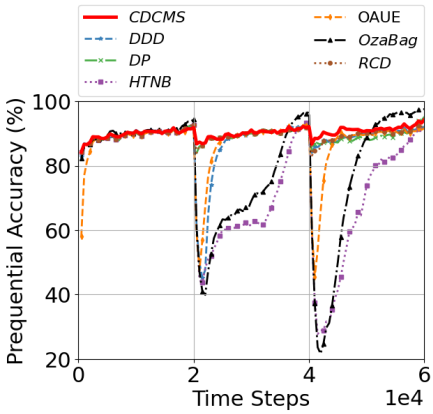
(f) SEA3

Figure B.4: Prequential Accuracy of DP and Existing Approaches over Artificial Data Streams (Agrawal7-9, SEA1-3)

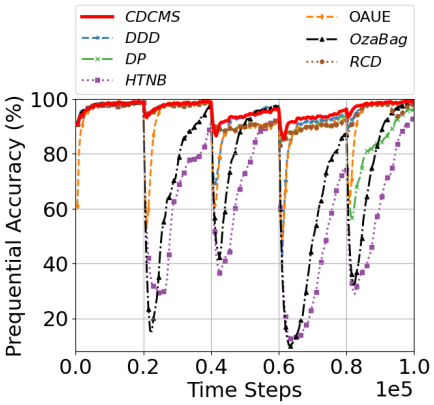
Appendix Three

All Performance Plots of CDCMS

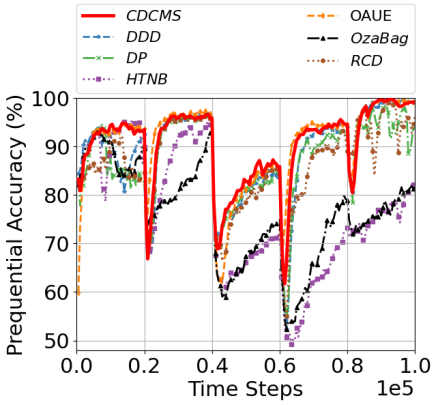
This section presents all results regarding the discussion in Section 4.2.3. Figures C.1-C.2, and C.3-C.4 the prequential accuracy of CDCMS and the existing approaches over the artificial data streams with abrupt and gradual drifts respectively.



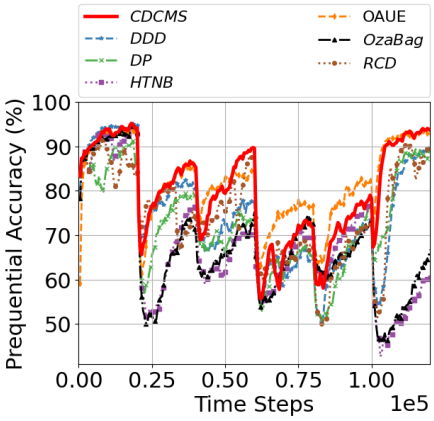
(a) Sine1



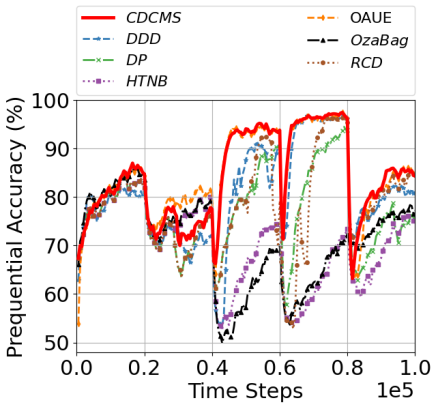
(b) Sine2



(c) Agrawal1

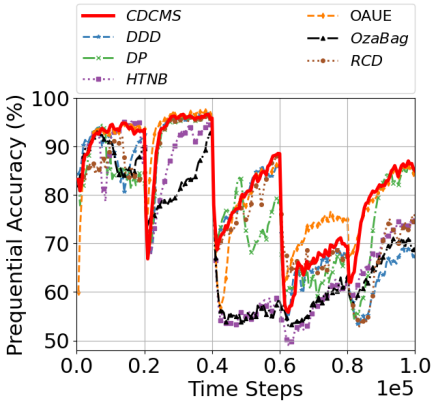


(d) Agrawal2

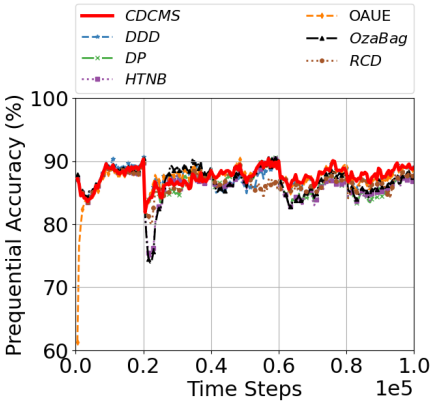


(e) Agrawal3

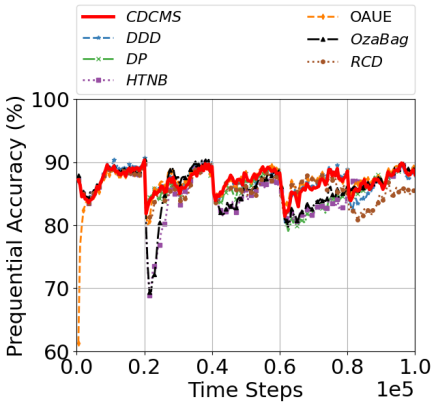
Figure C.1: Prequential Accuracy of CDCMS and Existing Approaches over Artificial Data Streams with Abrupt Drifts (Sine1-2, Agrawal1-3)



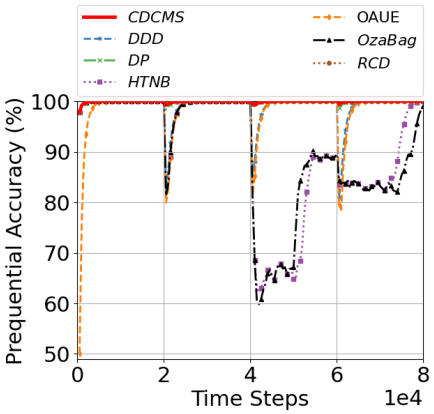
(a) Agrawal4



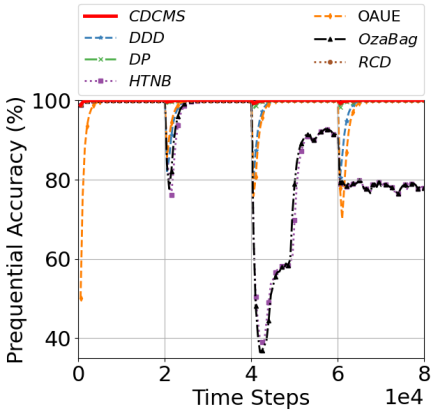
(b) SEA1



(c) SEA2

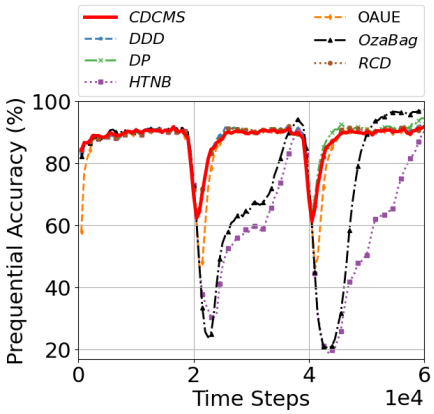


(d) STAGGER1

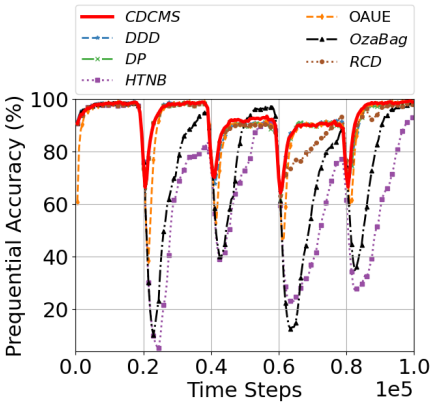


(e) STAGGER1

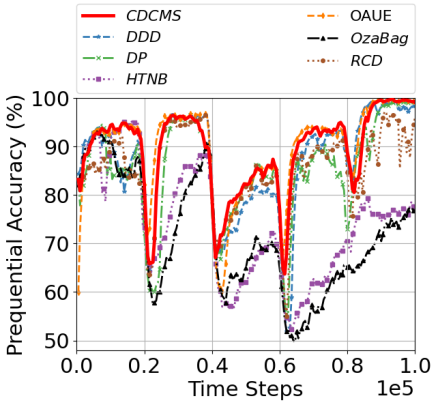
Figure C.2: Prequential Accuracy of CDCMS and Existing Approaches over Artificial Data Streams with Abrupt Drifts (Agrawal4, SEA1-2, STAGGER1-2)



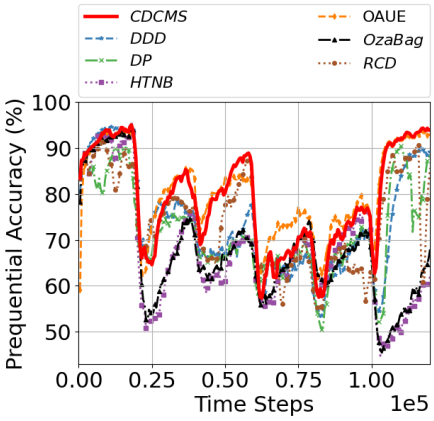
(a) Sine1



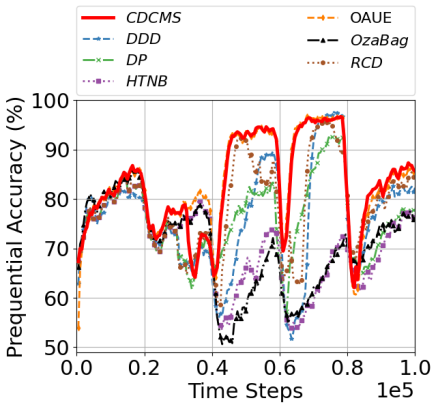
(b) Sine2



(c) Agrawal1

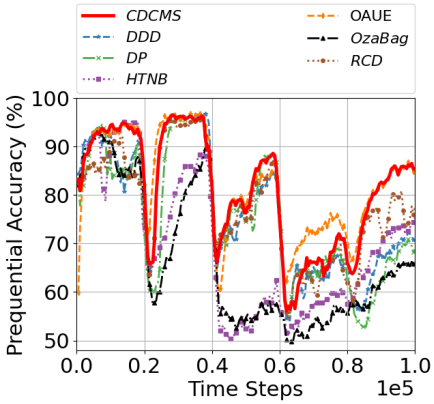


(d) Agrawal2

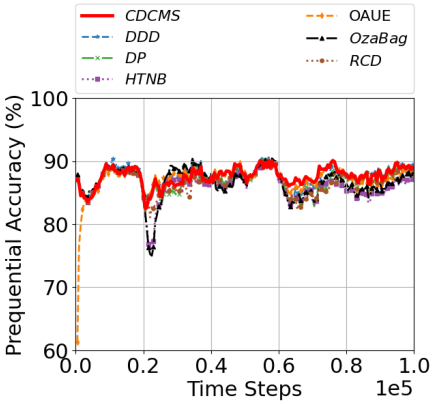


(e) Agrawal3

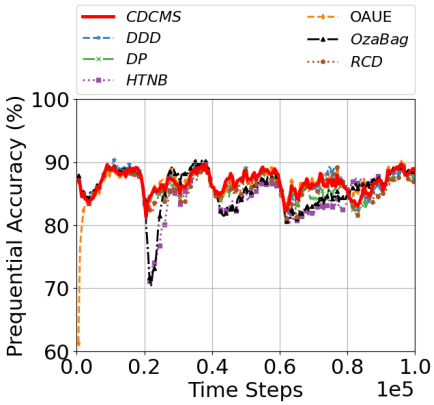
Figure C.3: Prequential Accuracy of CDCMS and Existing Approaches over Artificial Data Streams with Gradual Drifts (Sine1-2, Agrawal1-3)



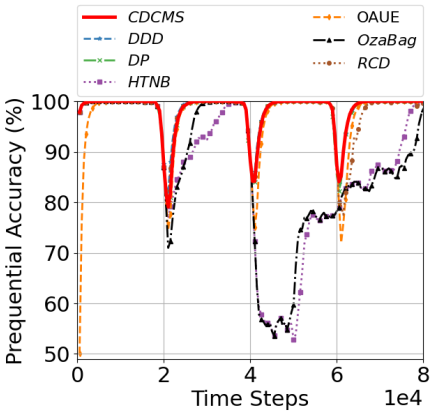
(a) Agrawal4



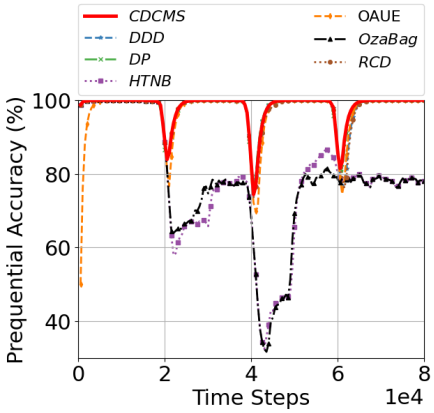
(b) SEA1



(c) SEA2



(d) STAGGER1



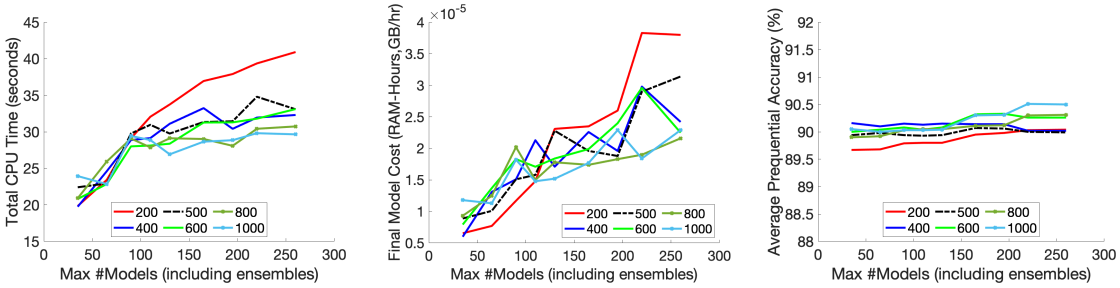
(e) STAGGER1

Figure C.4: Prequential Accuracy of CDCMS and Existing Approaches over Artificial Data Streams with Gradual Drifts (Agrawal4, SEA1-2, STAGGER1-2)

Appendix Four

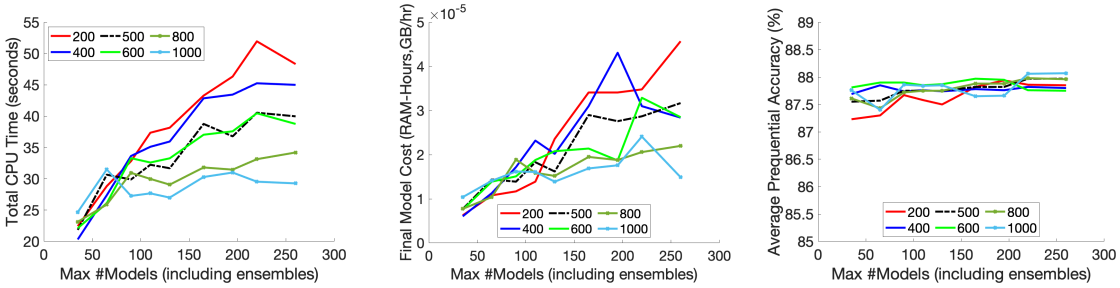
Plots of CDCMS's Time, Memory and Accuracy Relationship on Sine1-abrupt, Sine1-gradual, Sine2-abrupt and Sine2-gradual Streams

This section presents the all results of the empirical study on the time, memory and accuracy relationships of CDCMS. This study is presented in Section 4.2.4.2.



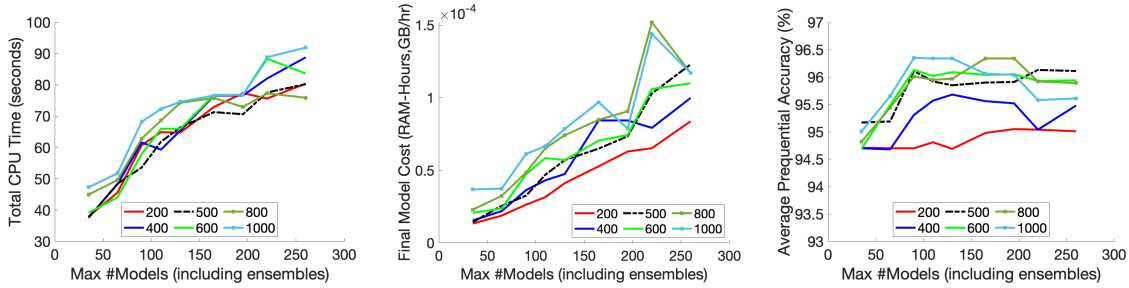
(a) Runtime (b) Memory (c) Accuracy

Figure D.1: Overall runtime, memory and accuracy of CDCMS for different maximum total numbers of models in Sine1-abrupt. The different coloured lines represent different sliding window sizes.



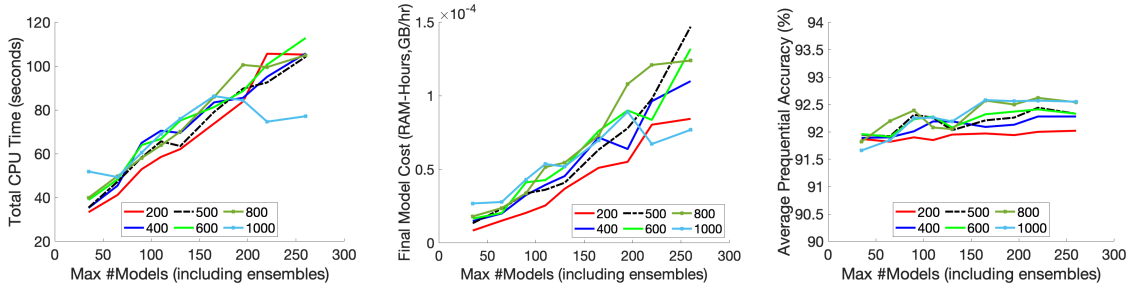
(a) Runtime (b) Memory (c) Accuracy

Figure D.2: Overall runtime, memory and accuracy of CDCMS for different maximum total numbers of models in Sine1-gradual. The different coloured lines represent different sliding window sizes.



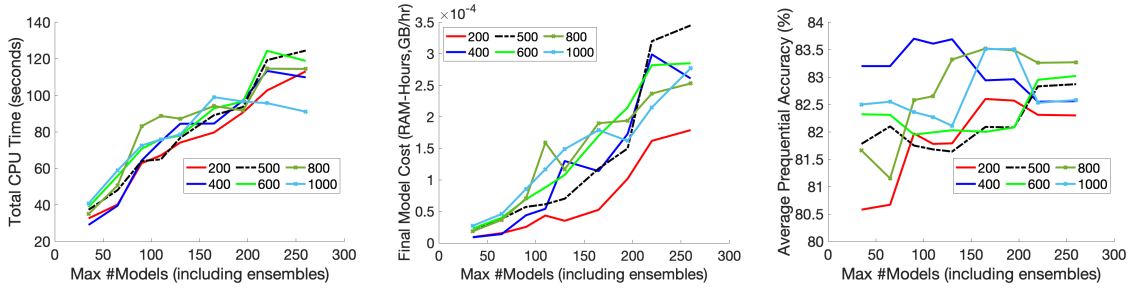
(a) Runtime (b) Memory (c) Accuracy

Figure D.3: Overall runtime, memory and accuracy of CDCMS for different maximum total numbers of models in Sine2-abrupt. The different coloured lines represent different sliding window sizes.



(a) Runtime (b) Memory (c) Accuracy

Figure D.4: Overall runtime, memory and accuracy of CDCMS for different maximum total numbers of models in Sine2-gradual. The different coloured lines represent different sliding window sizes.



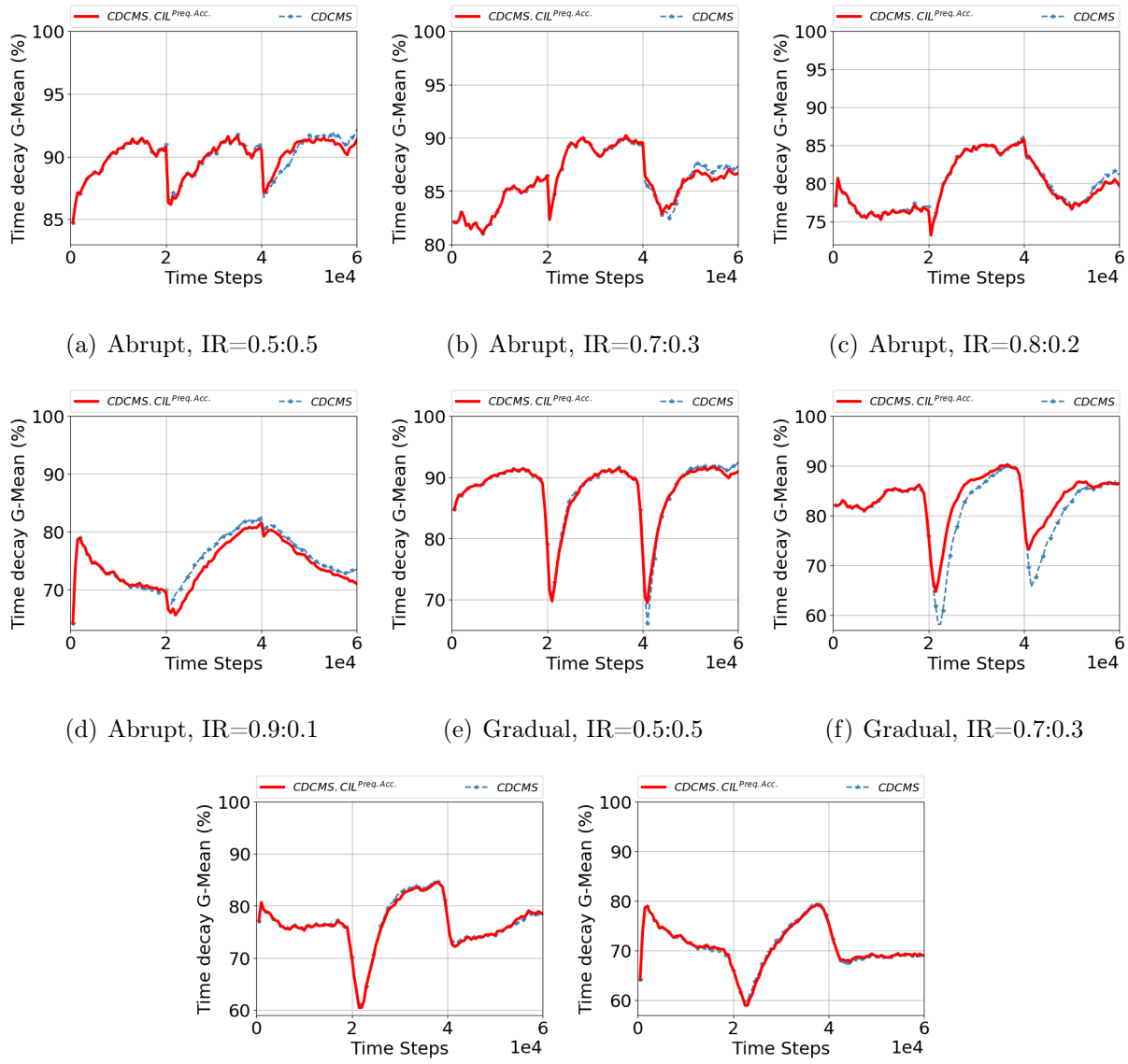
(a) Runtime (b) Memory (c) Accuracy

Figure D.5: Overall runtime, memory and accuracy of CDCMS for different maximum total numbers of models in Agrawal3-gradual. The different coloured lines represent different sliding window sizes.

Appendix Five

All Performance Plots of the Comparison between CDCMS.CIL$^{\text{Preq. Acc.}}$ and CDCMS

This section presents all results of comparing CDCMS.CIL$^{\text{Preq. Acc.}}$ against CDCMS, regarding the discussion in Section 5.3.1. Figures E.1-E.10 present the their time decay G-Mean over the artificial data streams.

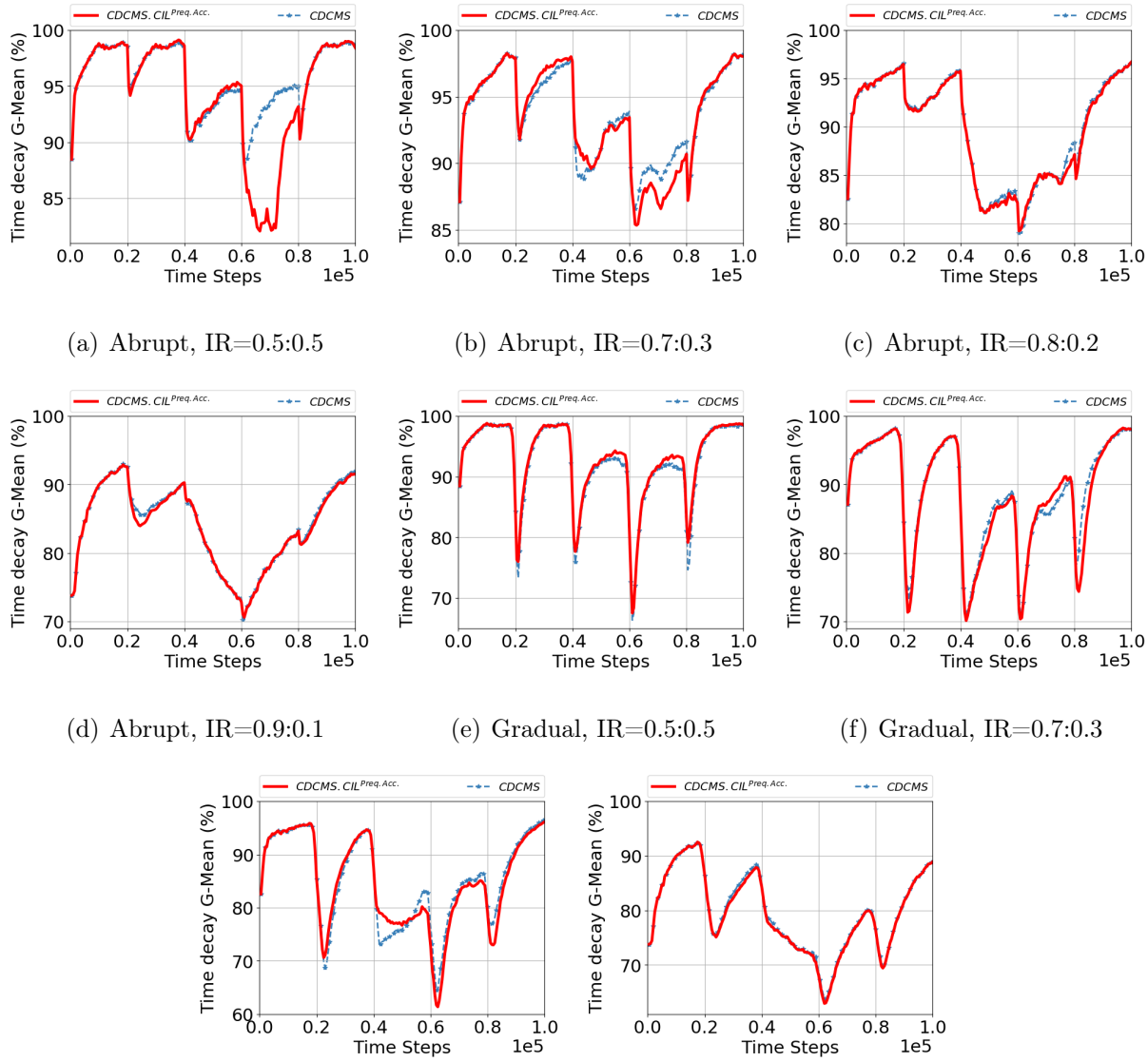


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure E.1: CDCMS.CIL vs CDCMS: Sine1 (Time decay G-Mean)

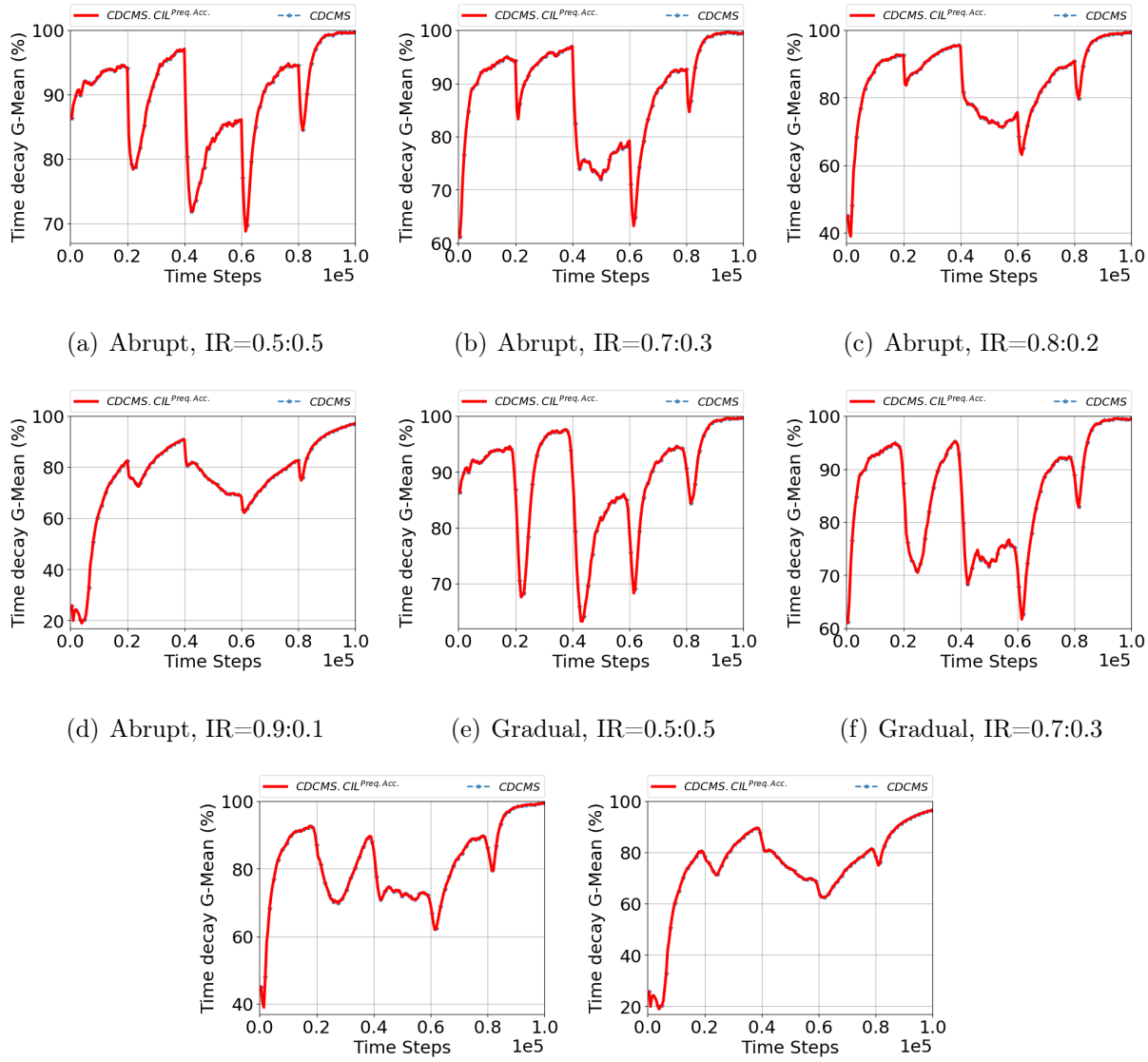


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure E.2: CDCMS.CIL vs CDCMS: Sine2 (Time decay G-Mean)

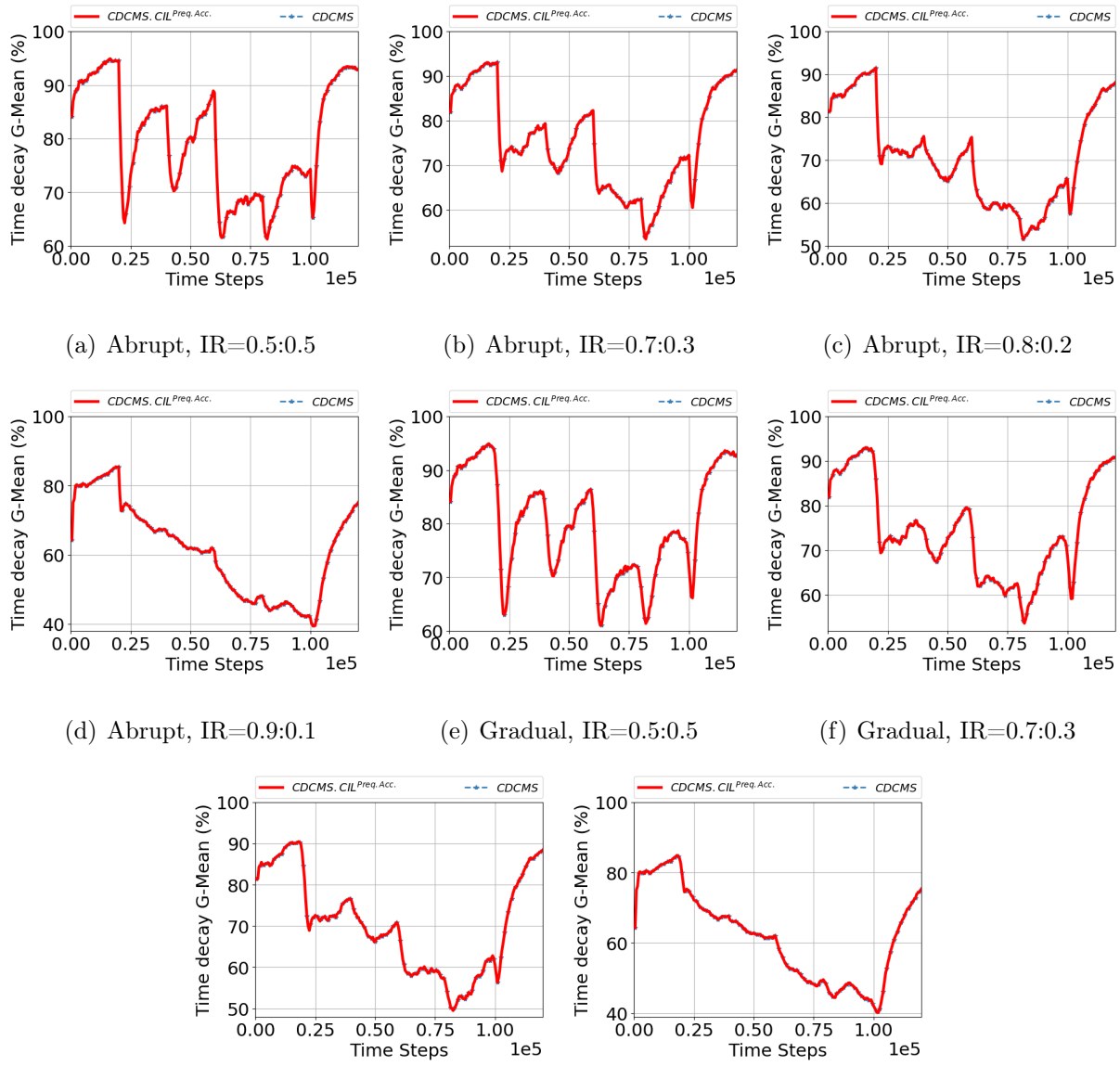


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure E.3: CDCMS.CIL vs CDCMS: Agrawal1 (Time decay G-Mean)

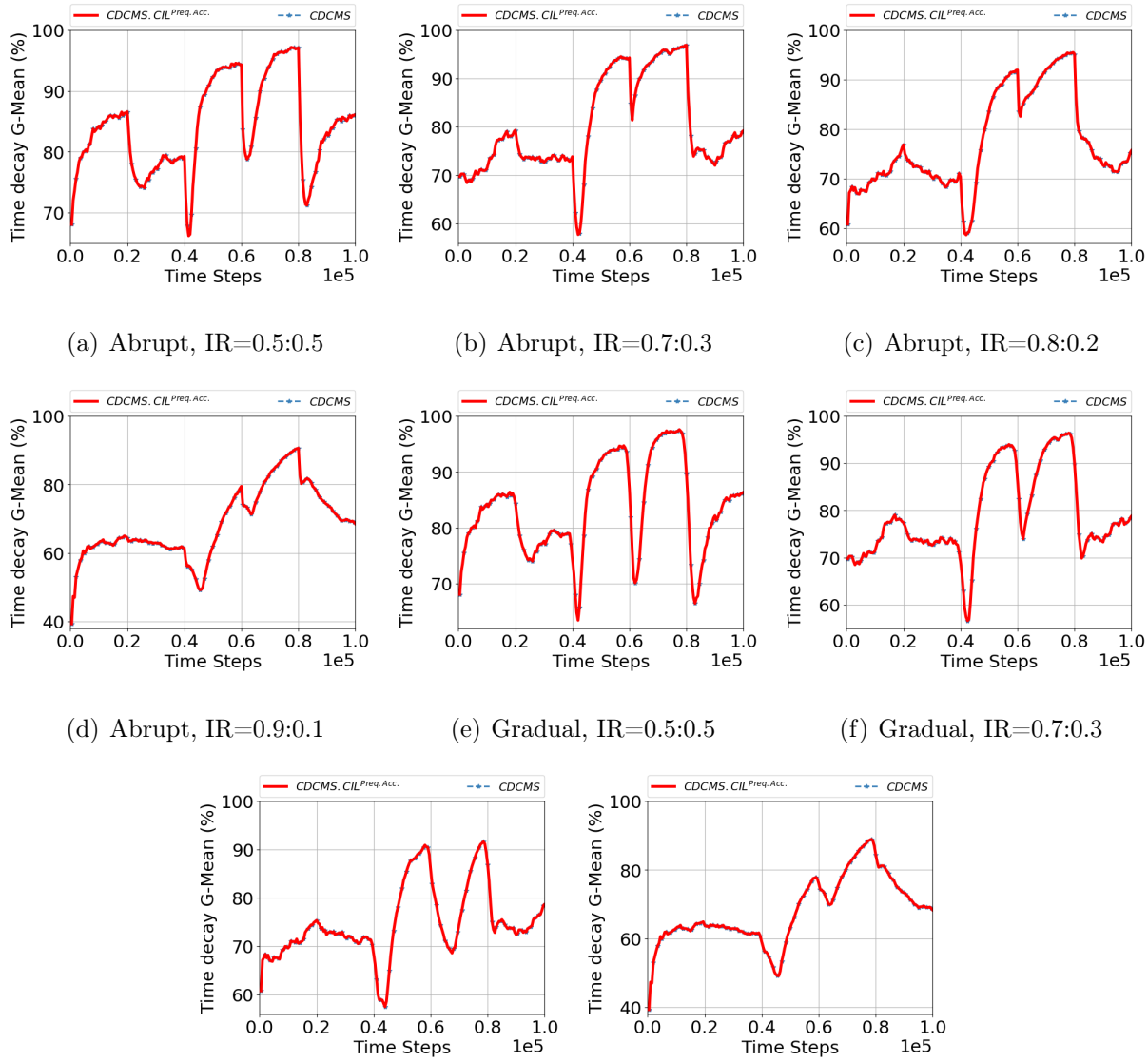


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure E.4: CDCMS.CIL vs CDCMS: Agrawal2 (Time decay G-Mean)

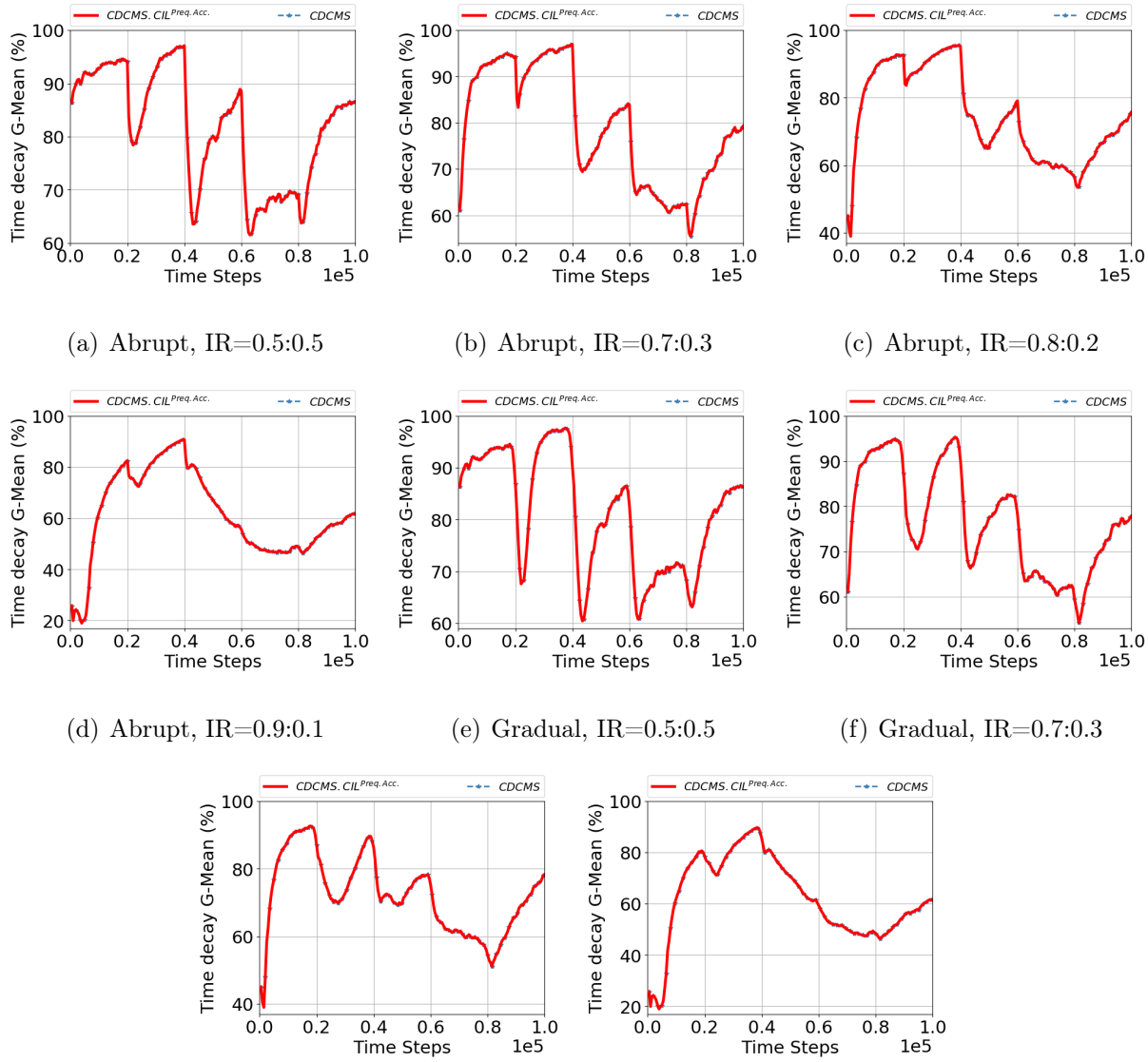


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure E.5: CDCMS.CIL vs CDCMS: Agrawal3 (Time decay G-Mean)

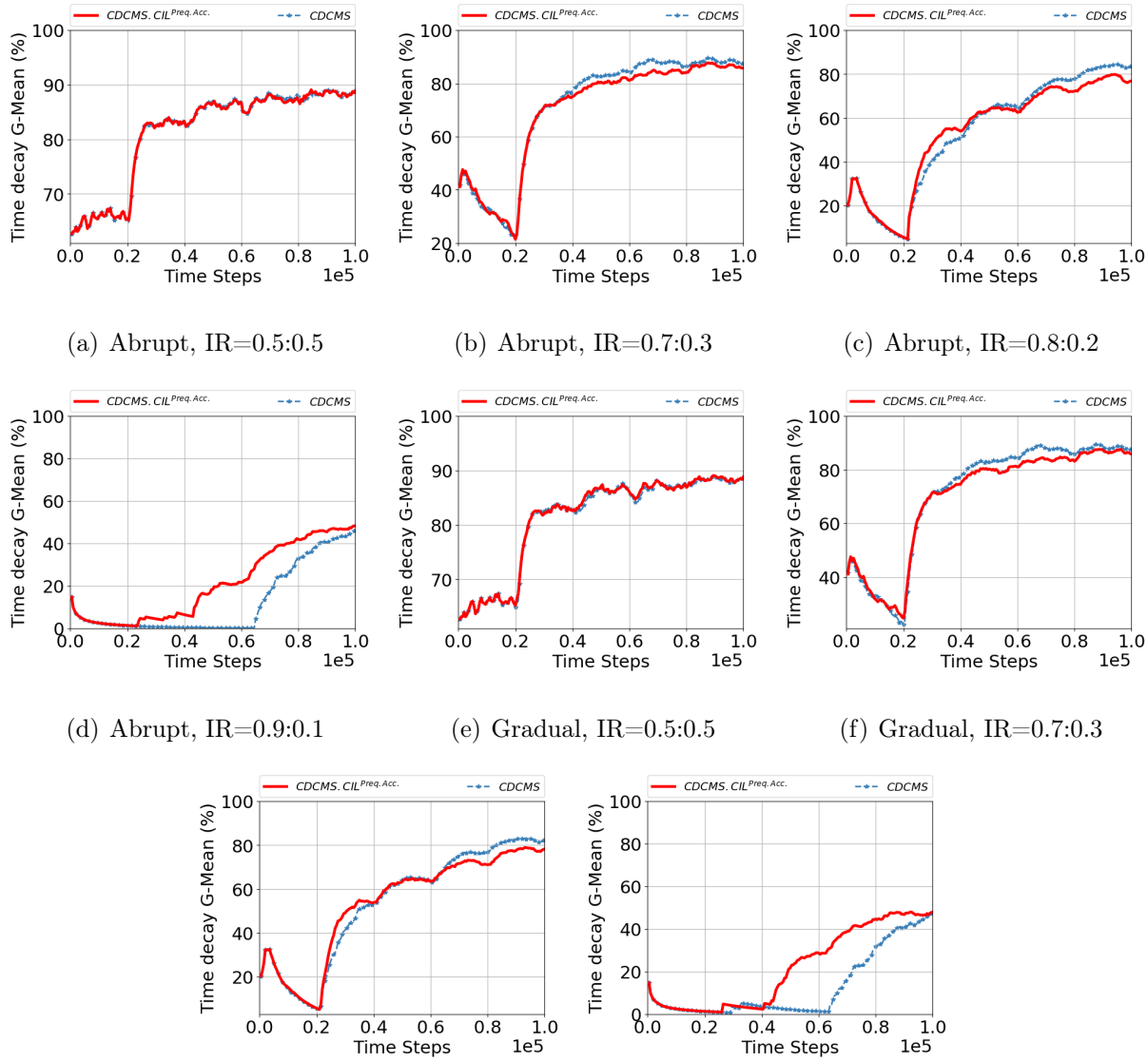


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure E.6: CDCMS.CIL vs CDCMS: Agrawal4 (Time decay G-Mean)

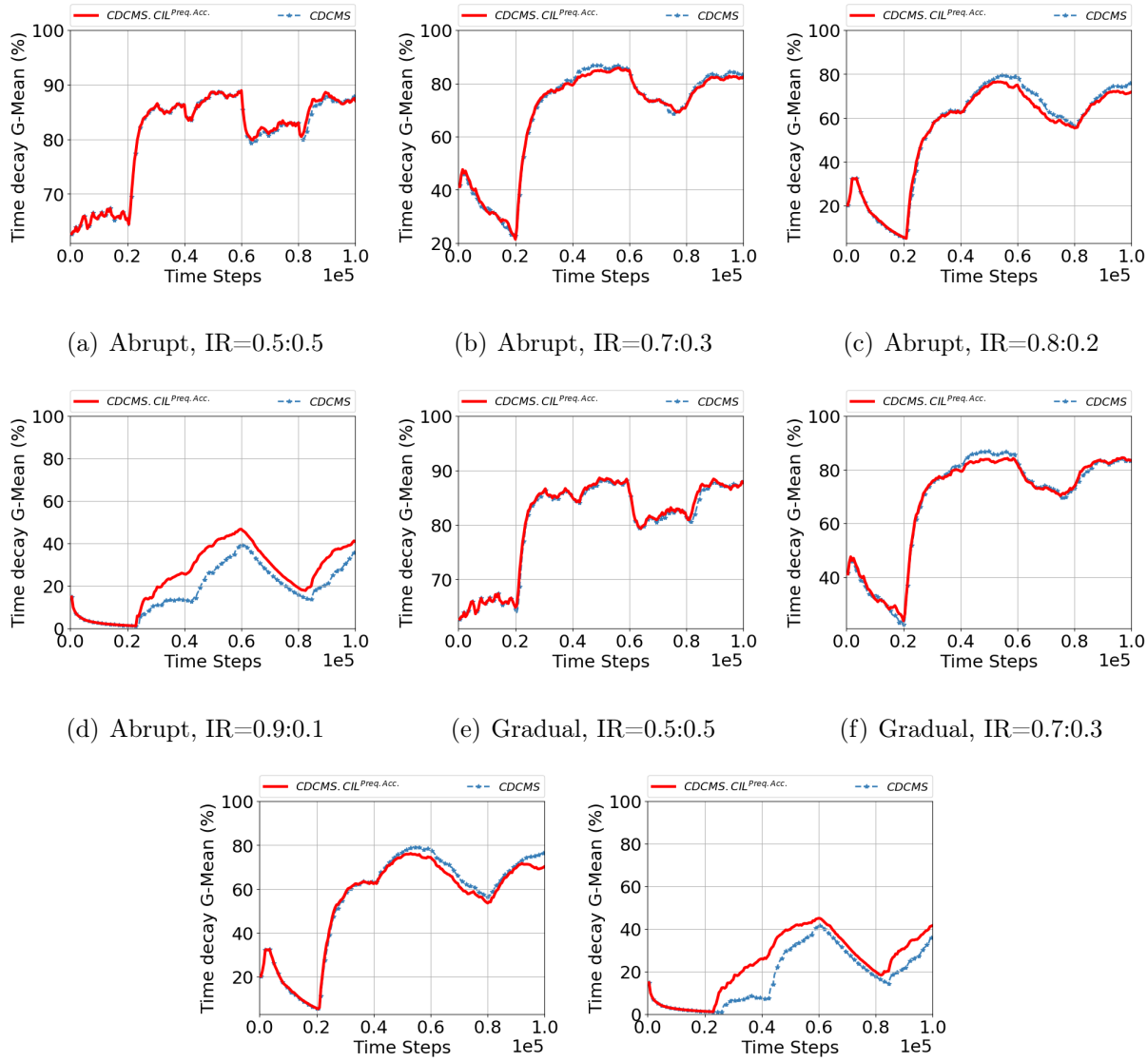


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure E.7: CDCMS.CIL vs CDCMS: SEA1 (Time decay G-Mean)

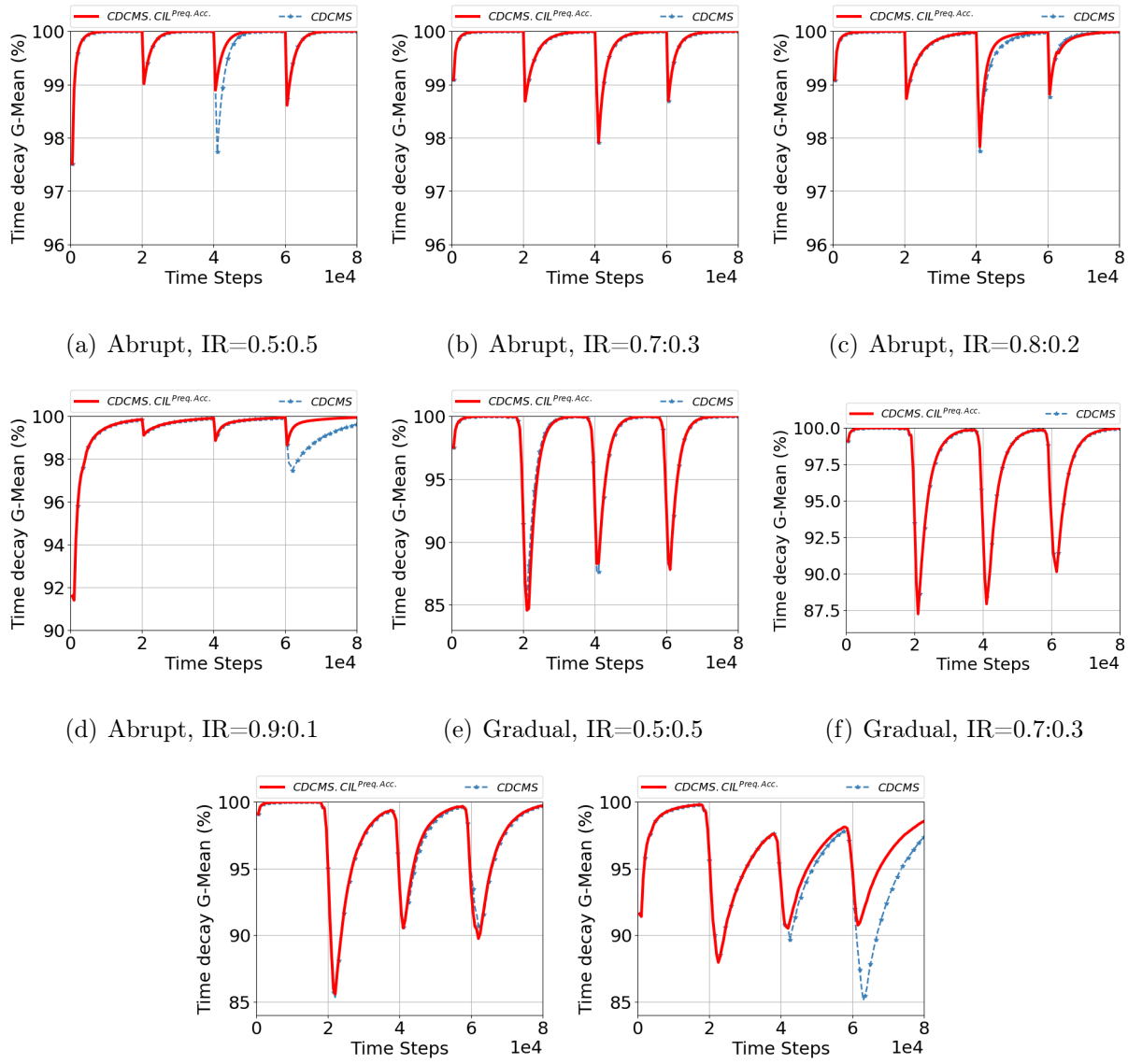


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure E.8: CDCMS.CIL vs CDCMS: SEA2 (Time decay G-Mean)

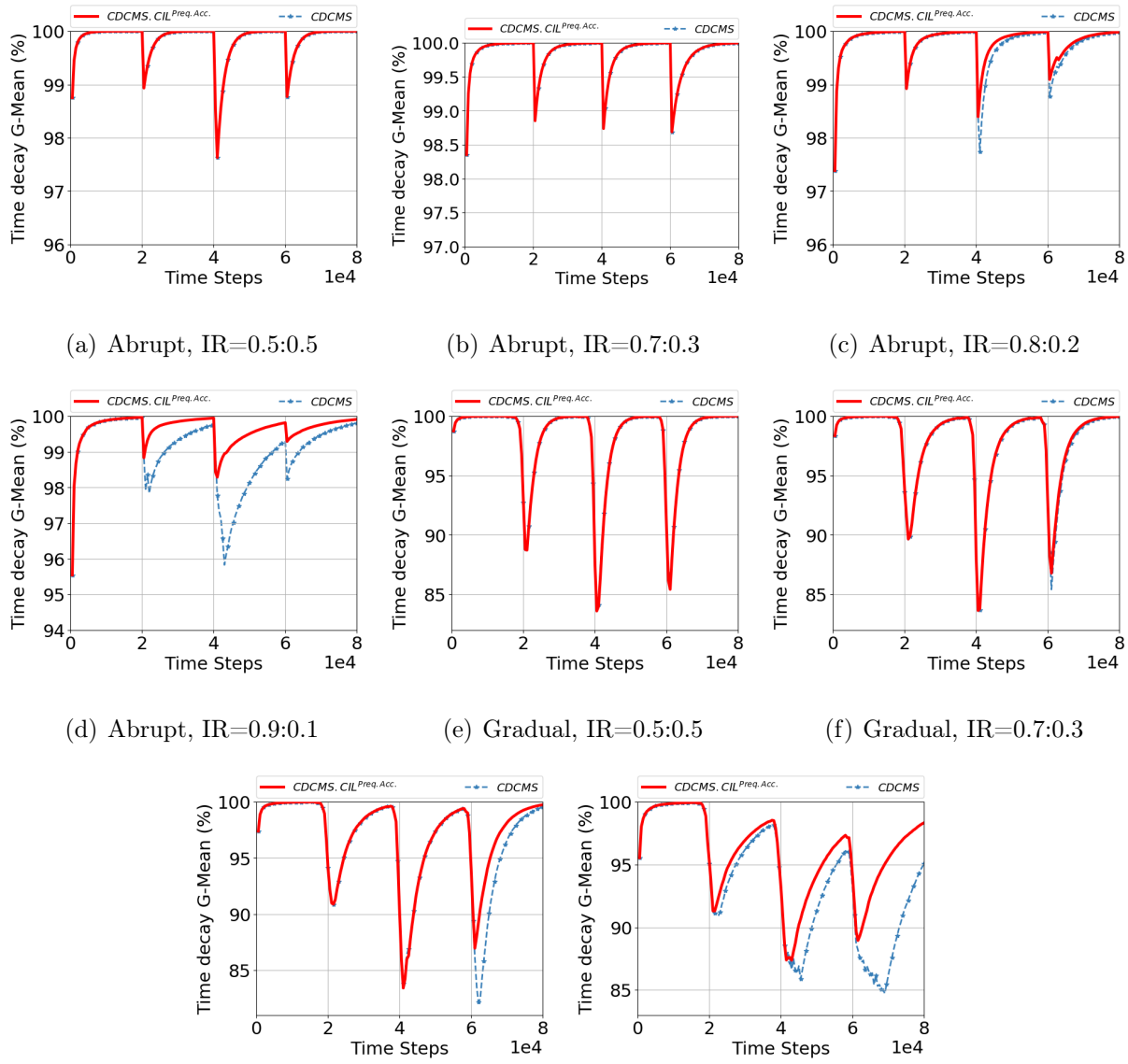


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure E.9: CDCMS.CIL vs CDCMS: STAGGER1 (Time decay G-Mean)



(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

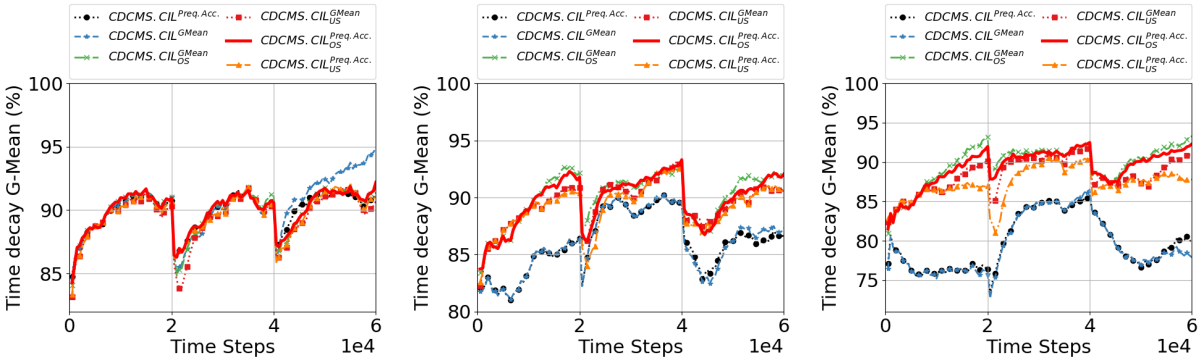
(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure E.10: CDCMS.CIL vs CDCMS: STAGGER2 (Time decay G-Mean)

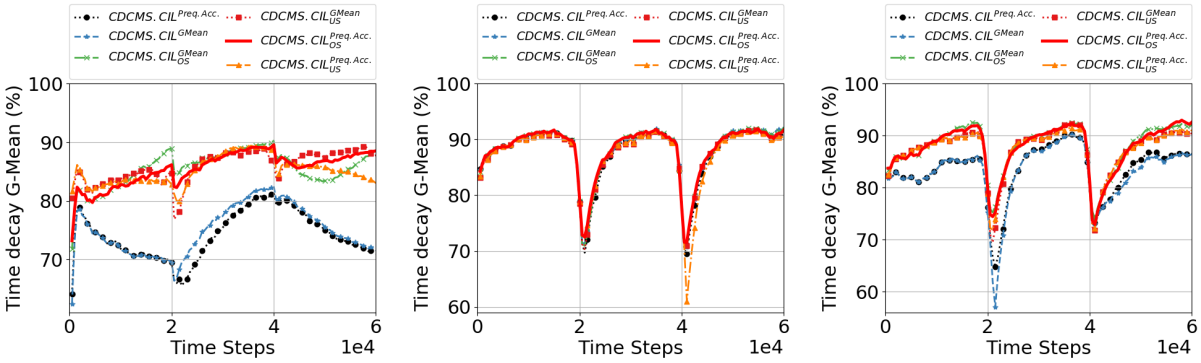
Appendix Six

All Performance Plots of the Comparison between Different Versions of CDCMS.CIL

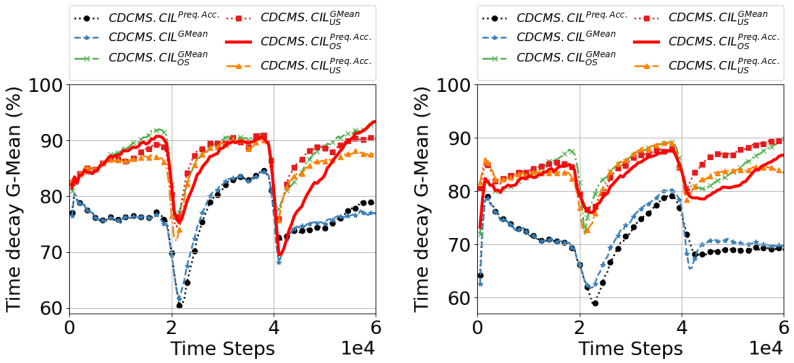
This section presents all results of comparing different version of CDCMS.CIL, regarding the discussion in Section 5.3.2. These versions are: CDCMS.CIL$^{\text{Preq. Acc.}}$, CDCMS.CIL$_{\text{OS}}^{\text{Preq. Acc.}}$, CDCMS.CIL$_{\text{US}}^{\text{Preq. Acc.}}$, CDCMS.CIL$^{\text{GMean}}$, CDCMS.CIL$_{\text{OS}}^{\text{GMean}}$, CDCMS.CIL$_{\text{US}}^{\text{GMean}}$, Figures F.1-F.10 present the their time decay G-Mean over the artificial data streams.



(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

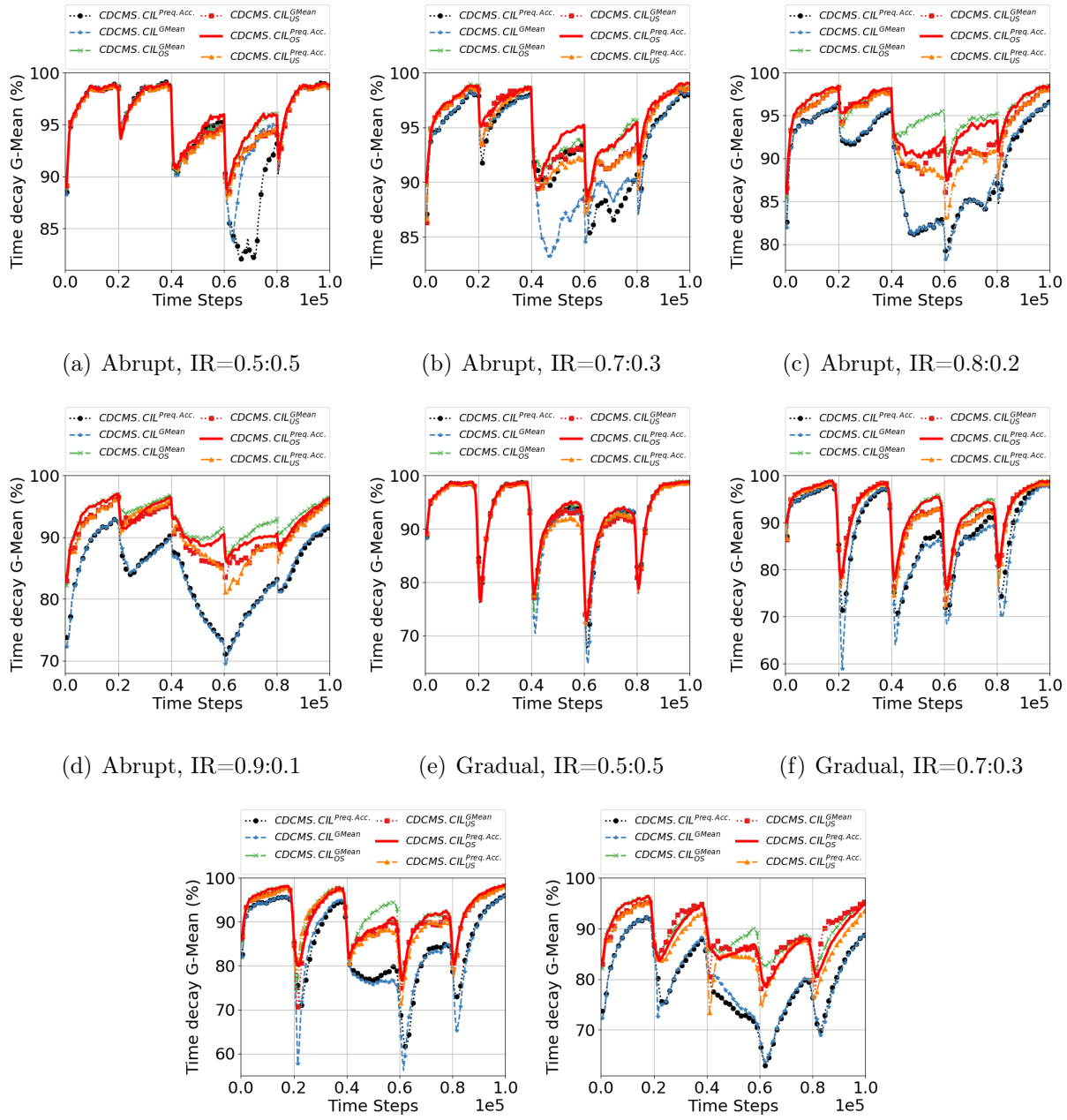


(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3



(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure F.1: Comparing Different Versions of CDCMS.CIL: Sine1 (Time decay G-Mean)

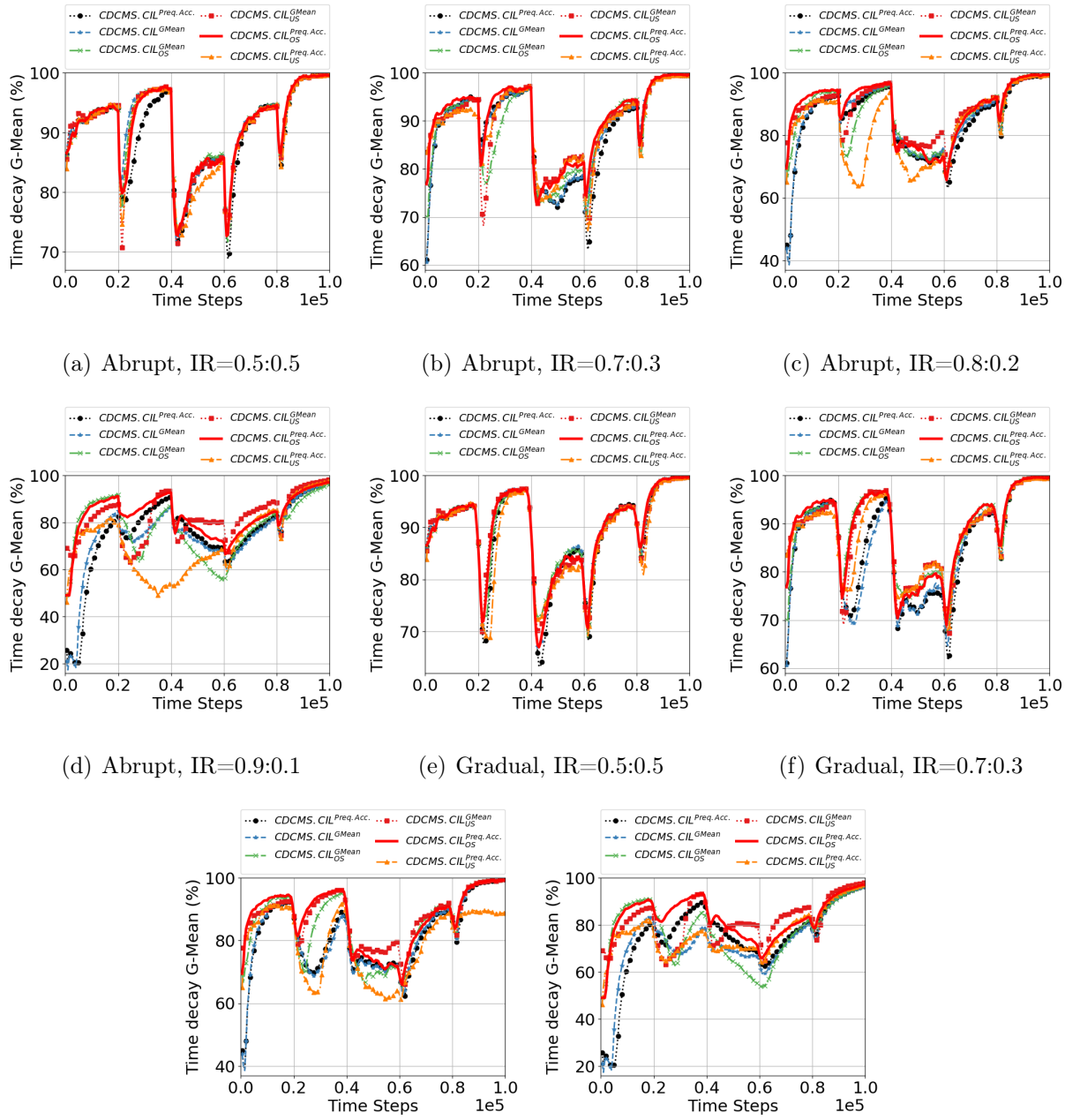


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure F.2: Comparing Different Versions of CDCMS.CIL: Sine2 (Time decay G-Mean)

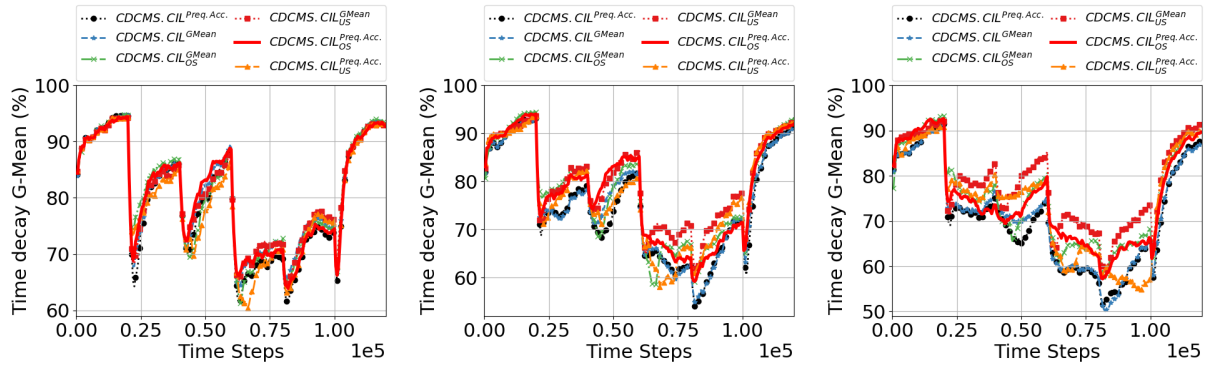


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

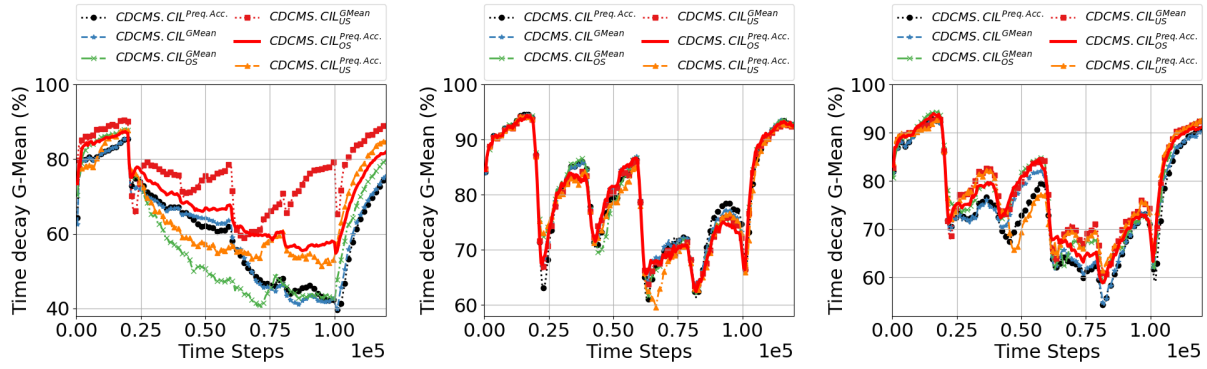
(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

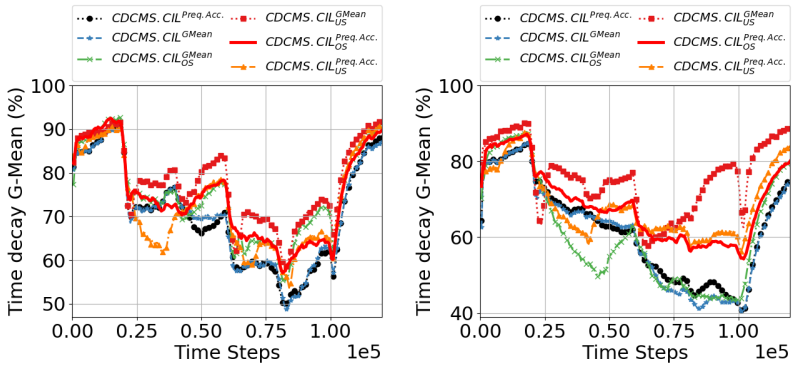
Figure F.3: Comparing Different Versions of CDCMS.CIL: Agrawal1 (Time decay G-Mean)



(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

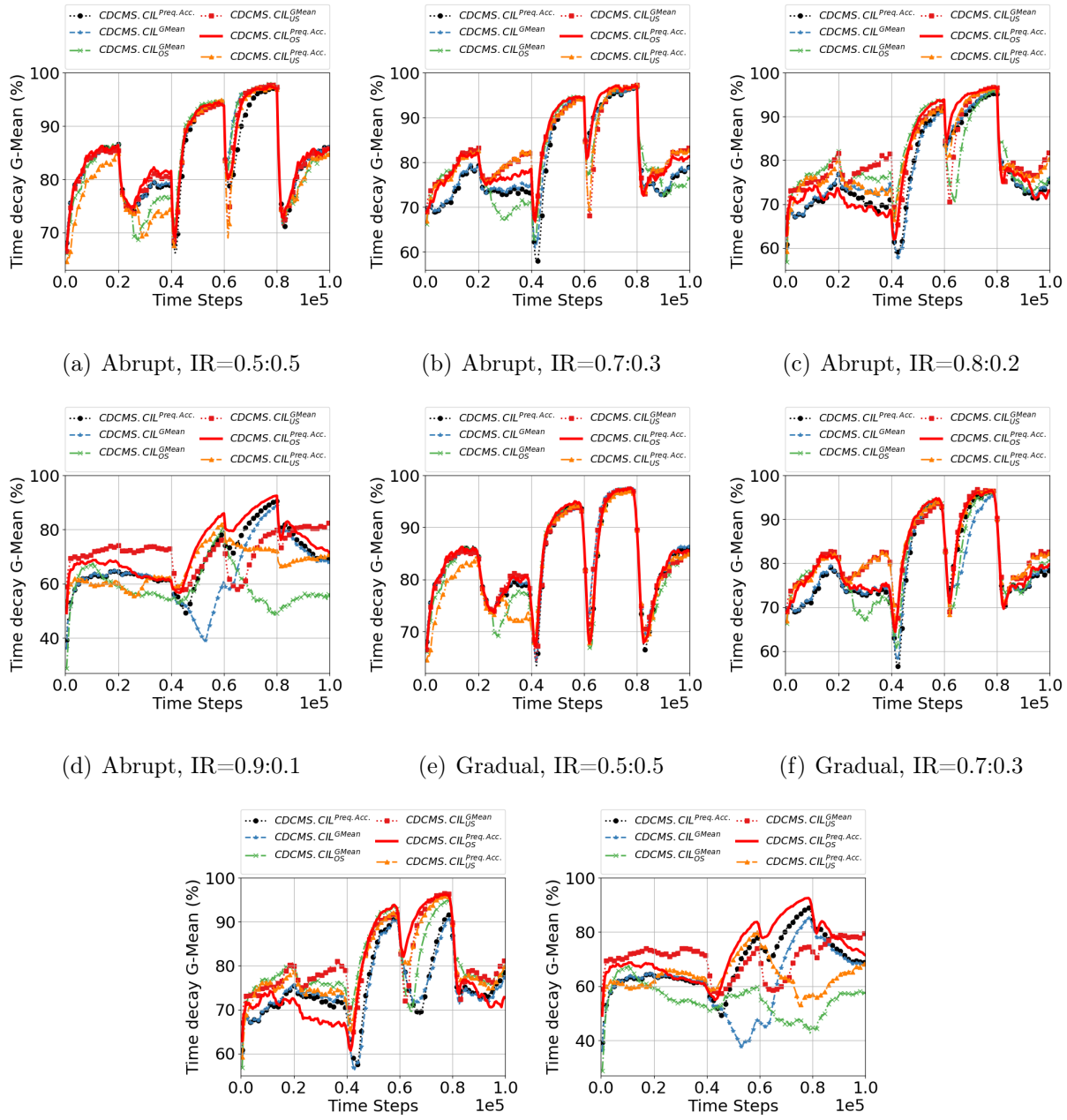


(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3



(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure F.4: Comparing Different Versions of CDCMS.CIL: Agrawal2 (Time decay G-Mean)

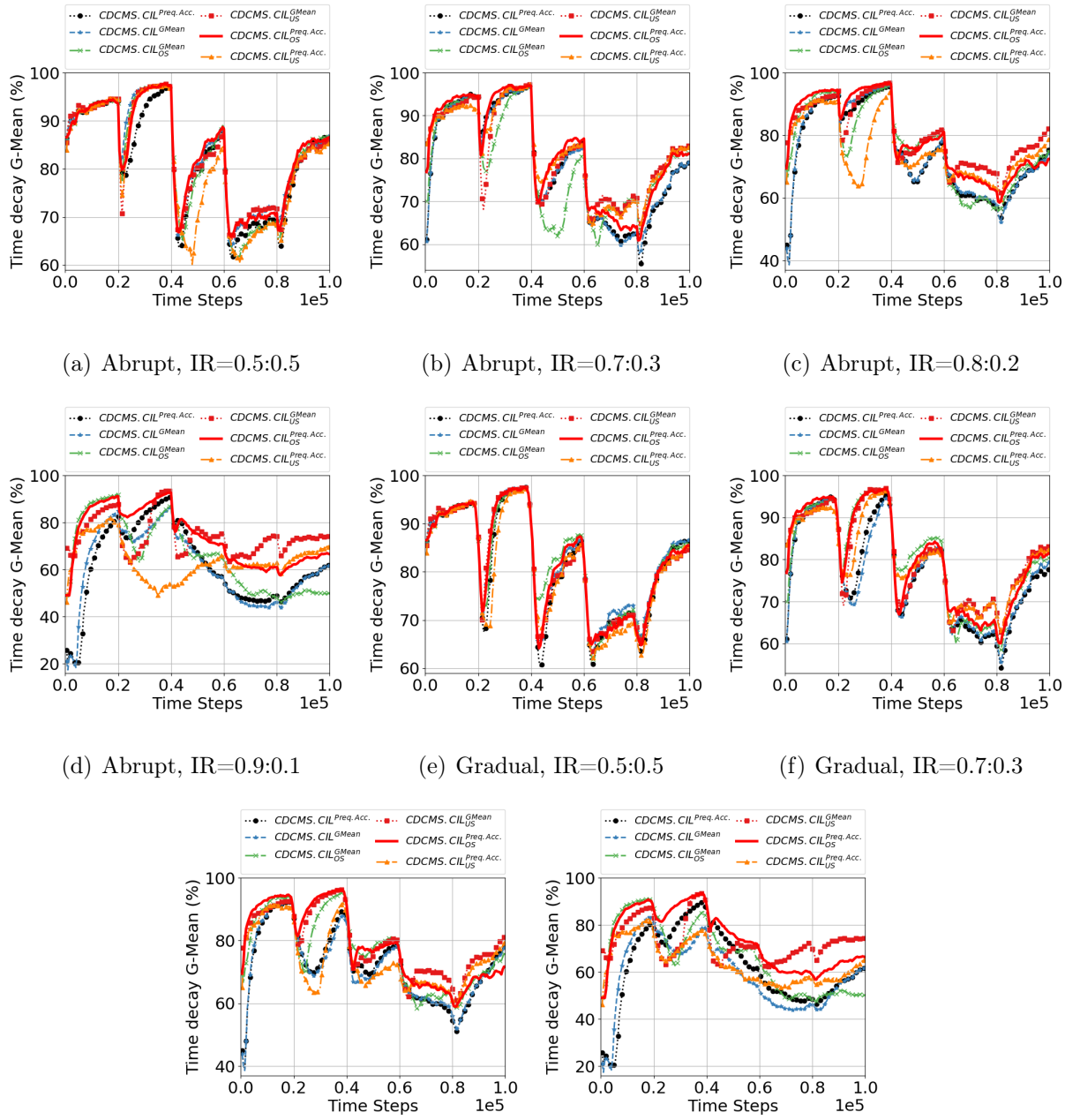


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure F.5: Comparing Different Versions of CDCMS.CIL: Agrawal3 (Time decay G-Mean)

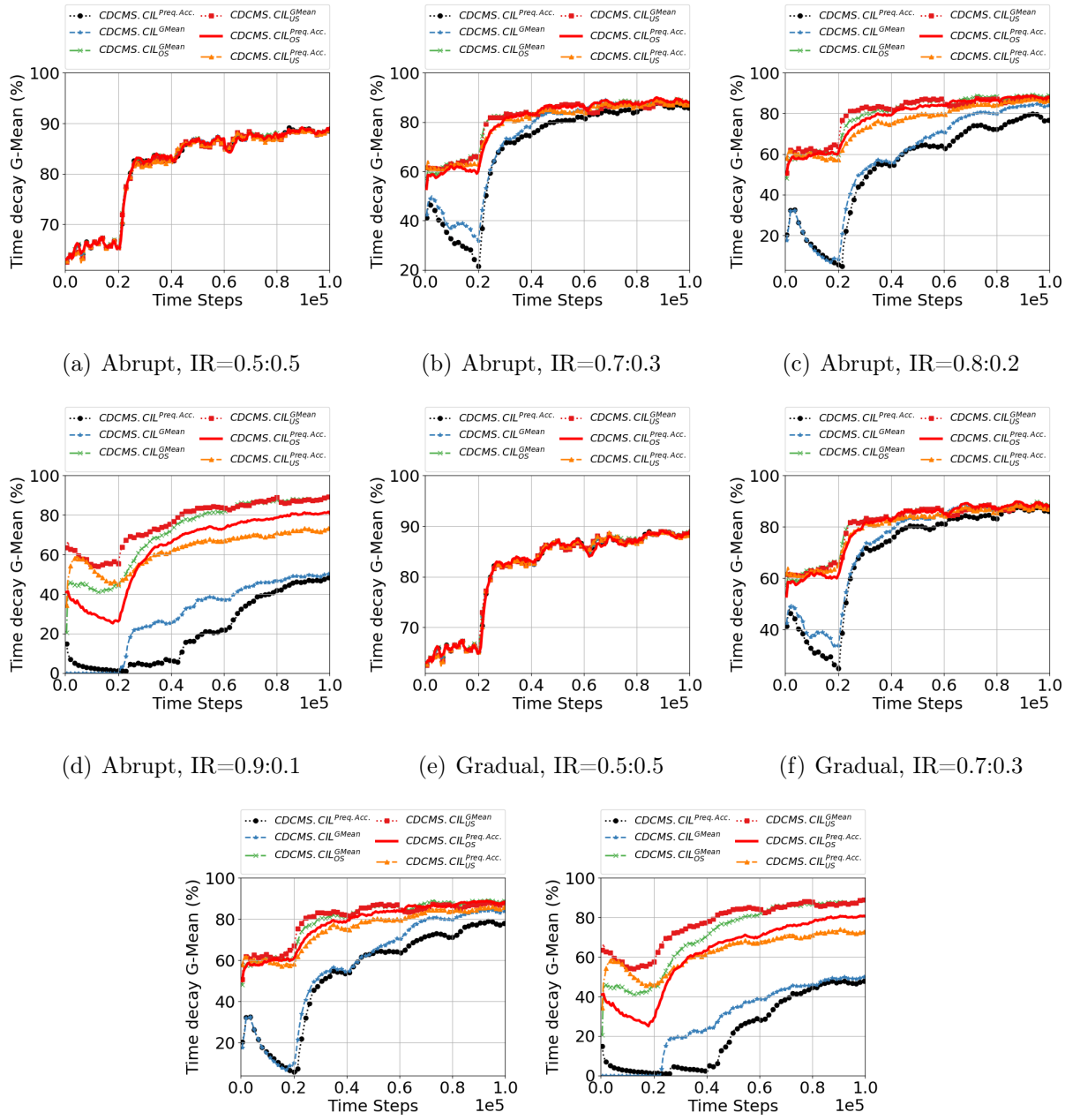


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure F.6: Comparing Different Versions of CDCMS.CIL: Agrawal4 (Time decay G-Mean)

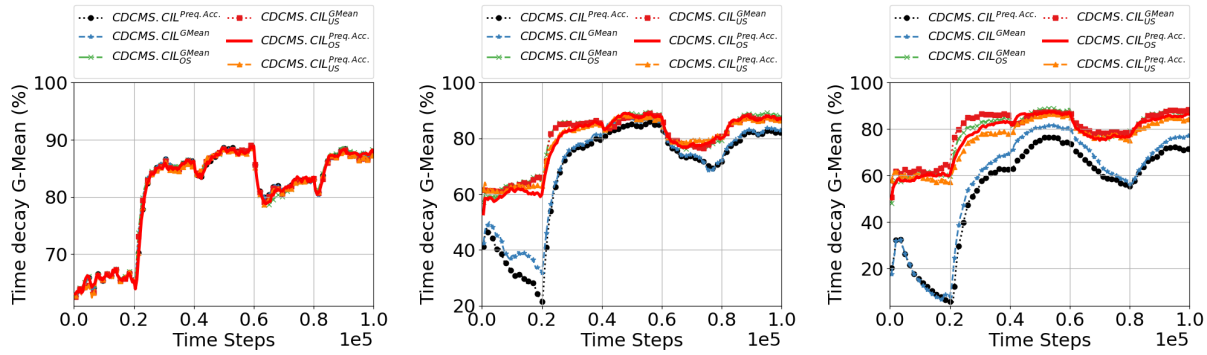


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

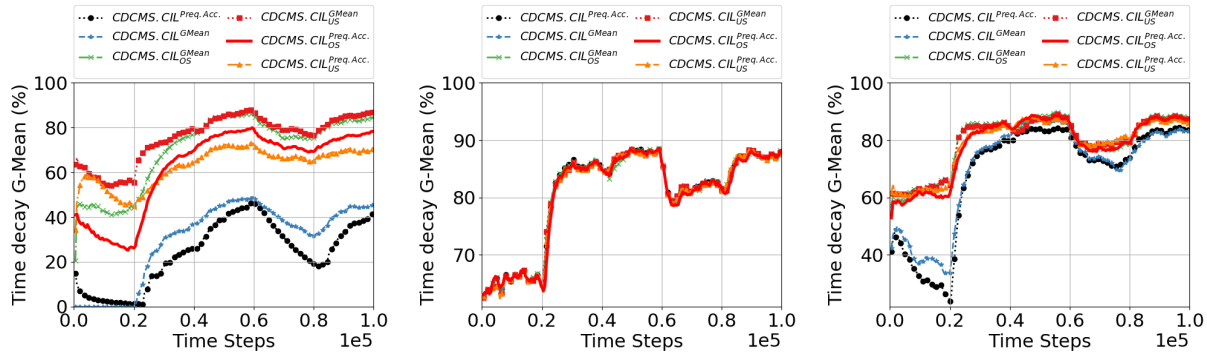
(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

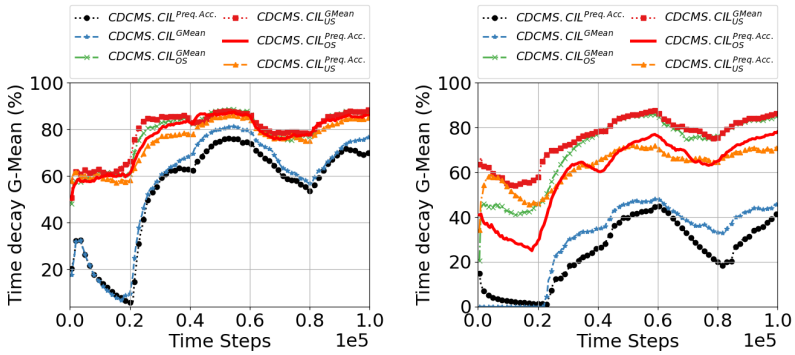
Figure F.7: Comparing Different Versions of CDCMS.CIL: SEA1 (Time decay G-Mean)



(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

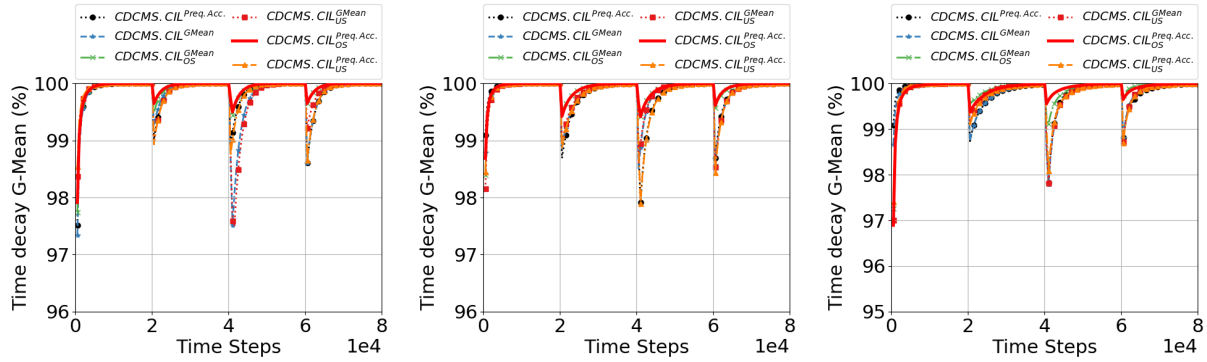


(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

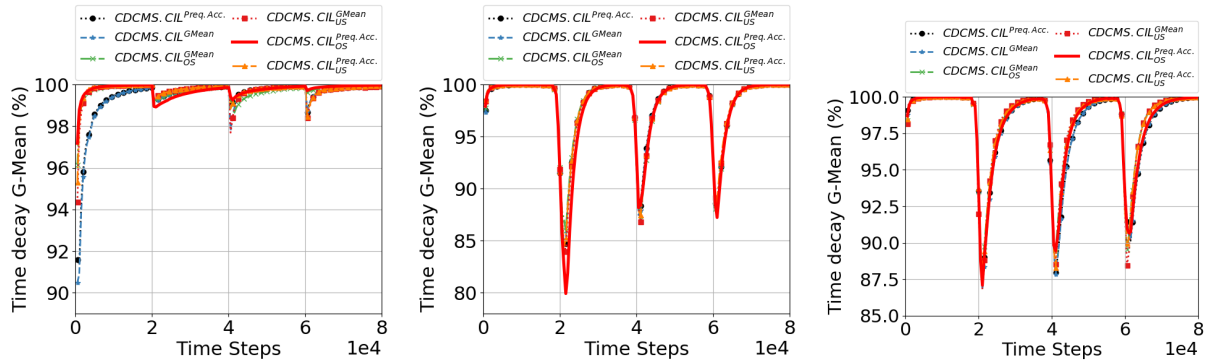


(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

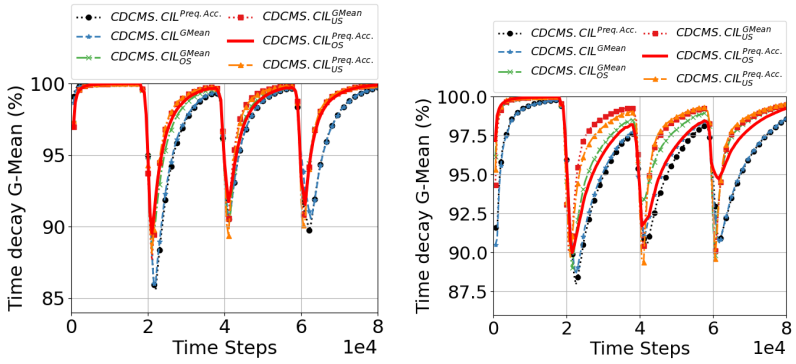
Figure F.8: Comparing Different Versions of CDCMS.CIL: SEA2 (Time decay G-Mean)



(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

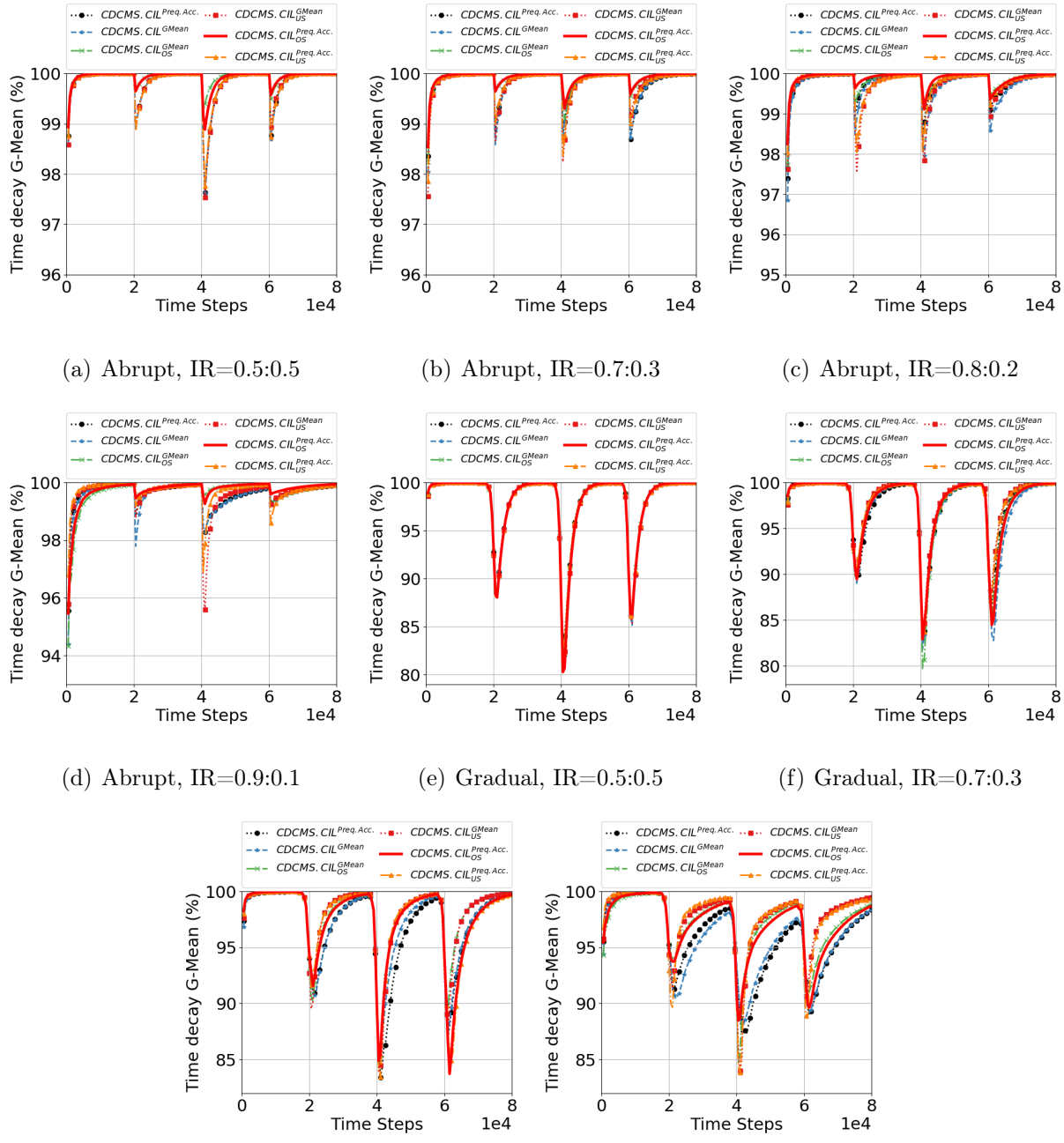


(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3



(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure F.9: Comparing Different Versions of CDCMS.CIL: STAGGER1 (Time decay G-Mean)



(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

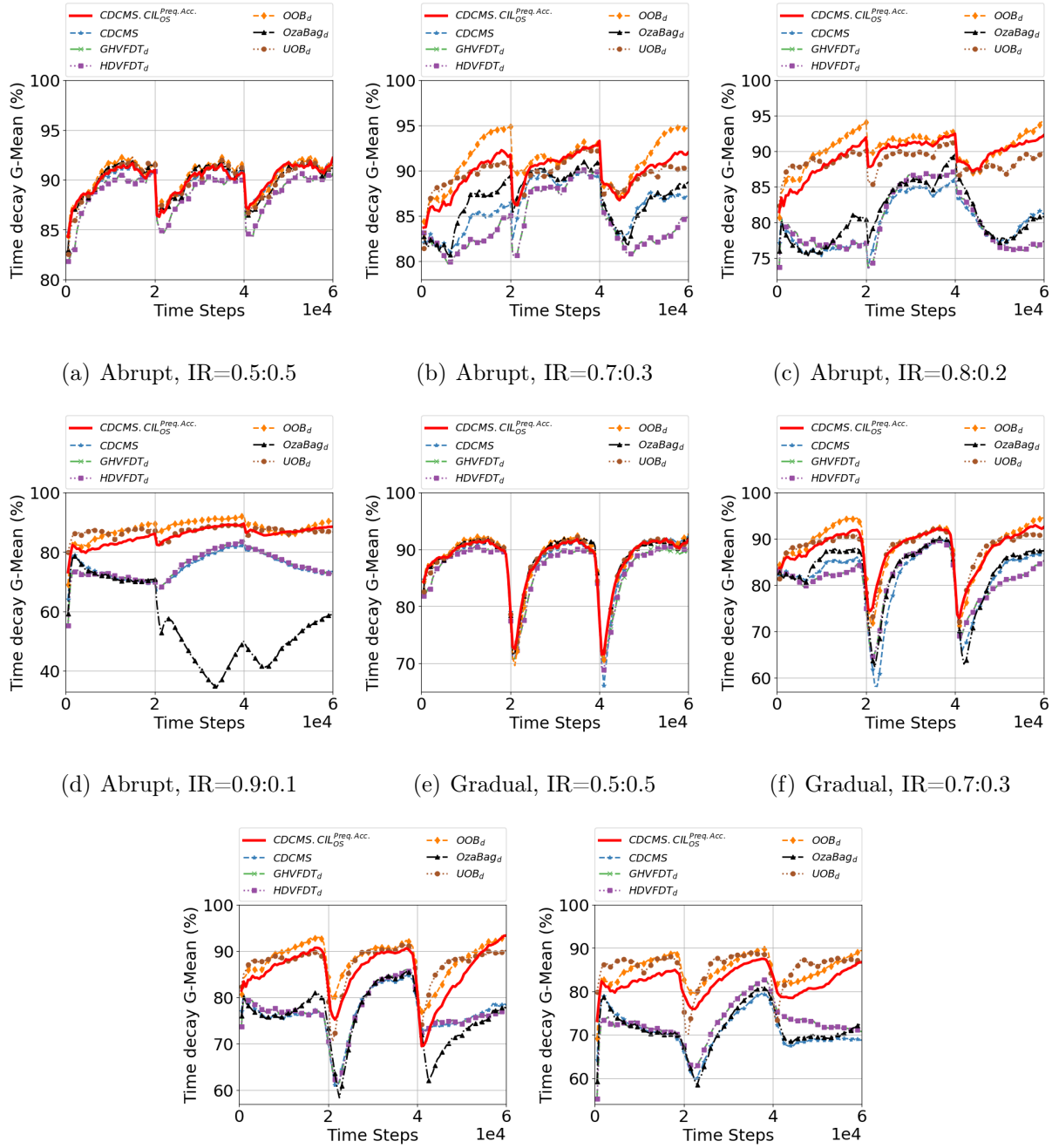
(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure F.10: Comparing Different Versions of CDCMS.CIL: STAGGER2 (Time decay G-Mean)

Appendix Seven

All Performance Plots of the Comparison between CDCMS.CIL$^{\text{Preq. Acc.}}_{\text{OS}}$ and Existing Approaches with and without Drift Detector

This section presents all results of comparing CDCMS.CIL$^{\text{Preq. Acc.}}_{\text{OS}}$ against GH-VFDT, HD-VFDT, OzaBag, OOB and UOB with and without drift detector, regarding the discussion in Section 5.3.3. Table G.1, G.2 and G.3 present their rankings in time decay G-Mean, class 0 (majority class) and class 1 (minority class) recalls respectively. Figures G.1-G.10 present the their time decay G-Mean over the artificial data streams. Figure G.11 present the their time decay G-Mean over the real-world data streams, except Covtype streams. Figure G.12 present the their time decay G-Mean over the Covtype streams.

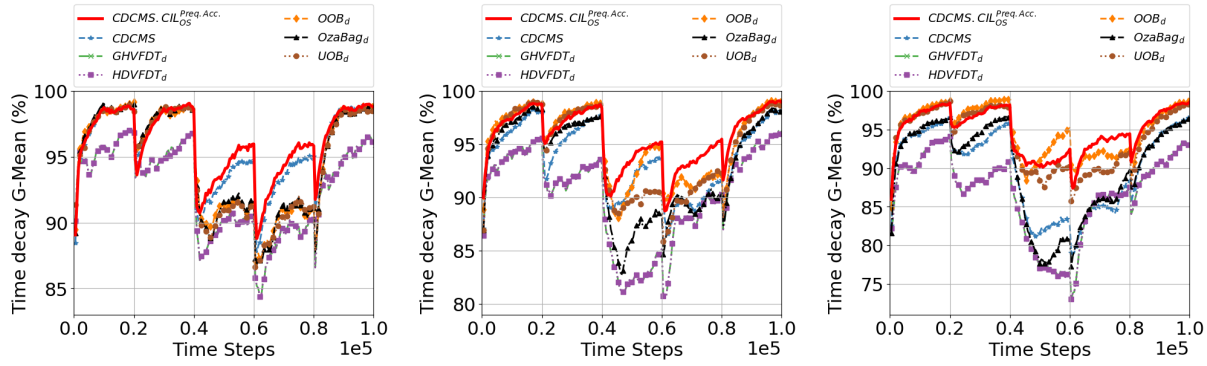


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

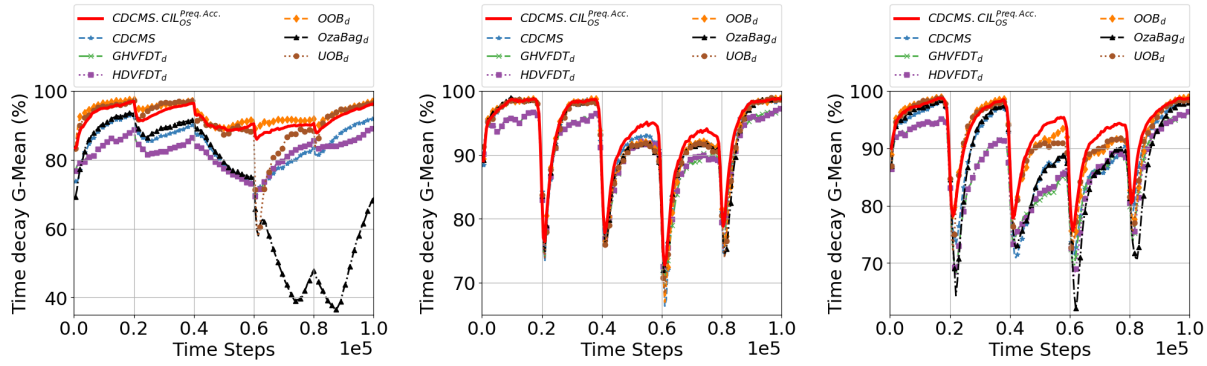
(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

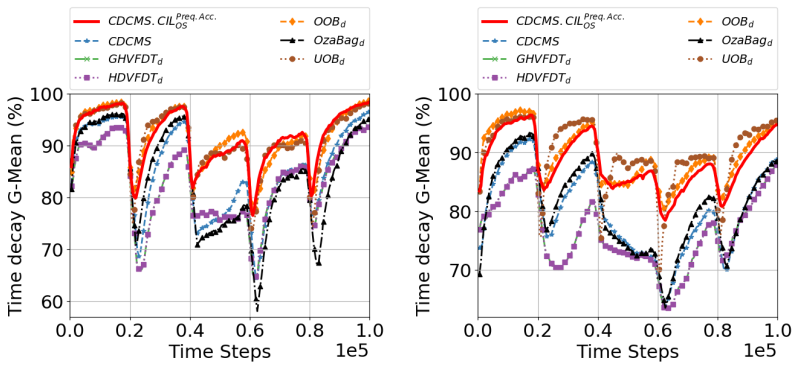
Figure G.1: Comparing CDCMS.CIL against Existing Approaches: Sine1 (Time decay G-Mean)



(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

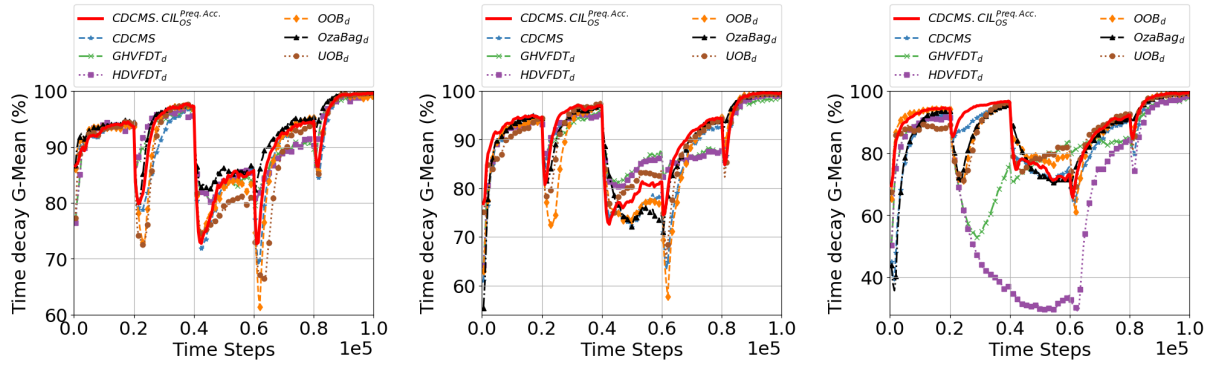


(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

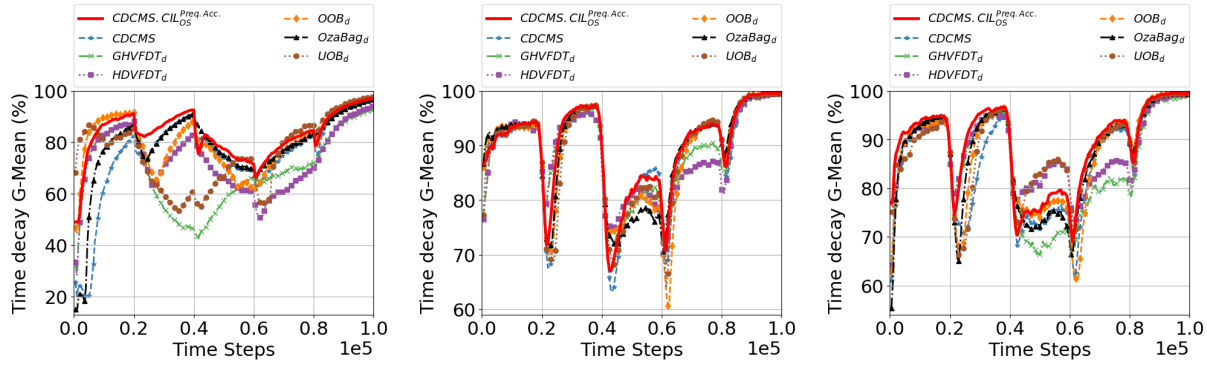


(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

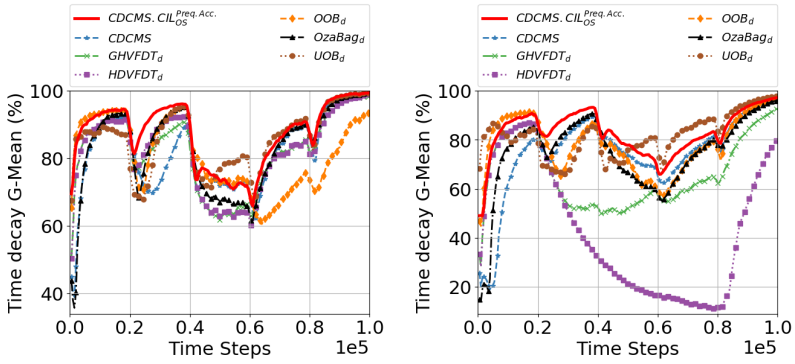
Figure G.2: Comparing CDCMS.CIL against Existing Approaches: Sine2 (Time decay G-Mean)



(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

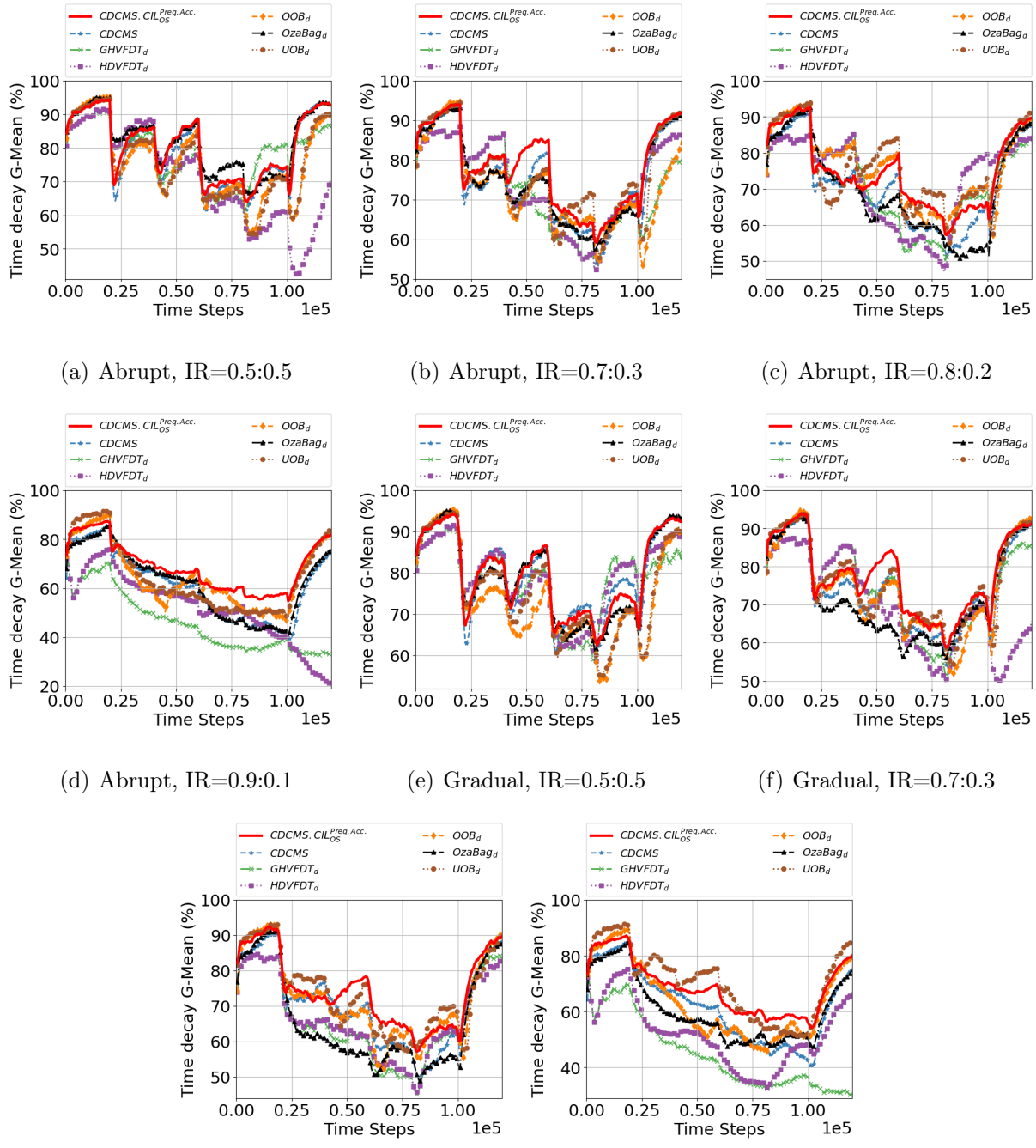


(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3



(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure G.3: Comparing CDCMS.CIL against Existing Approaches: Agrawal1 (Time decay G-Mean)

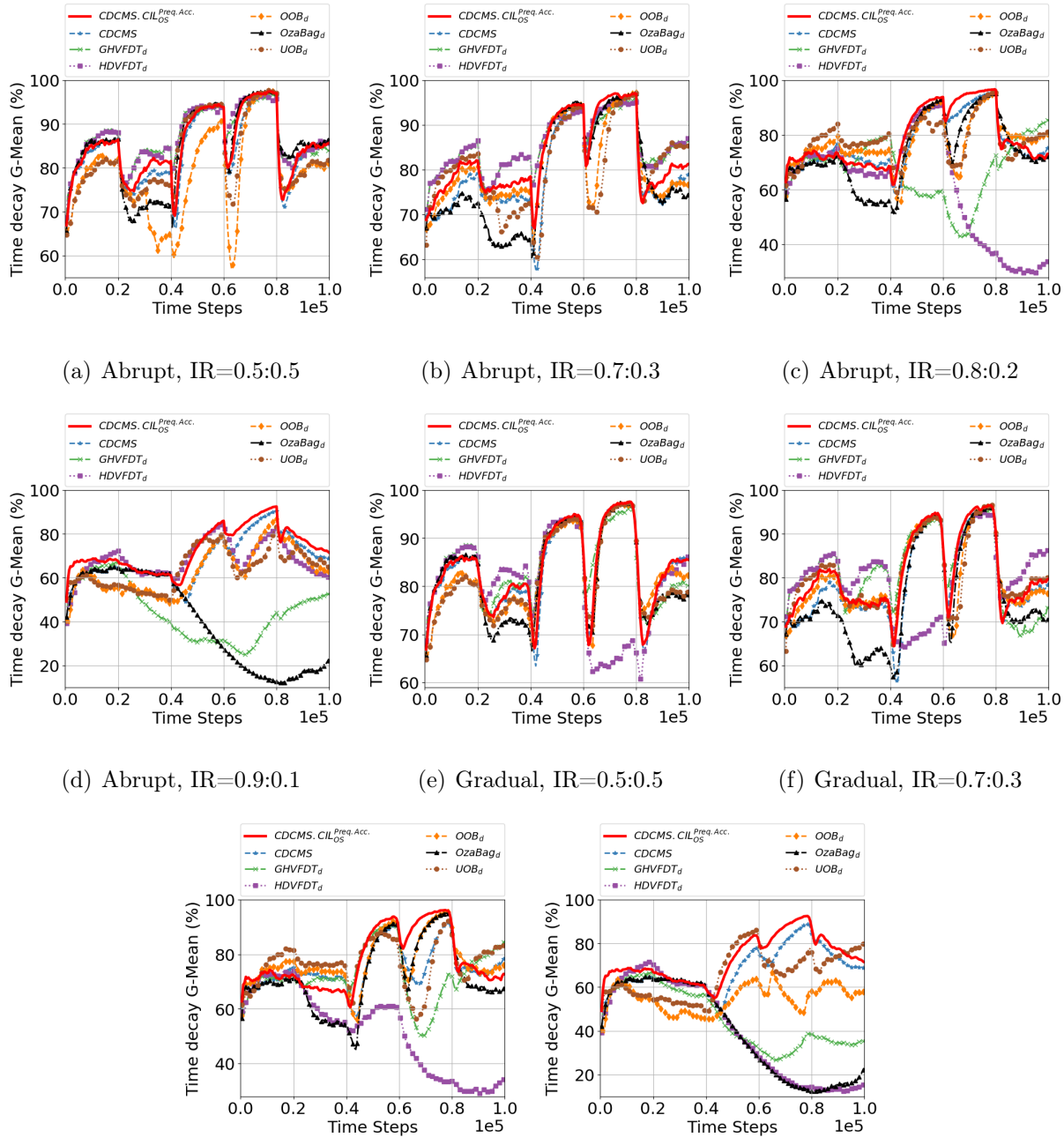


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure G.4: Comparing CDCMS.CIL against Existing Approaches: Agrawal2 (Time decay G-Mean)

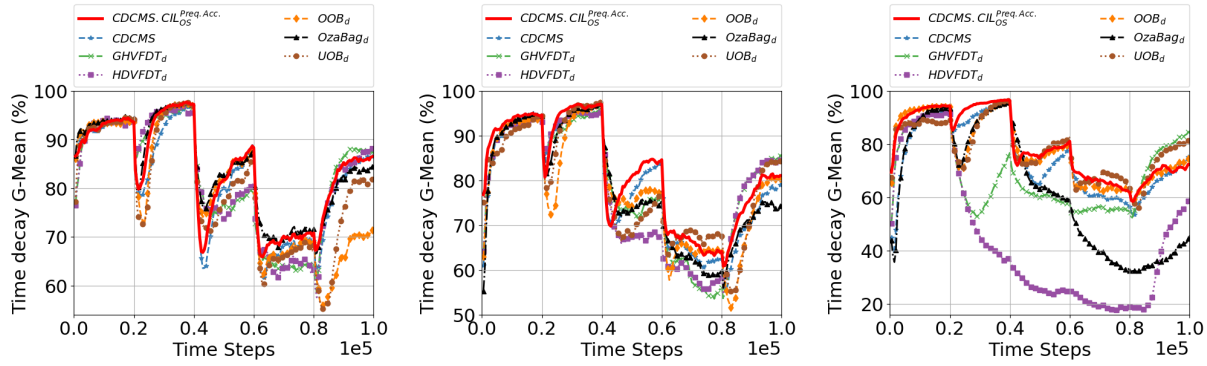


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

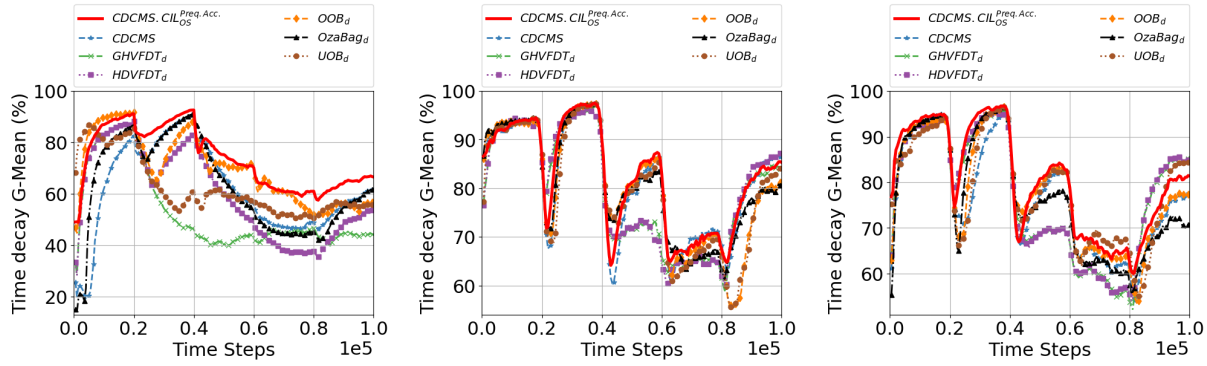
(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

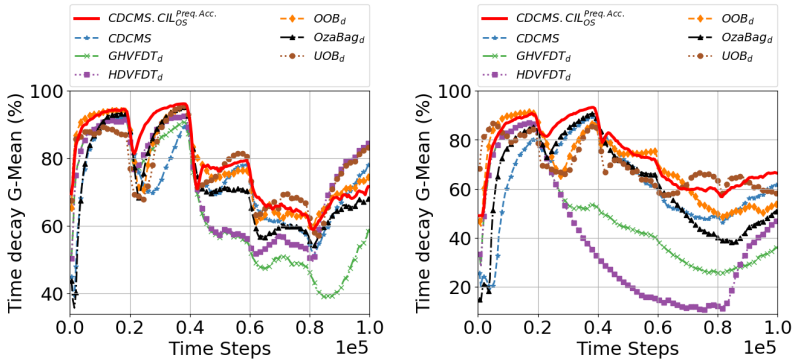
Figure G.5: Comparing CDCMS.CIL against Existing Approaches: Agrawal3 (Time decay G-Mean)



(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

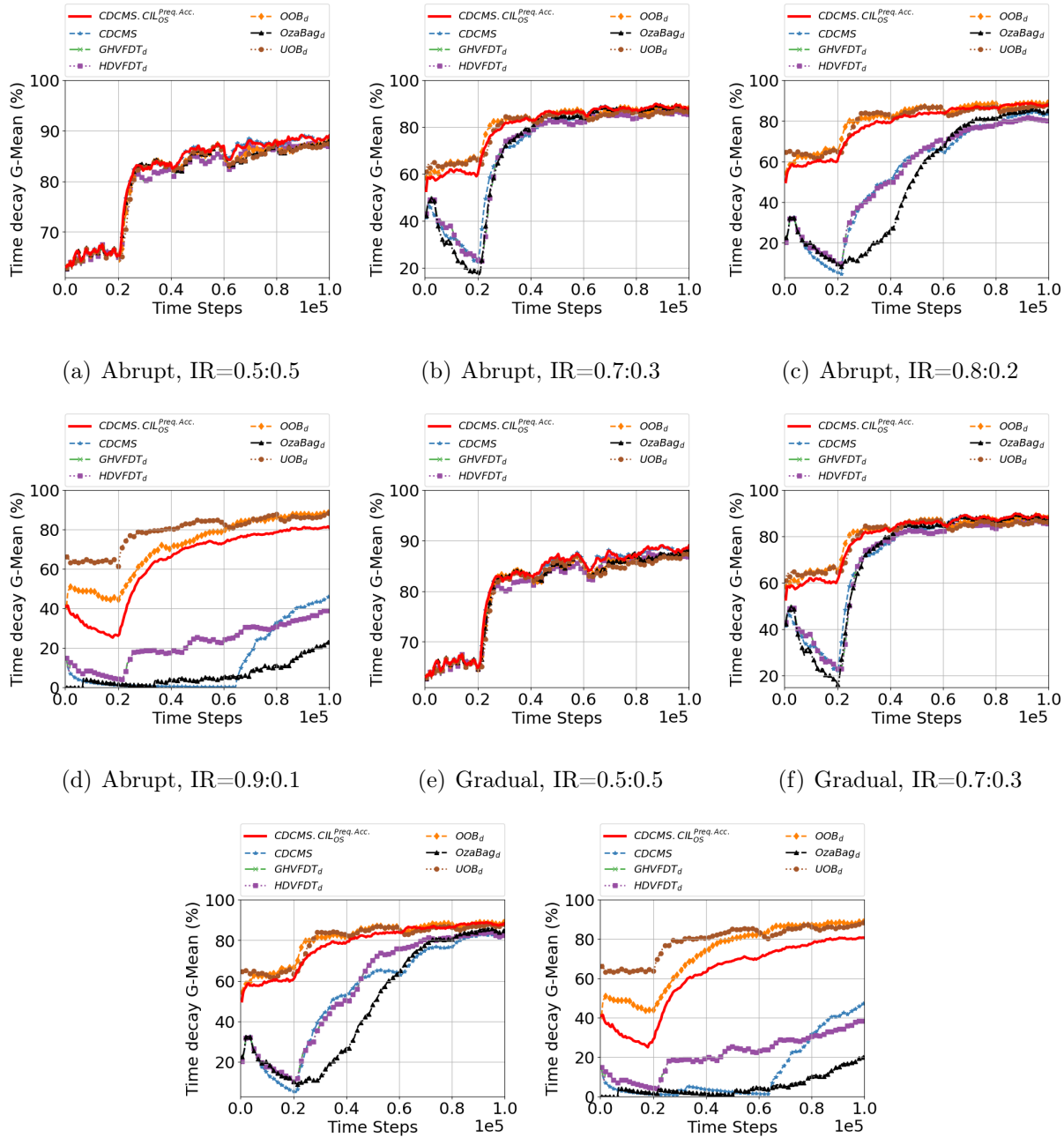


(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3



(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure G.6: Comparing CDCMS.CIL against Existing Approaches: Agrawal4 (Time decay G-Mean)

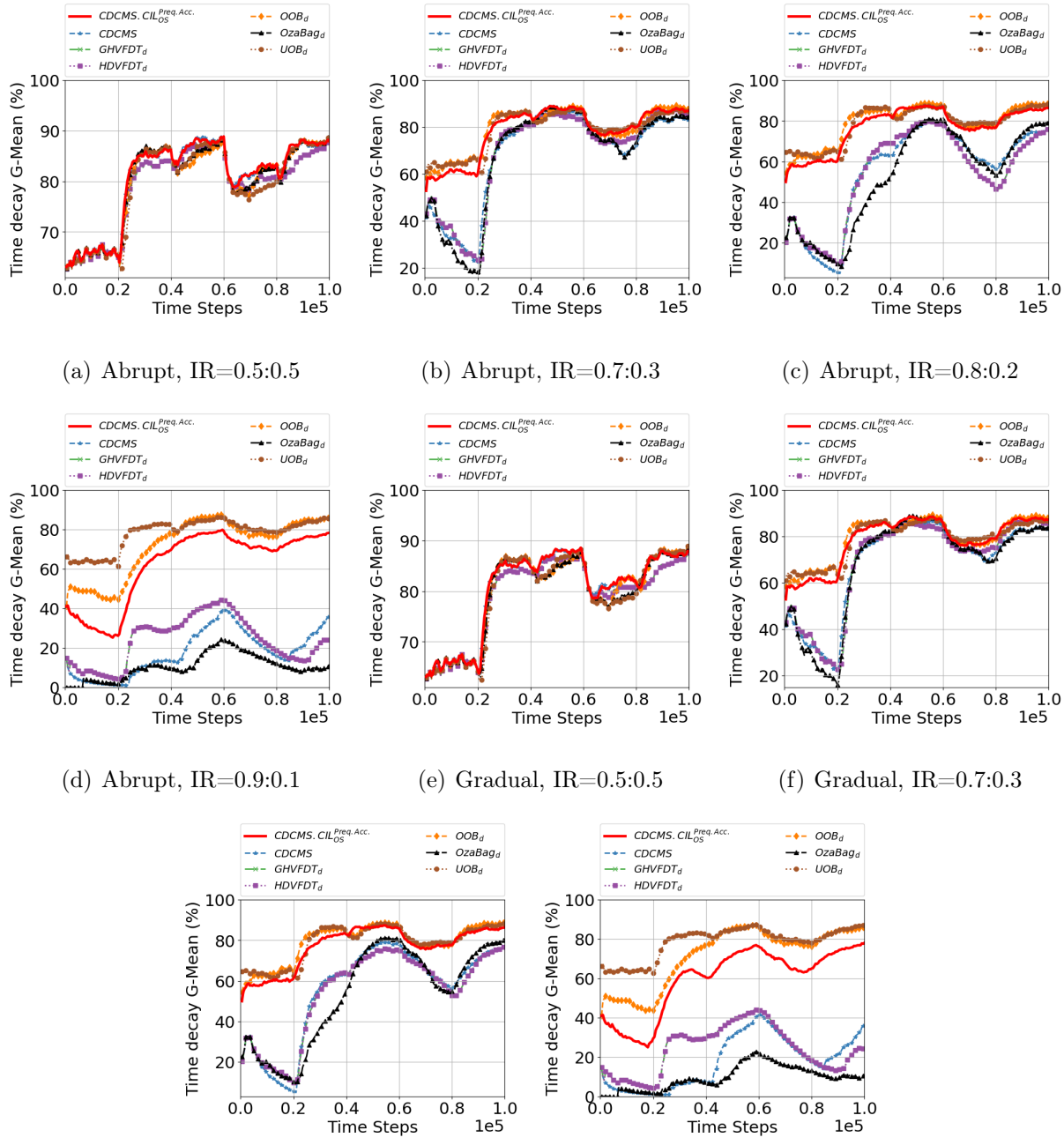


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure G.7: Comparing CDCMS.CIL against Existing Approaches: SEA1 (Time decay G-Mean)

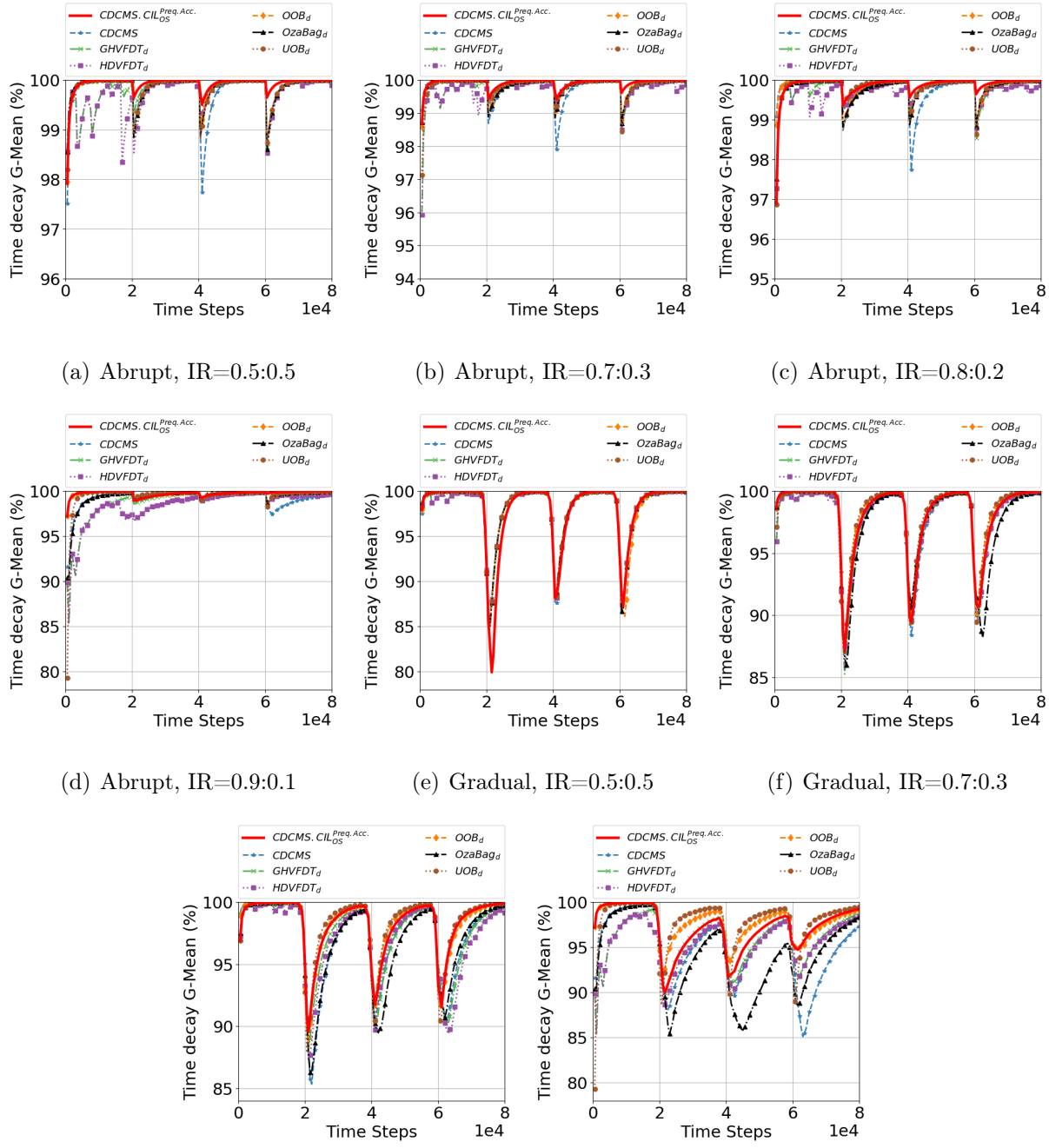


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure G.8: Comparing CDCMS.CIL against Existing Approaches: SEA2 (Time decay G-Mean)

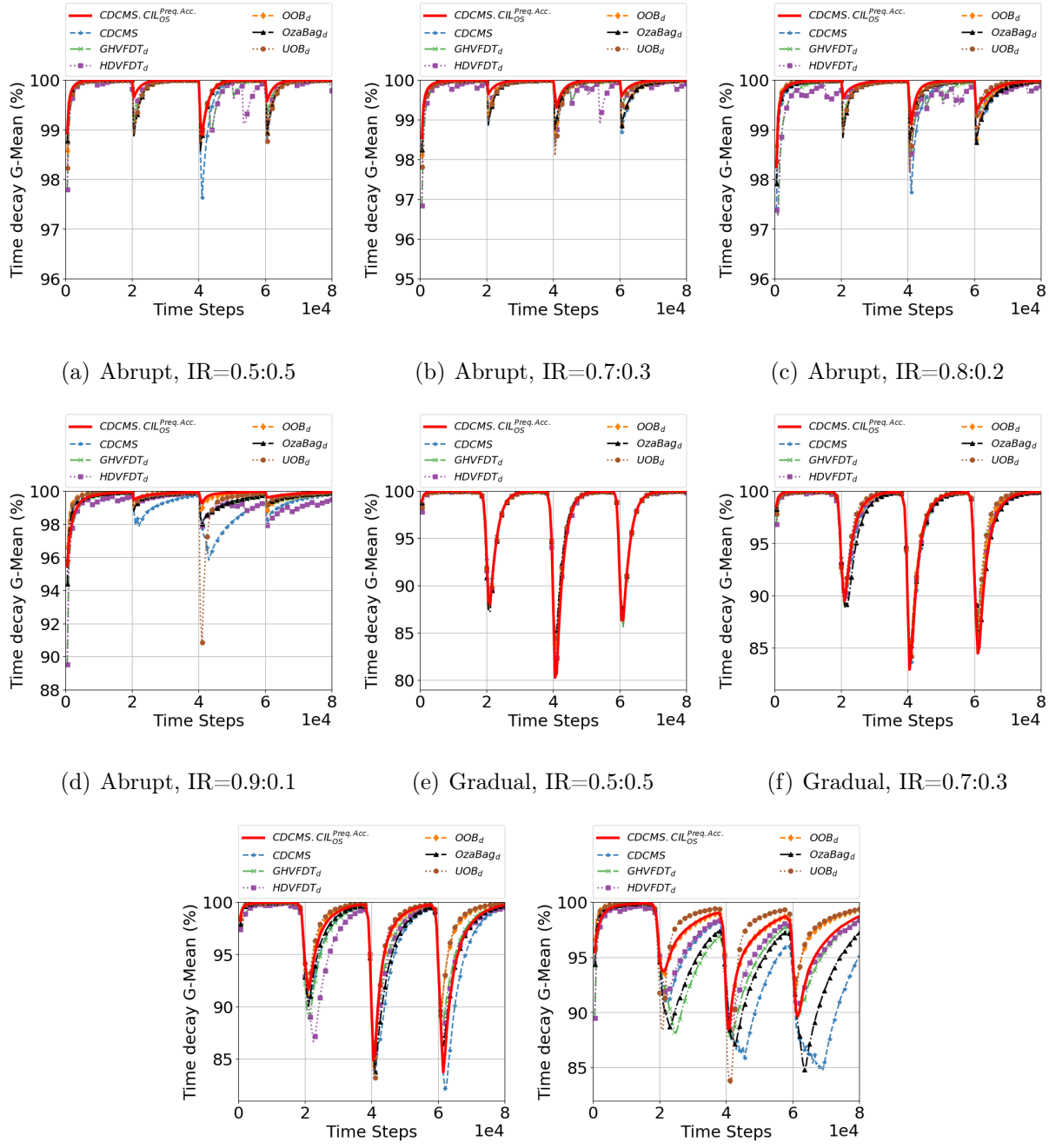


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure G.9: Comparing CDCMS.CIL against Existing Approaches: STAGGER1 (Time decay G-Mean)

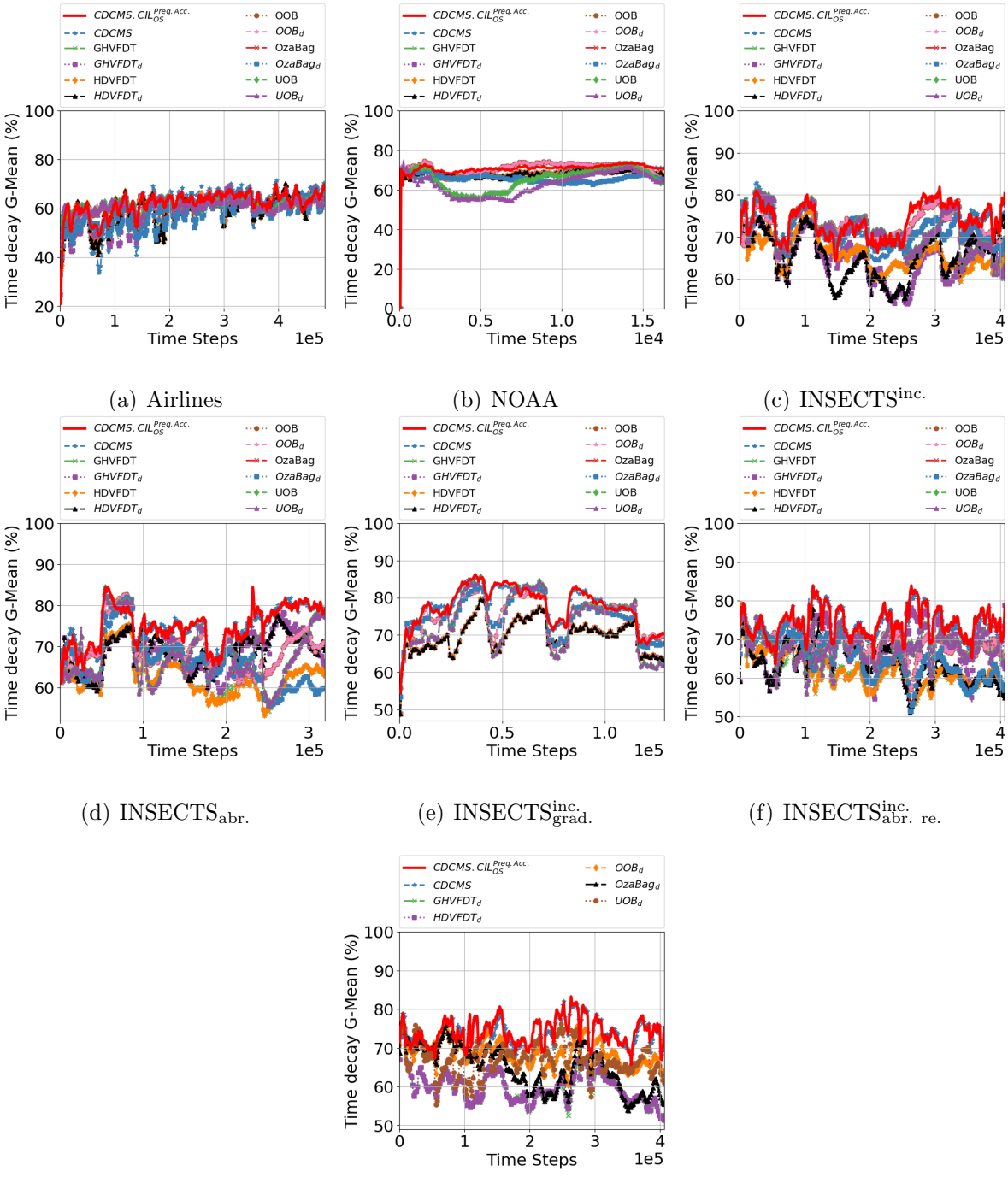


(a) Abrupt, IR=0.5:0.5 (b) Abrupt, IR=0.7:0.3 (c) Abrupt, IR=0.8:0.2

(d) Abrupt, IR=0.9:0.1 (e) Gradual, IR=0.5:0.5 (f) Gradual, IR=0.7:0.3

(g) Gradual, IR=0.8:0.2 (h) Gradual, IR=0.9:0.1

Figure G.10: Comparing CDCMS.CIL against Existing Approaches: STAGGER2 (Time decay G-Mean)

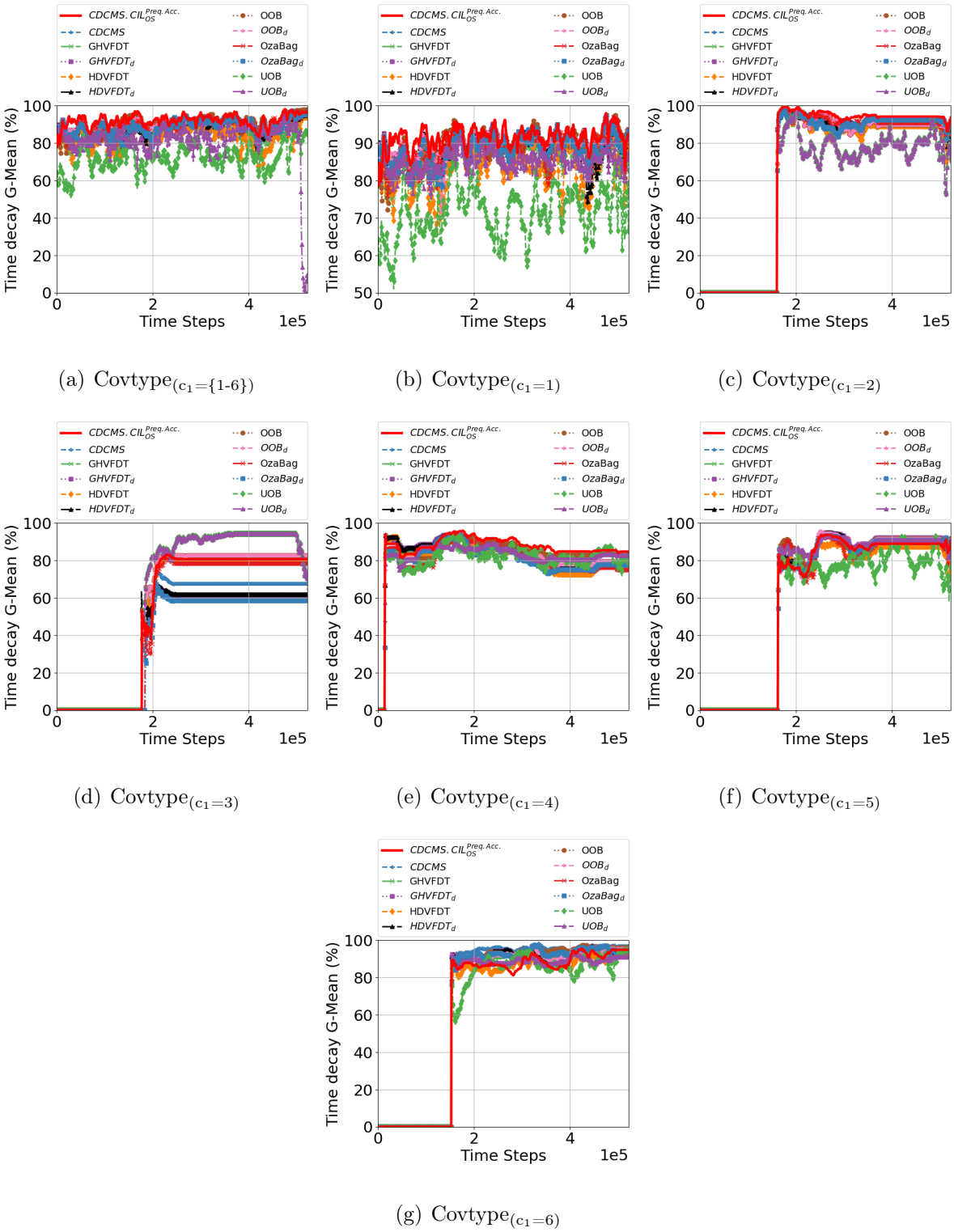


(a) Airlines (b) NOAA (c) INSECTS$^{inc.}$

(d) INSECTS$_{abr.}$ (e) INSECTS$^{inc.}_{grad.}$ (f) INSECTS$^{inc.}_{abr.\ re.}$

(g) INSECTS$^{inc.}_{re.}$

Figure G.11: Comparing CDCMS.CIL against Existing Approaches: Airlines, NOAA, and INSECTS Streams (Time decay G-Mean)



(a) Covtype$_{(c_1=\{1\text{-}6\})}$ (b) Covtype$_{(c_1=1)}$ (c) Covtype$_{(c_1=2)}$

(d) Covtype$_{(c_1=3)}$ (e) Covtype$_{(c_1=4)}$ (f) Covtype$_{(c_1=5)}$

(g) Covtype$_{(c_1=6)}$

Figure G.12: Comparing CDCMS.CIL against Existing Approaches: Covtype Streams (Time decay G-Mean)

Table G.1: Statistical (Friedman) Ranking of Time Decay G-Mean on Streams Grouped by Factors (Comparison against Existing Approaches with and without Employing a Drift Detector)

Groups	GH-VFDT	HD-VFDT	OzaBag	OOB	UOB	GH-VFDT$_d$	HD-VFDT$_d$	OzaBag$_d$	OOB$_d$	UOB$_d$	CDCMS	CDCMS.CIL$_{OS}^{Preq.\ Acc.}$
Grouped by imbalance ratio (Artificial data streams)												
0.5:0.5	10.6817	10.205	8.8633	8.5617	9.18	6.2167	6.3133	3.4983	4.0617	5.3917	3.055	1.9717
0.7:0.3	10.57	10.2317	10.1325	7.2833	7.9417	6.1383	6.3367	6.0775	2.6517	3.285	5.3667	1.985
0.8:0.2	10.1808	9.7942	10.61	6.395	6.8867	6.7467	7.2483	7.035	2.3283	2.5483	5.8933	2.3333
0.9:0.1	10.04	9.7842	10.3833	5.84	5.3167	7.4592	7.9283	6.5017	3.2	2.385	6.5433	2.6183
Grouped by concept drift speed												
Abrupt	10.4862	10.0271	9.8221	6.8342	7.5542	6.6417	6.9133	5.6104	3.1367	3.6692	5.3025	2.0025
Gradual	10.25	9.9804	10.1725	7.2058	7.1083	6.6388	7.0	5.9458	2.9842	3.1358	5.1267	2.4517
Real-World	9.0024	9.619	7.5667	3.4476	8.369	6.8952	7.1548	7.969	4.3429	6.2905	4.5786	2.7643
Grouped by streams												
Sine	11.0542	10.3667	11.1583	8.1208	9.2917	6.3125	6.6667	4.4542	2.1	3.1708	3.7417	1.5625
Agrawal	10.1875	11.4917	9.1875	6.9042	6.5	7.4542	7.7083	4.9292	4.0958	3.9917	4.4917	1.0583
SEA	8.5875	8.6958	8.8562	2.7125	4.0042	8.5875	8.6958	9.4438	2.8583	4.0042	7.8083	3.7458
STAGGER	11.6125	9.175	10.9708	8.7583	9.4833	5.2	6.1708	4.4458	2.5667	2.1542	5.025	2.4375
Airlines	10.7333	9.0	11.1333	4.7333	2.6	7.0	8.0	11.1333	3.8333	2.9	5.9333	1.0
NOAA	7.2	7.2	11.0	2.2667	10.0	7.2	7.2	11.0	2.2667	6.8	4.4	1.4667
Covtype	7.8	9.5667	6.8238	3.2667	10.1762	5.281	5.3333	7.6286	5.1857	6.4333	6.1857	4.319
INSECTS	10.7	10.3	7.2067	3.68	6.6667	9.0733	9.5267	7.2067	3.68	6.6667	2.0933	1.2
All streams	10.1349	9.928	9.6161	6.5571	7.4596	6.5498	6.8465	5.8746	3.2528	3.8248	5.1064	2.8493

- The p-values of Friedman tests are all $\leq$2.2E-16.

- Highlighted ranks denote significant superior performance.

Table G.2: Statistical (Friedman) Ranking of Time Decay Majority Class (Class 0) on Streams Grouped by Factors (Comparison against Existing Approaches with and without Employing a Drift Detector)

Groups	GH-VFDT	HD-VFDT	OzaBag	OOB	UOB	GH-VFDT$_d$	HD-VFDT$_d$	OzaBag$_d$	OOB$_d$	UOB$_d$	CDCMS	CDCMS.CIL$_{OS}^{Preq.\ Acc.}$
Grouped by imbalance ratio (Artificial data streams)												
0.5:0.5	9.1942	9.0158	9.8817	9.745	9.4383	5.585	5.625	4.0508	4.9692	5.2333	3.26	2.0017
0.7:0.3	7.9075	7.9242	6.3825	10.5883	11.7283	4.385	4.2233	1.6842	6.6883	9.02	2.2367	5.2317
0.8:0.2	7.1308	6.8108	6.2533	10.49	11.7967	4.3758	4.0292	1.3683	7.6367	9.87	2.5317	5.7067
0.9:0.1	6.6358	6.3317	5.4933	10.1967	11.9	4.2542	4.015	1.66	8.2233	10.2383	2.5883	6.4633
Grouped by concept drift speed												
Abrupt	7.8492	7.6375	6.9921	10.1658	11.2483	4.7679	4.6612	2.2467	6.7421	8.4725	2.8092	4.4075
Gradual	7.585	7.4038	7.0133	10.3442	11.1833	4.5321	4.285	2.135	7.0167	8.7083	2.4992	5.2942
Real-World	7.2521	9.05	3.3083	5.7958	10.7	6.2708	6.7333	4.0854	6.1542	9.5646	4.4604	4.625
Grouped by streams												
Sine	7.8125	8.7542	8.7458	10.3958	11.7	4.2938	4.3396	1.9667	5.9	8.2042	1.7333	4.1542
Agrawal	8.05	6.4417	7.7042	11.1083	10.8583	4.7417	3.8167	1.4833	7.35	9.1333	2.45	4.8625
SEA	5.2542	5.3458	4.4562	9.4083	10.5375	5.2542	5.3458	3.6979	9.7375	10.5375	2.05	6.375
STAGGER	8.9125	9.1833	9.6083	10.9708	11.2083	4.0167	4.575	2.575	4.2625	5.675	3.7625	3.25
Airlines	3.0	6.0	1.5	8.4667	11.6667	5.0	7.0	1.5	8.6333	11.3333	4.0	9.9
NOAA	10.5	10.5	1.5	6.2	6.5	10.5	10.5	1.5	6.2	7.0	3.0	4.1
Covtype	7.5762	10.1619	4.0714	4.6	11.8571	6.0476	6.5333	5.8476	5.3952	9.2381	4.1143	2.5571
INSECTS	6.6467	8.18	1.9067	6.2067	11.5	6.1	6.5	1.9067	6.2067	11.5	4.5333	6.8133
All streams	7.6396	7.7755	6.387	9.5118	11.1299	4.9201	4.8498	2.5066	6.7585	8.7528	2.9552	4.8132

- The p-values of Friedman tests are all $\leq$2.2E-16.

- Highlighted ranks denote significant superior performance.

Table G.3: Statistical (Friedman) Ranking of Time Decay Minority Class (Class 1) on Streams Grouped by Factors (Comparison against Existing Approaches with and without Employing a Drift Detector)

Groups	GH-VFDT	HD-VFDT	OzaBag	OOB	UOB	GH-VFDT$_d$	HD-VFDT$_d$	OzaBag$_d$	OOB$_d$	UOB$_d$	CDCMS	CDCMS.CIL$_{OS}^{Preq.\ Acc.}$
Grouped by imbalance ratio (Artificial data streams)												
0.5:0.5	10.7883	9.8033	8.2817	7.9783	9.22	5.8283	5.885	3.6142	4.1025	5.4267	4.02	3.0517
0.7:0.3	10.7983	10.35	10.0492	6.8833	5.3817	6.7808	7.0375	6.7183	3.2275	1.425	6.41	2.9383
0.8:0.2	10.3925	9.9658	10.3767	6.1783	4.1867	7.2725	7.7758	7.61	3.0583	1.34	6.6417	3.2017
0.9:0.1	10.2517	9.8175	10.3033	5.6783	3.3183	7.8258	8.095	6.9633	3.465	1.945	7.0067	3.33
Grouped by concept drift speed												
Abrupt	10.6129	10.0162	9.6312	6.5717	5.835	6.9458	7.2158	6.0062	3.5233	2.6492	6.0592	2.9333
Gradual	10.5025	9.9521	9.8742	6.7875	5.2183	6.9079	7.1808	6.4467	3.4033	2.4192	5.98	3.3275
Real-World	8.5271	9.15	7.8188	4.45	5.2229	7.1688	7.0396	8.1062	5.2979	4.5667	5.7479	4.9042
Grouped by streams												
Sine	11.3167	10.55	10.7	8.8208	6.9875	6.8417	7.1667	4.6917	2.6083	1.9792	4.0708	2.2667
Agrawal	10.4	11.5542	8.8833	4.8375	4.5833	7.975	8.175	6.3042	4.4917	2.9083	5.4875	2.4
SEA	8.9292	8.9458	7.6021	3.2792	2.8458	8.9292	8.9458	8.5271	3.2	2.8458	9.1375	4.8125
STAGGER	11.7875	9.5875	10.7417	8.6083	9.25	5.0042	6.0667	4.5771	2.7979	2.0625	5.075	2.4417
Airlines	10.0	8.0	11.5	4.8333	1.3	9.0	6.0	11.5	4.1667	1.7	7.0	3.0
NOAA	2.7	2.7	11.4667	5.7	9.0333	2.7	2.7	11.4667	5.7	7.7333	9.4	6.7
Covtype	8.2381	10.2333	7.0238	3.8905	6.8476	5.5429	5.8381	7.681	5.9238	5.4762	6.5952	4.7095
INSECTS	10.7	10.3	8.3	5.08	1.5	9.7267	9.5	8.3	5.08	1.5	4.4667	3.5467
All streams	10.2193	9.8451	9.4304	6.308	5.476	6.9672	7.1719	6.5398	3.7691	2.8729	5.9743	3.426

- The p-values of Friedman tests are all $\leq$2.2E-16.

- Highlighted ranks denote significant superior performance.

References

Abad, Miguel Ángel, João Bártolo Gomes, and Ernestina Menasalvas Ruiz (2016). "Predicting recurring concepts on data-streams by means of a meta-model and a fuzzy similarity function". In: *Expert Systems with Applications* 46, pp. 87–105.

Abdallah, Zahraa et al. (June 2016). "AnyNovel: detection of novel concepts in evolving data streams". In: *Evolving Systems* 7, pp. 73–93. DOI: 10.1007/s12530-016-9147-7.

Aggarwal, Charu C. et al. (2003). "A Framework for Clustering Evolving Data Streams". In: *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*. VLDB '03, pp. 81–92.

Agrawal, Rakesh, Tomasz Imielinski, and Arun Swami (1993). "Database mining: A performance perspective". In: *IEEE Transactions on Knowledge and Data Engineering* 5.6, pp. 914–925.

Bach, Stephen H. and Marcus A. Maloof (2008). "Paired Learners for Concept Drift". In: pp. 23–32. DOI: 10.1109/ICDM.2008.119.

Baena-Garc, Manuel et al. (2005). "Early Drift Detection Method". In.

Barros, Roberto, Juan Gonzalez Hidalgo, and Danilo Cabral (Jan. 2018). "Wilcoxon Rank Sum Test Drift Detector". In: *Neurocomputing* 275, pp. 1954–1963.

Batista, Gustavo E. A. P. A., Ronaldo Cristiano Prati, and Maria Carolina Monard (Sept. 2005). "Balancing Strategies and Class Overlapping". In: pp. 24–35. ISBN: 3-540-28795-7. DOI: 10.1007/11552253_3.

Bernardo, Alessio and Emanuele Della Valle (Nov. 2021). "VFC-SMOTE: very fast continuous synthetic minority oversampling for evolving data streams". In: *Data Mining and Knowledge Discovery* 35. DOI: 10.1007/s10618-021-00786-0.

Bernardo, Alessio, Heitor Murilo Gomes, et al. (2020). "C-SMOTE: Continuous Synthetic Minority Oversampling for Evolving Data Streams". In: pp. 483–492. DOI: 10.1109/BigData50022.2020.9377768.

Bifet, Albert and Ricard Gavaldà (Apr. 2007). "Learning from Time-Changing Data with Adaptive Windowing". In: vol. 7. DOI: 10.1137/1.9781611972771.42.

Bifet, Albert, Geoff Holmes, et al. (2010). "MOA: Massive Online Analysis". In: *Journal of Machine Learning Research* 11.52, pp. 1601–1604. URL: http://jmlr.org/papers/v11/bifet10a.html.

Bifet, Albert, Gianmarco Morales, et al. (Aug. 2015). "Efficient Online Evaluation of Big Data Stream Classifiers". In: pp. 59–68. DOI: 10.1145/2783258.2783372.

Blackard, Jock A. and Denis J. Dean (Dec. 1999). "Comparative Accuracies of Artificial Neural Networks and Discriminant Analysis in Predicting Forest Cover Types from Cartographic Variables". In: *Computers and Electronics in Agriculture* 24, pp. 131–151. DOI: 10.1016/S0168-1699(99)00046-0.

Bruzzone, L. and D. Fernández Prieto (Nov. 1999). "An Incremental—Learning Neural Network for the Classification of Remote—Sensing Images". In: *Pattern Recognition Letters* 20.11–13, pp. 1241–1248. ISSN: 0167-8655. DOI: 10.1016/S0167-8655(99)00091-4. URL: https://doi.org/10.1016/S0167-8655(99)00091-4.

Brzezinski, Dariusz, Leandro L. Minku, et al. (2021). "The impact of data difficulty factors on classification of imbalanced and concept drifting data streams". In: *Knowl. Inf. Syst.* 63, pp. 1429–1469.

Brzezinski, Dariusz and Jerzy Stefanowski (Sept. 2014). "Prequential AUC for Classifier Evaluation and Drift Detection in Evolving Data Streams". In: vol. 8983. DOI: 10.1007/978-3-319-17876-9_6.

Brzezinski, Dariusz and Jerzy Stefanowski (Aug. 2017). "Prequential AUC: Properties of the Area Under the ROC Curve for Data Streams with Concept Drift". In: *Knowledge and Information Systems* 52, pp. 531–562. DOI: 10.1007/s10115-017-1022-8.

Brzeziński, Dariusz and Jerzy Stefanowski (May 2011). "Accuracy Updated Ensemble for Data Streams with Concept Drift". In: vol. 6679, pp. 155–163. ISBN: 978-3-642-21221-5. DOI: 10.1007/978-3-642-21222-2_19.

— (May 2014). "Combining Block-based and Online Methods in Learning Ensembles from Concept Drifting Data Streams". In: *Information Sciences* 265, pp. 50–67.

Cabral, George G. et al. (May 2019). "Class Imbalance Evolution and Verification Latency in Just-in-Time Software Defect Prediction". In: *International Conference on Software Engineering*, pp. 666–676.

Chawla, Nitesh et al. (Jan. 2003). "SMOTEBoost: Improving Prediction of the Minority Class in Boosting". In: vol. 2838, pp. 107–119. ISBN: 978-3-540-20085-7. DOI: 10.1007/978-3-540-39804-2_12.

Chawla, Nitesh V. et al. (June 2002). "SMOTE: Synthetic Minority over-Sampling Technique". In: *Journal of Artificial Intelligence Research* 16.1, pp. 321–357. ISSN: 1076-9757.

Chen, Sheng and Haibo He (2009). "SERA: Selectively recursive approach towards nonstationary imbalanced stream data mining". In: pp. 522–529. DOI: 10.1109/IJCNN.2009.5178874.

— (Mar. 2011). "Toward incremental learning of nonstationary imbalanced data stream: A multiple selectively recursive approach". In: *Evolving Systems* 2, pp. 35–50. DOI: 10.1007/s12530-010-9021-y.

Chiu, Chun Wai and Leandro L. Minku (2018). "Diversity-Based Pool of Models for Dealing with Recurring Concepts". In: *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN.2018.8489190.

Chiu, Chun Wai and Leandro L. Minku (2022). "A Diversity Framework for Dealing With Multiple Types of Concept Drift Based on Clustering in the Model Space". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.3, pp. 1299–1309. DOI: 10.1109/TNNLS.2020.3041684.

Cieslak, David A. and Nitesh V. Chawla (2008). "Learning Decision Trees for Unbalanced Data". In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Walter Daelemans, Bart Goethals, and Katharina Morik. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 241–256. ISBN: 978-3-540-87479-9.

Cieslak, David A., Thomas Ryan Hoens, et al. (Jan. 2012). "Hellinger distance decision trees are robust and skew-insensitive". In: *Data Min. Knowl. Discov.* 24, pp. 136–158. DOI: 10.1007/s10618-011-0222-1.

Cohen, Lior et al. (2008). "Real-time data mining of non-stationary data streams from sensor networks". In: *Information Fusion* 9.3. Special Issue on Distributed Sensor Networks, pp. 344–353. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2005.05.005. URL: https://www.sciencedirect.com/science/article/pii/S1566253505000655.

Dal Pozzolo, Andrea et al. (Sept. 2017). "Credit Card Fraud Detection: A Realistic Modelling and a Novel Learning Strategy". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14.

Datar, Mayur et al. (Sept. 2002). "Maintaining Stream Statistics over Sliding Windows". In: *SIAM J. Comput.* 31, pp. 1794–1813. DOI: 10.1137/S0097539701398363.

Delany, Sarah Jane et al. (Aug. 2005). "A Case-Based Technique for Tracking Concept Drift in Spam Filtering". In: *Knowledge-Based Systems* 18, pp. 187–195.

Dempster, Arthur P., Nan M. Laird, and Donald B. Rubin (Jan. 1977). "Maximum Likelihood from Incomplete Data via the EM algorithm". In: *Journal of the Royal Statistical Socieety. Series B* 39, pp. 1–38.

Ditzler, Gregory and Robi Polikar (2013). "Incremental Learning of Concept Drift from Streaming Imbalanced Data". In: *IEEE Transactions on Knowledge and Data Engineering* 25.10, pp. 2283–2301. DOI: 10.1109/TKDE.2012.136.

Ditzler, Gregory, Manuel Roveri, et al. (Nov. 2015). "Learning in Nonstationary Environments: A Survey". In: *IEEE Computational Intelligence Magazine* 10, pp. 12–25.

Domingos, Pedro and Geoff Hulten (2000). "Mining High-Speed Data Streams". In: KDD '00. Boston, Massachusetts, USA: Association for Computing Machinery, pp. 71–80. ISBN: 1581132336. DOI: 10.1145/347090.347107. URL: https://doi.org/10.1145/347090.347107.

Duda, Richard O., Peter E. Hart, and David G. Stork (2001). *Pattern classification, 2nd Edition*. Wiley. ISBN: 9780471056690. URL: https://www.worldcat.org/oclc/41347061.

Elwell, Ryan and Robi Polikar (2011). "Incremental Learning of Concept Drift in Nonstationary Environments". In: *IEEE Transactions on Neural Networks* 22.10, pp. 1517–1531. DOI: 10.1109/TNN.2011.2160459.

Escovedo, Tatiana et al. (Oct. 2017). "DetectA: Abrupt Concept Drift Detection in Non-stationary Environments". In: *Applied Soft Computing* 62.

Ester, Martin et al. (1996). "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Proc. of 2nd International Conference on Knowledge Discovery and*, pp. 226–231.

Fawcett, Tom (Jan. 2004). "ROC Graphs: Notes and Practical Considerations for Researchers". In: *Machine Learning* 31, pp. 1–38.

— (June 2006). "Introduction to ROC analysis". In: *Pattern Recognition Letters* 27, pp. 861–874. DOI: 10.1016/j.patrec.2005.10.010.

Ferrer-Troyano, Francisco, Jesus S. Aguilar-Ruiz, and Jose C. Riquelme (2005). "Incremental Rule Learning Based on Example Nearness from Numerical Data Streams". In: SAC '05. Santa Fe, New Mexico: Association for Computing Machinery, pp. 568–572. ISBN:

1581139640. DOI: 10.1145/1066677.1066808. URL: https://doi.org/10.1145/1066677.1066808.

Frey, Brendan J. and Delbert Dueck (Mar. 2007). "Clustering by Passing Messages Between Data Points". In: *Science (New York, N.Y.)* 315, pp. 972–6. DOI: 10.1126/science.1136800.

Frías-Blanco, Isvani et al. (2015). "Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds". In: *IEEE Transactions on Knowledge and Data Engineering* 27.3, pp. 810–823. DOI: 10.1109/TKDE.2014.2345382.

Gama, João and Petr Kosina (Oct. 2009). "Tracking Recurring Concepts with Meta-learners". In: pp. 423–434. ISBN: 978-3-642-04685-8. DOI: 10.1007/978-3-642-04686-5_35.

Gama, João, Pedro Medas, et al. (Sept. 2004). "Learning with Drift Detection". In: vol. 8, pp. 286–295. ISBN: 978-3-540-23237-7. DOI: 10.1007/978-3-540-28645-5_29.

Gama, João, Rqauel Sebastião, and Pedro Pereira Rodrigues (Oct. 2013). "On Evaluating Stream Learning Algorithms". In: *Machine Learning* 90, pp. 317–346.

Gama, João, Indrė Žliobaitė, et al. (Mar. 2014). "A Survey on Concept Drift Adaptation". In: *ACM Computing Survey* 46.4. ISSN: 0360-0300. DOI: 10.1145/2523813. URL: https://doi.org/10.1145/2523813.

Gao, Jing et al. (Apr. 2007). "A General Framework for Mining Concept-Drifting Data Streams with Skewed Distributions". In: DOI: 10.1137/1.9781611972771.1.

Ghazikhani, Adel, Reza Monsefi, and Hadi sadoghi yazdi (Dec. 2013). "Ensemble of online neural networks for non-stationary and imbalanced data streams". In: *Neurocomputing* 122, pp. 535–544. DOI: 10.1016/j.neucom.2013.05.003.

— (Feb. 2014). "Online neural network model for non-stationary and imbalanced data stream classification". In: *International Journal of Machine Learning and Cybernetics* 5, pp. 51–62. DOI: 10.1007/s13042-013-0180-6.

Giacinto, Giorgio and Fabio Roli (2001). "Design of effective neural network ensembles for image classification purposes". In: *Image Vis. Comput.* 19, pp. 699–707.

Gomes, João Bártolo et al. (2014). "Mining Recurring Concepts in a Dynamic Feature Space". In: *IEEE Transactions on Neural Networks and Learning Systems* 25, pp. 95–110.

Gonçalves Jr, Paulo et al. (Dec. 2014). "A Comparative Study on Concept Drift Detectors". In: *Expert Systems with Applications* 41, pp. 8144–8156. DOI: 10.1016/j.eswa.2014.07.019.

Gonçalves Jr, Paulo Mauricio and Roberto Barros (July 2013). "RCD: A Recurring Concept Drift Framework". In: *Pattern Recognition Letters* 34, pp. 1018–1025.

Han, Hui, Wen-Yuan Wang, and Bing-Huan Mao (Sept. 2005). "Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning". In: vol. 3644, pp. 878–887. ISBN: 978-3-540-28226-6. DOI: 10.1007/11538059_91.

Hansen, L.K. and Peter Salamon (Nov. 1990). "Neural Network Ensembles". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 12, pp. 993–1001. DOI: 10.1109/34.58871.

Hartigan, J. A. and M. A. Wong (1979). "A k-means clustering algorithm". In: *JSTOR: Applied Statistics* 28.1, pp. 100–108.

Hastie, Trevor and Werner Stuetzle (1989). "Principal Curves". In: *Journal of the American Statistical Association* 84.406, pp. 502–516. DOI: 10.1080/01621459.1989.10478797.

He, Haibo, Yang Bai, et al. (2008). "ADASYN: Adaptive synthetic sampling approach for imbalanced learning". In: pp. 1322–1328. DOI: 10.1109/IJCNN.2008.4633969.

He, Haibo and Edwardo A. Garcia (2009). "Learning from Imbalanced Data". In: *IEEE Transactions on Knowledge and Data Engineering* 21.9, pp. 1263–1284. DOI: 10.1109/TKDE.2008.239.

Hoens, Thomas Ryan and Nitesh V. Chawla (2012). "Learning in Non-Stationary Environments with Class Imbalance". In: KDD '12. Beijing, China: Association for Computing Machinery, pp. 168–176. ISBN: 9781450314626. DOI: 10.1145/2339530.2339558. URL: https://doi.org/10.1145/2339530.2339558.

Hoens, Thomas Ryan, Nitesh V. Chawla, and Robi Polikar (2011). "Heuristic Updatable Weighted Random Subspaces for Non-stationary Environments". In: pp. 241–250. DOI: 10.1109/ICDM.2011.75.

Hoens, Thomas Ryan, Robi Polikar, and Nitesh V. Chawla (2011). "Learning from streaming data with concept drift and imbalance: an overview". In: *Progress in Artificial Intelligence* 1, pp. 89–101.

Hulten, Geoff, Laurie Spencer, and Pedro Domingos (2001). "Mining Time-Changing Data Streams". In: KDD '01. San Francisco, California: Association for Computing Machinery, pp. 97–106. ISBN: 158113391X. DOI: 10.1145/502512.502529. URL: https://doi.org/10.1145/502512.502529.

Ikonomovska, Elena, João Gama, and Sašo Džeroski (July 2011). "Learning model trees from evolving data streams". In: *Data Mining and Knowledge Discovery* 23, pp. 128–168. DOI: 10.1007/s10618-010-0201-y.

Japkowicz, Nathalie (2001). "Concept-Learning in the Presence of Between-Class and Within-Class Imbalances". In: *Canadian Conference on AI*.

Japkowicz, Nathalie and Shaju Stephen (Oct. 2002). "The Class Imbalance Problem: A Systematic Study". In: *Intelligent Data Analysis* 6.5, pp. 429–449. ISSN: 1088-467X.

Jo, Taeho and Nathalie Japkowicz (June 2004). "Class Imbalances versus Small Disjuncts". In: *ACM SIGKDD Explorations Newsletter* 6.1, pp. 40–49. ISSN: 1931-0145. DOI: 10.1145/1007730.1007737. URL: https://doi.org/10.1145/1007730.1007737.

Kadlec, Petr, Bogdan Gabrys, and Sibylle Strandt (Apr. 2009). "Data-Driven Soft Sensors in the Process Industry". In: *Computers and Chemical Engineering* 33, pp. 795–814.

Kelly, Mark G., David J. Hand, and Niall M. Adams (1999). "The Impact of Changing Populations on Classifier Performance". In: KDD '99. San Diego, California, USA: Association for Computing Machinery, pp. 367–371. ISBN: 1581131437. DOI: 10.1145/312129.312285. URL: https://doi.org/10.1145/312129.312285.

Kolter, J. Zico and Marcus A. Maloof (Dec. 2003). "Dynamic Weighted Majority: A New Ensemble Method for Tracking Concept Drift". In: *IEEE International Conference on Data Mining*, pp. 123–130.

Kosina, Petr, João Gama, and Raquel Sebastião (2010). "Drift Severity Metric". In: NLD: IOS Press, pp. 1119–1120. ISBN: 9781607506058.

Kranen, Philipp et al. (Nov. 2011). "The ClusTree: Indexing micro-clusters for anytime stream mining". In: *Knowl. Inf. Syst.* 29, pp. 249–272. DOI: 10.1007/s10115-010-0342-8.

Krawczyk, Bartosz et al. (Sept. 2017). "Ensemble Learning for Data Stream Analysis: A Survey". In: *Information Fusion* 37, pp. 132–156.

Kubat, Miroslav and Stan Matwin (1997). "Addressing the Curse of Imbalanced Training Sets: One-Sided Selection". In: ed. by Douglas H. Fisher. Morgan Kaufmann, pp. 179–186.

Kuncheva, Ludmila I. and Christopher J. Whitaker (May 2003). "Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy". In: *Machine Learning* 51, pp. 181–207.

Lee, Hansoo, Jonggeun Kim, and Sungshin Kim (Dec. 2017). "Gaussian-Based SMOTE Algorithm for Solving Skewed Class Distributions". In: vol. 17, pp. 229–234. DOI: 10.5391/IJFIS.2017.17.4.229.

Li, Cen (Jan. 2007). "Classifying imbalanced data using a bagging ensemble variation (BEV)". In: pp. 203–208. DOI: 10.1145/1233341.1233378.

Liang, Nan-ying et al. (2006). "A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks". In: *IEEE Transactions on Neural Networks* 17.6, pp. 1411–1423. DOI: 10.1109/TNN.2006.880583.

Littlestone, Nick and Manfred K. Warmuth (Feb. 1994). "The Weighted Majority Algorithm". In: *Information and Computation* 108.2, pp. 212–261. ISSN: 0890-5401. DOI: 10.1006/inco.1994.1009. URL: https://doi.org/10.1006/inco.1994.1009.

Lu, Yang, Yiu-ming Cheung, and Yuan Yan Tang (2017). "Dynamic Weighted Majority for Incremental Learning of Imbalanced Data Streams with Concept Drift". In: IJCAI'17. Melbourne, Australia: AAAI Press, pp. 2393–2399. ISBN: 9780999241103.

Lyon, Rob et al. (May 2014). "Hellinger Distance Trees for Imbalanced Streams". In: DOI: 10.1109/ICPR.2014.344.

Mao, Wentao, Jinwan Wang, and Liyun Wang (2015). "Online sequential classification of imbalanced data by combining extreme learning machine and improved SMOTE algorithm". In: pp. 1–8. DOI: 10.1109/IJCNN.2015.7280620.

Minku, Leandro L. (Jan. 2019). "Transfer Learning in Non-stationary Environments: Methods and Applications". In: pp. 13–37. ISBN: 978-3-319-89802-5. DOI: 10.1007/978-3-319-89803-2_2.

Minku, Leandro L. and Siqing Hou (2017). "Clustering Dycom: An Online Cross-Company Software Effort Estimation Study". In: PROMISE. Toronto, Canada: Association for Computing Machinery, pp. 12–21. ISBN: 9781450353052. DOI: 10.1145/3127005.3127007. URL: https://doi.org/10.1145/3127005.3127007.

Minku, Leandro L., Allan P. White, and Xin Yao (2010). "The Impact of Diversity on Online Ensemble Learning in the Presence of Concept Drift". In: *IEEE Transactions on Knowledge and Data Engineering* 22.5, pp. 730–742. DOI: 10.1109/TKDE.2009.156.

Minku, Leandro L. and Xin Yao (May 2012). "DDD: A New Ensemble Approach for Dealing with Concept Drift". In: *IEEE Transactions on Knowledge and Data Engineering* 24, pp. 619–633.

Mirza, Bilal, Zhiping Lin, and Nan Liu (2015). "Ensemble of subset online sequential extreme learning machine for class imbalance and concept drift". In: *Neurocomputing* 149. Advances in neural networks Advances in Extreme Learning Machines, pp. 316–329. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2014.03.075. URL: https://www.sciencedirect.com/science/article/pii/S0925231214011448.

Mirza, Bilal, Zhiping Lin, and Kar-Ann Toh (Dec. 2013). "Weighted Online Sequential Extreme Learning Machine for Class Imbalance Learning". In: *Neural Processing Letters* 38. DOI: 10.1007/s11063-013-9286-9.

Mitchell, Tom Michael (1997). *Machine learning*. eng. London.

Moreno-Torres, Jose G. et al. (Jan. 2012). "A Unifying View on Dataset Shift in Classification". In: *Pattern Recognition* 45.1, pp. 521–530. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2011.06.019. URL: https://doi.org/10.1016/j.patcog.2011.06.019.

Moulton, Richard et al. (Jan. 2019). "Clustering in the Presence of Concept Drift". In: pp. 339–355. ISBN: 978-981-13-6048-0. DOI: 10.1007/978-3-030-10925-7_21.

Muller, Mervin E. (Apr. 1959). "A Note on a Method for Generating Points Uniformly on n-Dimensional Spheres". In: *Communications of the ACM* 2.4, pp. 19–20. ISSN: 0001-0782. DOI: 10.1145/377939.377946. URL: https://doi.org/10.1145/377939.377946.

Napierala, Krystyna and Jerzy Stefanowski (2012). "Identification of Different Types of Minority Class Examples in Imbalanced Data". In: HAIS'12. Salamanca, Spain: Springer-Verlag, pp. 139–150. ISBN: 9783642289309. DOI: 10.1007/978-3-642-28931-6_14. URL: https://doi.org/10.1007/978-3-642-28931-6_14.

— (July 2015). "Types of minority class examples and their influence on learning classifiers from imbalanced data". In: *Journal of Intelligent Information Systems* 46.3, pp. 563–597. DOI: 10.1007/s10844-015-0368-1.

Napierała, Krystyna, Jerzy Stefanowski, and Szymon Wilk (2010). "Learning from Imbalanced Data in Presence of Noisy and Borderline Examples". In: RSCTC'10. Warsaw, Poland: Springer-Verlag, pp. 158–167. ISBN: 3642135285.

Nishida, Kyosuke and Koichiro Yamauchi (Oct. 2007). "Detecting Concept Drift Using Statistical Testing". In: pp. 264–269. ISBN: 978-3-540-75487-9. DOI: 10.1007/978-3-540-75488-6_27.

Oliveira, Gustavo, Leandro L. Minku, and Adriano L I Oliveira (2021). "Tackling Virtual and Real Concept Drifts: An Adaptive Gaussian Mixture Model Approach". In: *IEEE*

Transactions on Knowledge and Data Engineering, pp. 1–1. DOI: 10.1109/TKDE.2021.3099690.

Oza, Nikunj C. (2005). "Online bagging and boosting". In: vol. 3, 2340–2345 Vol. 3. DOI: 10.1109/ICSMC.2005.1571498.

Page, E. S. (June 1954). "Continuous Inspection Schemes". In: *Biometrika* 41.1-2, pp. 100–115. ISSN: 0006-3444. DOI: 10.1093/biomet/41.1-2.100. URL: https://doi.org/10.1093/biomet/41.1-2.100.

Pearson, Karl (n.d.). "LIII. On lines and planes of closest fit to systems of points in space". In: *Philosophical Magazine Series 1* 2 (), pp. 559–572.

Pesaranghader, Ali and Herna L. Viktor (2016). "Fast Hoeffding Drift Detection Method for Evolving Data Streams". In: *ECML/PKDD*, pp. 96–111.

Polikar, Robi et al. (2001). "Learn++: an incremental learning algorithm for supervised neural networks". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 31.4, pp. 497–508. DOI: 10.1109/5326.983933.

Prati, Ronaldo Cristiano, Gustavo E. A. P. A. Batista, and Maria Carolina Monard (Jan. 2004). "Class Imbalances versus Class Overlapping: An Analysis of a Learning System Behavior". In: pp. 312–321. ISBN: 978-3-540-21459-5. DOI: 10.1007/978-3-540-24694-7_32.

Ramamurthy, Sasthakumar and Raj Bhatnagar (2007). "Tracking recurrent concept drift in streaming data using ensemble classifiers". In: pp. 404–409. DOI: 10.1109/ICMLA.2007.80.

Ross, Gordon J. et al. (Jan. 2012). "Exponentially Weighted Moving Average Charts for Detecting Concept Drift". In: *Pattern Recogn. Lett.* 33.2, pp. 191–198. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2011.08.019. URL: https://doi.org/10.1016/j.patrec.2011.08.019.

Schlimmer, Jeffrey C. and Richard H. Granger Jr. (Sept. 1986). "Incremental Learning from Noisy Data". In: *Machine Learning* 1, pp. 317–354.

Silva, Jonathan A. et al. (Oct. 2013). "Data Stream Clustering: A Survey". In: *ACM Computing Surveys* 46.1. ISSN: 0360-0300. DOI: 10.1145/2522968.2522981. URL: https://doi.org/10.1145/2522968.2522981.

Sobhani, Parinaz and Hamid Beigy (Jan. 2011). "New Drift Detection Method for Data Streams". In: pp. 88–97. ISBN: 978-3-642-23856-7. DOI: 10.1007/978-3-642-23857-4_12.

Song, Liyan, Leandro L. Minku, and Xin Yao (2018). "A Novel Automated Approach for Software Effort Estimation Based on Data Augmentation". In: ESEC/FSE 2018. Lake Buena Vista, FL, USA: Association for Computing Machinery, pp. 468–479. ISBN: 9781450355735. DOI: 10.1145/3236024.3236052. URL: https://doi.org/10.1145/3236024.3236052.

Souza, Vinicius M. A. et al. (2020). "Challenges in Benchmarking Stream Learning Algorithms with Real-world Data". In: *Data Mining and Knowledge Discovery* 34, pp. 1805–1858. DOI: 10.1007/s10618-020-00698-5.

Street, W. Nick and YongSeog Kim (2001). "A Streaming Ensemble Algorithm (SEA) for Large-scale Classification". In: pp. 377–382.

Theeramunkong, Thanaruk et al., eds. (2009). *Advances in Knowledge Discovery and Data Mining, 13th Pacific-Asia Conference, PAKDD 2009*. Vol. 5476. Lecture Notes in Computer Science. Springer. ISBN: 978-3-642-01306-5. DOI: 10.1007/978-3-642-01307-2. URL: https://doi.org/10.1007/978-3-642-01307-2.

Tomek, Ivan (1976). "Two Modifications of CNN". In: *IEEE Transactions on Systems, Man, and Cybernetics* 7(2), pp. 679–772.

Utgoff, Paul E. (Nov. 1989). "Incremental Induction of Decision Trees". In: *Machine Learning* 4.2, pp. 161–186. ISSN: 0885-6125. DOI: 10.1023/A:1022699900025. URL: https://doi.org/10.1023/A:1022699900025.

Van Rijsbergen, Cornelis Joost (1979). *Information Retrieval*.

Vargha, Andrãs and Harold D. Delaney (2000). "A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong". In: *Journal of Educational and Behavioral Statistics* 25.2, pp. 101–132. DOI: 10.3102/10769986025002101.

Wang, Boyu and Joelle Pineau (2016). "Online Bagging and Boosting for Imbalanced Data Streams". In: *IEEE Transactions on Knowledge and Data Engineering* 28.12, pp. 3353–3366. DOI: 10.1109/TKDE.2016.2609424.

Wang, Heng and Zubin Abraham (2015). "Concept drift detection for streaming data". In: pp. 1–9. DOI: 10.1109/IJCNN.2015.7280398.

Wang, Shuo, Huanhuan Chen, and Xin Yao (2010). "Negative correlation learning for classification ensembles". In: pp. 1–8. DOI: 10.1109/IJCNN.2010.5596702.

Wang, Shuo and Leandro L. Minku (2020). "AUC Estimation and Concept Drift Detection for Imbalanced Data Streams with Multiple Classes". In: pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9207377.

Wang, Shuo, Leandro L. Minku, Davide Ghezzi, et al. (2013). "Concept drift detection for online class imbalance learning". In: pp. 1–10. DOI: 10.1109/IJCNN.2013.6706768.

Wang, Shuo, Leandro L. Minku, and Xin Yao (2015). "Resampling-Based Ensemble Methods for Online Class Imbalance Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 27.5, pp. 1356–1368. DOI: 10.1109/TKDE.2014.2345380.

— (2018). "A Systematic Study of Online Class Imbalance Learning With Concept Drift". In: *IEEE Transactions on Neural Networks and Learning Systems* 29.10, pp. 4802–4821. DOI: 10.1109/TNNLS.2017.2771290.

Wang, Shuo and Xin Yao (2009). "Diversity analysis on imbalanced data sets by using ensemble models". In: pp. 324–331. DOI: 10.1109/CIDM.2009.4938667.

Webb, Geoffrey I. et al. (July 2016). "Characterizing Concept Drift". In: *Data Mining and Knowledge Discovery* 30.4, pp. 964–994. ISSN: 1384-5810. DOI: 10.1007/s10618-015-0448-4. URL: https://doi.org/10.1007/s10618-015-0448-4.

Yu, Shujian et al. (2019). *Concept Drift Detection and Adaptation with Hierarchical Hypothesis Testing*. arXiv: 1707.07821 [stat.ML].

Yule, George Udny (1900). "On the Association of Attributes in Statistics". In: *Philosophical Transactions* A.194, pp. 257–319.

Zhang, Kun et al. (Dec. 2006). "Forecasting Skewed Biased Stochastic Ozone Days: Analyses and Solutions". In: vol. 14, pp. 753–764. DOI: 10.1007/s10115-007-0095-1.

Zhang, Tian, Raghu Ramakrishnan, and Miron Livny (June 1997). "BIRCH: A New Data Clustering Algorithm and Its Applications". In: *Data Mining and Knowledge Discovery* 1.2, pp. 141–182. ISSN: 13845810. DOI: 10.1023/a:1009783824328. URL: http://dx.doi.org/10.1023/a:1009783824328.

Zhu, Xingquan (n.d.). *Stream Data Mining Repository*. http://www.cse.fau.edu/~xqzhu/stream.html. Accessed: 2019-05-30.

Žliobaitė, Indrė (Jan. 2010). "Learning under Concept Drift: an Overview". In: *CoRR* abs/1010.4784.

— (June 2011). "Combining similarity in time and space for training set formation under concept drift". In: *Intelligent Data Analysis* 15, pp. 589–611. DOI: 10.3233/IDA-2011-0484.