Sequential extreme learning machines for class imbalance and concept drift

Mirza Bilal

2015

Mirza Bilal. (2015). Sequential extreme learning machines for class imbalance and concept drift. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/65290>

<https://doi.org/10.32657/10356/65290>

Sequential Extreme Learning Machines
for
Class Imbalance and Concept Drift

Mirza Bilal

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2015

ACKNOWLEDGEMENTS

This research work received both financial and technical supports from the School of Electrical & Electronic Engineering, Nanyang Technological University.

First and foremost, I would like to express my deepest gratitude to my supervisor, Associate Professor Lin Zhiping for his professional guidance, patience and encouragement throughout my research work. I would like to take this opportunity to thank Prof. Toh Kar-Ann from Yonsei University, Seoul, Korea for his collaboration concerning the work presented in Chapter 2 and Dr. Liu Nan from Singapore General Hospital for his collaboration concerning the work presented in Chapter 3. I would also like to thank the laboratory technicians at Valens Centre of Excellence Lab.

Second, I would like to thank my friends and fellow classmates in Singapore for their support and help.

Finally, I would like to thank my parents whose love and support have always been the greatest inspiration for me.

Table of Contents

Acknowledgements	2
Summary	5
List of Tables.....	7
List of Figures	8
1. Introduction	10
1.1 Background and Motivation	10
1.2 Objectives.....	15
1.3 Contributions.....	16
1.4 Literature Review on Class Imbalance and Concept Drift Learning	18
1.4.1 Class Imbalance Learning	18
1.4.2 Concept Drift Learning	21
2. Weighted Online Sequential Extreme Learning Machine for Binary Class Imbalance Learning without Concept Drift	25
2.1 Introduction	25
2.2 Preliminaries.....	27
2.2.1 ELM and Online Sequential ELM Methods.....	27
2.2.2 Total Error Rate Methods.....	30
2.3 Proposed Method.....	33
2.3.1 Weight Selection in Weighted ELM for Class Imbalance Learning	34
2.3.2 Weighted Online Sequential Extreme Learning Machine.....	37
2.4 Experiments.....	43
2.4.1 Datasets	43
2.4.2 Experimental Settings	44
2.4.3 Classification Results	45
2.4.4 Statistical Significance Test	47
2.4.5 Summary of Experimental Results and Discussion.....	49
2.5 Conclusion.....	54
3. Ensemble of Subset Online Sequential Extreme Learning Machine for Binary Class Imbalance and Concept Drift.....	55
3.1 Introduction	55
3.2 Proposed Method.....	57
3.2.1 The Main Ensemble	57
3.2.2 ELM-Store for Saving Old Information.....	61
3.2.3 Change Detector.....	64
3.3 Experiments.....	69
3.3.1 Classification Results on Benchmark Imbalanced Datasets with Concept Drift.....	69
3.3.2 Classification Results on Benchmark Imbalanced Datasets without Concept Drift	81

3.3.3 Discussion on Results and Computational Complexity	87
3.4 Conclusion.....	88
4. Meta-Cognitive Online Sequential Extreme Learning Machine for Binary Class Imbalance and Concept Drift	89
4.1 Introduction	89
4.2 Preliminaries.....	91
4.2.1 Meta-Cognitive Learning & Knowledge Measures	91
4.3 Proposed Method.....	93
4.3.1 Hybrid Approach for Class Imbalance Learning	94
4.3.2 Change Detection and Adaptive Window for Concept Drift Learning	103
4.3.3 A Unifying Equation for Class Imbalance and Concept Drift.....	106
4.4 Experiments.....	112
4.4.1 MOS-ELM Performance Comparison on Imbalanced Datasets.....	112
4.4.2 MOS-ELM Performance Comparison on Non-Stationary Datasets.....	116
4.4.3 MOS-ELM Performance Comparison on Non-Stationary Datasets with Imbalanced Classes	119
4.5 Discussion	121
5. Imbalance Problem in Multiclass Sequential Learning	123
5.1 Introduction	123
5.1.1 Multiclass Imbalance Learning with Stationary Datasets	123
5.1.2 Multiclass Imbalance Learning with Concept Drift	125
5.2 Proposed Method.....	126
5.3 Experiments.....	130
5.4 Discussion	135
6. Conclusions and Future Works.....	136
6.1 Conclusions	136
6.2 Future Works.....	138
6.2.1 Addition of New Classes in Non-Stationary Learning.....	138
6.2.2 Class Imbalance Problem for Semi-supervised Learning.....	139
6.2.3 Online Sequential Learning using Deep Architectures	139
6.2.4 Optimization based OS-ELM for Class Imbalance and Concept Drift	140
Author's Publications	141
Bibliography.....	142

SUMMARY

In this thesis, we present research work on sequential extreme learning machines for class imbalance and concept drift.

In Chapter 1, we give the background and literature review of class imbalance and concept drift problems in sequential learning and list the objectives and major contributions of the thesis.

In Chapter 2, we propose a weighted online sequential extreme learning machine (WOS-ELM) algorithm for binary class imbalance learning (CIL) without concept drift. WOS-ELM is a general online learning method that alleviates the class imbalance problem in both chunk-by-chunk and one-by-one learning modes. One of the new features of WOS-ELM is that an appropriate weight setting for CIL is selected in a computationally efficient manner. In one-by-one learning of WOS-ELM, a new sample can update the classification model without waiting for a chunk to be completed. Extensive empirical evaluations on 15 imbalanced datasets show that WOS-ELM obtains comparable or better classification performance than competing methods.

In Chapter 3, a computationally efficient framework, referred to as ensemble of subset online sequential extreme learning machine (ESOS-ELM), is proposed for binary class imbalance learning from a concept-drifting data stream. The proposed framework comprises a main ensemble representing short-term memory, an information storage module representing long-term memory and a change detection mechanism to promptly detect concept drifts. In the main ensemble of ESOS-ELM, each OS-ELM network is trained with a balanced subset of the data stream. Using ELM theory, a computationally efficient storage scheme is proposed to leverage the prior knowledge of recurring concepts. A distinctive feature of ESOS-ELM is that it can learn from new samples sequentially in both the chunk-by-chunk and one-by-one

modes. ESOS-ELM can also be effectively applied to imbalanced data without concept drift. On most of the datasets used in our experiments, ESOS-ELM performs better than the state-of-the-art methods for both stationary and non-stationary environments.

In Chapter 4, meta-cognitive online sequential extreme learning machine (MOS-ELM) is proposed for binary class imbalance and concept drift learning. In MOS-ELM, meta-cognition is used to self-regulate the learning, i.e., it selects suitable learning strategies for handling class imbalance and concept drift problems. A single OS-ELM based output update equation is also proposed which can be used under balanced, imbalanced, stationary and non-stationary conditions. The performance of MOS-ELM is evaluated under different conditions and compared with methods specific to those conditions. On most of the datasets, MOS-ELM outperforms the competing methods.

In Chapter 5, MOS-ELM is extended to multiclass data stream learning. In the context of sequential learning, it is the first method to alleviate the imbalance problem for both binary class and multiclass data streams.

Finally, Chapter 6 concludes the thesis with recommendations for future research.

LIST OF TABLES

Table 2-1 Confusion matrix	31
Table 2-2 Weight setting for TER	36
Table 2-3 Details of the datasets	43
Table 2-4 Number of hidden neurons and size of initialization set	45
Table 2-5 Classification results comparison for chunk-by-chunk learning	50
Table 2-6 Classification results comparison for chunk-by-chunk learning	51
Table 2-7 Classification results comparison for one-by-one learning	52
Table 3-1 Classification results comparison for non-stationary environments (one-by-one)	77
Table 3-2 Classification results comparison for non-stationary environments (chunk-by-chunk)	78
Table 3-3 Classification results comparison for balanced datasets	81
Table 3-4 Classification results comparison for stationary environments (chunk-by-chunk)	84
Table 3-5 Classification results comparison for stationary environments (one-by-one)	85
Table 4-1 <i>Gmean</i> obtained on imbalanced datasets	113
Table 4-2: Rank comparison among OS-ELM-SMOTE, WOS-ELM and MOS-ELM	115
Table 4-3 Classification results comparison for concept drifting datasets (balanced datasets)	117
Table 4-4 Classification results comparison for concept drifting datasets (imbalanced datasets)	119
Table 4-5 Classification results in terms of Mean, Dev and Min values using <i>Gmean</i>	122
Table 5-1 Details of multiclass imbalanced datasets	130
Table 5-2 <i>Gmean</i> values obtained by MOS-ELM and McELM	131
Table 5-3 Number of hidden neurons in MOS-ELM and McELM	131
Table 5-4 Drifting Gaussians dataset parametric equations	132
Table 5-5 <i>Gmean</i> values obtained on multiclass drifting datasets	133

LIST OF FIGURES

Figure 2-1: <i>Gmean</i> comparison between WOS-ELM and OS-ELM-SMOTE	53
Figure 2-2: <i>Gmean</i> comparison between WOS-ELM and OTER.....	53
Figure 2-3: Visualization of Nemenyi post-hoc test. The mean rank for each method over all the datasets is plotted on x -aix. Connected method are not significantly different at $p= 0.05$	54
Figure 3-1: ESOS-ELM for non-stationary recurring environments	57
Figure 3-2: One-by-one sequential learning in ESOS-ELM for imbalanced data streams	60
Figure 3-3: ESOS-ELM and WOS-ELM comparison on STAGGER Concepts	71
Figure 3-4: ESOS-ELM and DWM-CIL comparison on STAGGER Concepts.....	72
Figure 3-5: ESOS-ELM and WOS-ELM comparison on SEA Concepts.....	73
Figure 3-6: ESOS-ELM and DWM-CIL comparison on SEA Concepts	73
Figure 3-7: ESOS-ELM and WOS-ELM comparison on Rotating Spiral	75
Figure 3-8: ESOS-ELM and DWM-CIL comparison on Rotating Spiral.....	75
Figure 3-9: ESOS-ELM and WOS-ELM comparison on elect2 dataset.....	76
Figure 3-10: ESOS-ELM and Learn ⁺⁺ .NIE(<i>gm</i>) comparison on STAGGER concepts.....	78
Figure 3-11: ESOS-ELM and Learn ⁺⁺ .NIE(<i>gm</i>) comparison on SEA concepts.....	79
Figure 3-12: ESOS-ELM and Learn ⁺⁺ .NIE(<i>gm</i>) comparison on Rotating Spiral.....	79
Figure 3-13: ESOS-ELM and Learn ⁺⁺ .NIE(<i>gm</i>) comparison on elect2	79
Figure 3-14: Parameter tuning in ESOS-ELM	80
Figure 3-15: <i>Gmean</i> comparison between ESOS-ELM and WOS-ELM.....	83
Figure 3-16: Visualization of Nemenyi post-hoc test for chunk-by-chunk learning. The mean rank for each method over all the datasets is plotted on x -aix. Connected method are not significantly different at $p= 0.05$	86
Figure 3-17: Visualization of Nemenyi post-hoc test for one-by-one learning. The mean rank for each method over all the datasets is plotted on x -aix. Connected method are not significantly different at $p= 0.05$	86
Figure 4-1: Nelson and Narens model of meta-cognition.....	91
Figure 4-2: Meta-cognitive learning for class imbalance and concept drift	94
Figure 4-3: SMOTE oversampling	99

Figure 4-4: Adaptive window (a) window at time step ' k ' (b) window at time step ' $k+1$ '	104
Figure 4-5: Single OS-ELM update equation for class imbalance and concept drift.....	110
Figure 4-6: Visualization of Nemenyi post-hoc test for chunk-by-chunk learning. The mean rank for each method over all the datasets is plotted on x -axis. Connected methods are not significantly different at $p=0.05$	115
Figure 4-7: Classification results comparison for concept drifting datasets (balanced datasets) (a) STAGGER (b) SEA (c) Spiral (d) Elect2	118
Figure 4-8: Classification results comparison for concept drifting datasets (Imbalanced datasets) (a) STAGGER (b) SEA (c) Spiral (d) Elect2	120
Figure 5-1: MOS-ELM and WOS-ELM comparison on Gaussian1 dataset.....	134
Figure 5-2: MOS-ELM and McELM comparison on Gaussian1 dataset	134
Figure 5-3: MOS-ELM and WOS-ELM comparison on Gaussian2 dataset.....	134
Figure 6-1: Proposed methods and target applications.....	138

Chapter 1

INTRODUCTION

1.1 Background and Motivation

This thesis presents research work on two problem domains in sequential learning which is an important branch of machine learning: class imbalance and concept drift [1]-[4]. Sequential (or incremental) learners store the previously learnt information and update themselves only with the newly arrived samples from a data stream. They are computationally efficient as compared to batch learners [5]-[10] since batch learners require retraining with the complete dataset whenever new samples arrive. Recently, the class imbalance problem and the concept drift problem in sequential learning have attracted much attention of researchers from various application domains [1]. Class imbalance and concept drift are two problems that can exist either separately or concurrently in a data stream. Although a lot of work focus on

concept drift problem alone [11]-[17], the class imbalance problem and in particular, the combination of two problems in a data stream are largely unexplored.

Machine learning with imbalance in class distribution or class imbalance learning (CIL) has attracted much attention in the past decade or so for batch learning due to its wide applications in medical diagnoses, fraud detection, bioinformatics, oil spill detection, text classification etc [18]-[30]. In the binary classification problem, the class imbalance problem arises when one class has many more samples than the other [31]-[41]. A traditional classifier such as support vector machine (SVM) [42] [43] or extreme learning machine (ELM) [44] [45] trained with an imbalance dataset can be biased towards the majority class, i.e., it may achieve a very high majority class accuracy by compromising minority class accuracy [32] [46]. Hence, CIL methods are proposed to achieve high minority class accuracy without compromising much on the majority class accuracy. Generally speaking, two main approaches exist on tackling the class imbalance problem, the data sampling approach and the algorithm modification approach. In the data sampling approach, some majority class samples are removed (undersampling) [47]-[49] or some artificial minority class samples are added (oversampling) [50] [51] in order to balance the size of the two classes. There is usually some extra learning cost of data analysis and processing associated with the data sampling methods. In the algorithm modification approach, cost-sensitive learning methods are frequently used to tackle the CIL problem. Cost-sensitive learning methods shift the undesired bias of the majority class towards the more important minority class e.g. by assigning higher misclassification cost or higher weight to the minority class samples [52]-[59]. Most of these methods for the CIL problem were developed for batch learning.

It is not until recently that some sequential learning algorithms have been developed to address the class imbalance problem [60]-[67]. Most of these methods integrate data sampling into the sequential learning algorithms. In the context of sequential learning, oversampling is usually preferred over undersampling because of

the information loss associated with undersampling [3]. In general, data sampling methods learn data in chunks, or the so-called chunk-by-chuck learning. Hence, they treat the CIL problem in a way similar to the batch learning methods. A limitation of existing data sampling methods is that they do not consider one-by-one learning, i.e., these methods cannot be directly applied to one-by-one learning easily without accessing the old data. Very recently, an online total error rate (OTER) algorithm is proposed in [68] for learning one sample at a time, or the so-called one-by-one learning. The OTER algorithm does not consider the chunk-by-chunk learning and it uses fixed weights which may not be optimal for class imbalance learning.

In sequential learning, the statistical characteristics of training data streams may change over time, i.e., class concepts tend to drift in non-stationary environments. Concept drift occurs when the underlying distribution changes over time [69]. The function generating instances at the current time step needs not be the same as the one which generates instances at the next time step. Concept drift is said to have occurred in a data stream when a set of examples has legitimate class labels at one time step and has different labels at another time step. For example, in professor's email classification system, the concept of high priority email will change with semester and conference deadlines. Concept drift is a problem inherent in data streams which continues to limit incremental learner's performance. Concept drift learning raises the so-called stability-plasticity dilemma. Sequential learning framework should be able to achieve a meaningful balance between previously acquired information (stability) and learning new information (plasticity) [69]. In general, there are two types of concept drifts to be detected, sudden drift and gradual drift. Sudden drift is usually easier to detect compared to gradual drift since in the sudden drift the future samples no longer resemble the past data. Gradual drift is generally not very apparent as the performance degrades slowly if the drift is not taken care of. It can also be mistaken as noise. In concept drift learning, there are two challenges: first the drift needs to be detected and then appropriate actions need to be taken to predict the new data accurately. In

recurring environments [3] [69] [70], most of the existing methods may not be able to leverage the prior knowledge since old concepts are usually forgotten. Various methods have been proposed in the past for learning from drifting concepts, without taking into account the imbalance in class distributions. The class imbalance problem can further complicate sequential learning from drifting data streams.

Recently, class imbalance problem in drifting data streams has received some attention for chunk-by-chunk learning [1]. In an imbalanced environment, it becomes very difficult to detect the source of error in the minority class. Gao et al [61] proposed an ensemble method which collects the minority class samples from all the previous chunks and if required, remove some of the majority class samples from the current chunk, i.e., randomly select a subset of majority class samples in order to balance the two classes. This implicitly assumes that minority class is not drifting. The selectively recursive approach (SERA) [62] method also uses a subset of majority class samples to reduce the imbalance in class sizes. However, it is different from Gao's method in that it chooses the best minority class samples as determined by Mahalanobis distance, instead of selecting all the previous minority class samples. Thus, it would better detect the drifts in minority class [62] [63]. Similarly, Lichtenwalter and Chawla [66] also include the misclassified majority class samples along with all the minority class samples. The propagation of misclassified majority class samples defines a better decision boundary compared to the one defined by Gao's method. Learn⁺⁺.SMOTE [2] uses synthetic minority oversampling technique (SMOTE) [51] in the Learn⁺⁺.NSE [69] incremental learning algorithm. While Learn⁺⁺.UDNC [60] focuses on the CIL problem and has the ability to learn new-classes as well. Learn⁺⁺ family of methods is based on ensemble learning framework in which a new classifier is trained with each arriving chunk of data and added to the ensemble. Learn⁺⁺.NIE [3] is considered as the state-of-the-art method in such applications. It extends the Learn⁺⁺.NSE algorithm, originally proposed for balanced data streams, to imbalanced

applications. It uses bagging ensemble instead of a single classifier and undersamples the majority class when creating each bag. Learn⁺⁺.NIE works better than most of the competing methods in recurring environments when old concepts reappear since it does not remove old classifiers from the ensemble. For imbalance datasets, the prior knowledge of recurring concepts can be particularly useful since the minority class samples are usually rare. Learn⁺⁺.NIE stores all the previous hypotheses to better cope with recurring environments which results in greater computational complexity. Moreover, most of the existing methods assume that a full chunk of data is always available for training [1]. If the samples are arriving continuously or one-by-one, updating of the classification model is delayed until a full chunk is completed. Hence, the need for a CIL method for non-stationary environments, which can learn in both chunk-by-chunk and one-by-one modes, is timely.

Motivated by several problems in sequential learning, we propose to exploit extreme learning machine (ELM) [44] [45] in this thesis. ELM is becoming popular in large dataset and sequential learning applications due to its fast learning speed. ELM provides a single step least square estimation (LSE) method for training single hidden layer feed forward network (SLFN) instead of using iterative gradient descent methods such as back propagation algorithms. Online sequential extreme learning machine (OS-ELM) [8] has become one of the standard algorithms for sequential learning, i.e., when the complete dataset is not available at once and samples are received over time. An important feature of the OS-ELM method is that it can learn from data streams in both one-by-one and chunk-by-chunk modes. However, OS-ELM is originally proposed for balanced datasets and with the assumption that class concepts do not change over time [8]. As discussed earlier, many real word data streams are imbalanced and may have non-stationary distributions.

Several application-specific variants of OS-ELM exist [68] [71] [72] which can be chosen by users when considering the relative class sizes and changes in statistical properties of dataset. For example, an OS-ELM based method, referred to as timeliness

online sequential extreme learning machine (TOSELM) [72], adapts to the changing data distribution. For applications where data has a period of validity, i.e., after certain time period the old data becomes irrelevant, an online sequential extreme learning machine with forgetting mechanism (FOS-ELM) [71] is proposed to remove the outdated data. In non-stationary environments, FOS-ELM removes old (outdated) data after a fixed period of time while TOSELM uses an adjustable weighting mechanism to emphasize more on the new incremental data as compared to the historical data. Both FOS-ELM and TOSELM methods cannot be applied to the class imbalance problem. OTER [68] is another application specific OS-ELM method proposed for biometric scores fusion. Its output update equation is dependent on the relative class sizes. Application specific OS-ELM methods can be used interchangeably while learning from data streams for better generalization performance. It is difficult to predict nature of the data stream *a priori*, e.g., an imbalanced data stream may become balanced over time or vice versa. Thus, an automated or self-regulatory method should be implemented to adapt the learning accordingly.

The existing sequential learning methods proposed for imbalance data streams discuss only binary class case. Recently, some multiclass imbalance learning methods have been proposed for batch mode [73]-[77]. To the best of our knowledge, there is no multiclass sequential learner which can tackle the class imbalance problem in either stationary or non-stationary environments.

1.2 Objectives

With the motivation and background presented in the previous section, the objectives of this thesis are outlined as follows

- To develop a general online sequential learning method that alleviates the binary class imbalance problem without concept drift in both chunk-by-chunk

and one-by-one learning such that the update of the classification model is not delayed.

- To devise a framework for binary class imbalance learning from a concept-drifting data stream such that the drift is detected promptly and the prior knowledge of recurring concepts is leveraged in a computationally efficient way.
- To devise a self-regulatory mechanism which automatically selects the appropriate learning strategy for alleviating the class imbalance, the concept drift and the combined problem, and uses single OS-ELM output update equation for binary class imbalance and concept drift learning.
- To go beyond the binary class case and devise a multiclass sequential learning method that can tackle the class imbalance problem in data streams.

1.3 Contributions

Firstly, we propose a general online sequential learning method for the binary CIL problem without concept drift in Chapter 2. This method is referred to as weighted online sequential extreme learning machine (WOS-ELM). WOS-ELM is the first online sequential learning algorithm for the CIL problem with the capability of both chunk-by-chunk and one-by-one learning. Since the proposed method works for any chunk size, a chunk may or may not contain minority class samples. A computationally efficient weight selection procedure for alleviating the class imbalance problem is proposed. The weight tuning is performed to optimize a more appropriate measure for the CIL problem called G_{mean} . Extensive empirical comparison is performed between

WOS-ELM and data sampling methods for chunk-by-chunk learning, and also between WOS-ELM and the online total error rate method for one-by-one learning.

Secondly, a computationally efficient framework, referred to as ensemble of subset online sequential extreme learning machine (ESOS-ELM), is proposed in Chapter 3 for binary class imbalance learning from a concept-drifting data stream. In ESOS-ELM, a minority class sample is processed by I classifiers (I is the imbalance ratio) while a majority class sample is processed by a single classifier. The majority class samples are processed in a round robin fashion, i.e., the first majority class sample is processed by the first classifier, the second sample by the second classifier and so on. In this way, classifiers in the ensemble are trained with balanced subsets from the original imbalanced dataset. ESOS-ELM uses dynamic weighted majority voting for concept drift learning where voting weights are updated proportional to an appropriate performance measure for CIL. Note that the proposed framework tackles class imbalance and concept drift problems in both the one-by-one and chunk-by-chunk modes. For recurring environments, we propose a novel information storage mechanism, using ELM theory, which is efficient in terms of both memory and computation. A change detection mechanism is also employed in the learning framework to promptly react to both the sudden and gradual drifts.

In Chapter 4, a meta-cognitive online sequential extreme learning machine (MOS-ELM) is proposed for binary class imbalance and concept drift. It uses meta-cognitive learning [78] [79] to automatically select the appropriate learning strategy for class imbalance, concept drift and the combined problem. A unifying equation is proposed as a single OS-ELM output update equation that can be applied to class imbalance learning as well as concept drift learning. It also reduces to standard OS-ELM equation for balanced and stationary data streams. MOS-ELM is compared with various application-specific OS-ELM methods under different conditions and found to be superior to these methods in most cases.

In Chapter 5, MOS-ELM is extended and proposed as the first method to tackle class imbalance in multiclass data streams. Its performance is evaluated on eight stationary and two non-stationary data streams with different number of classes and imbalance ratios.

1.4 Literature Review on Class Imbalance and Concept Drift Learning

Class imbalance and concept drift are two problems in sequential learning which can significantly hinder the performance of traditional classifiers. Most of the research in these two areas has been done separately. In this section, some of the prominent works in the fields of class imbalance learning and concept drift learning are reviewed.

1.4.1 Class Imbalance Learning

Learning methods dealing with the class imbalance problem can be divided into two types: cost-sensitive methods and data sampling methods. The former approach modifies the learning algorithms while the latter resamples the datasets.

It has been established that the cost-sensitive learning problem and the CIL problem are interrelated [52]. In cost-sensitive learning methods, distinct misclassification costs are assigned to different classes and the optimal decision boundary is obtained by minimizing the total misclassification cost. The same approach can be used to solve the CIL problem by assigning a higher misclassification cost to the minority class. Several variants of cost-sensitive learning methods have been proposed to address the class imbalance problem [18]. When dealing with the CIL problem, the cost-sensitive learning methods have been commonly used with boosting algorithms, such as AdaBoost [54]. These cost-sensitive boosting methods introduce

different misclassification costs into the weight updating procedure of the boosting algorithms. AdaCost [55], CSB [56] and AsymBoost [59] are some of the popular cost-sensitive boosting methods applied to the CIL problem. For example, AdaCost introduces cost sensitivity inside the exponent of the weight updating formula of AdaBoost. It uses a cost-adjustment function that aggressively increases the weights of costly misclassifications and conservatively decreases the weights of high-cost examples that are correctly classified. A drawback associated with cost-sensitive learning methods is that the knowledge of the optimal cost ratio is often unknown *a priori*. Moreover, different cost ratios may lead to different results.

Data sampling can be done in two ways: remove some samples from the majority class (undersampling) or add some artificial samples to the minority class (oversampling). Samples are removed or added from the dataset until the classes are balanced or a desired imbalance ratio is achieved. The basic form of undersampling is random undersampling in which some samples from the majority class are randomly removed until the classes are balanced. The drawback of random undersampling is that it results in information loss. A more intelligent form of undersampling is the one-sided selection [47]. In this method only noisy, borderline and redundant samples from the majority class are removed. However, for many datasets, these samples represent only a small portion of samples in the majority class. Hence, in balancing the dataset some informative samples may still be removed. Similar to random undersampling, random oversampling is the most basic oversampling technique which duplicates minority class samples until the classes are balanced. A more intelligent form of oversampling is synthetic minority oversampling technique (SMOTE) [51]. For some specified integer k , k -nearest neighbors of a minority class sample under consideration are selected. From this group, a neighbor is chosen randomly and a synthetic sample is created along the line joining this random neighbor and the given minority class sample. Instead of simply duplicating the minority class samples, SMOTE creates artificial minority class samples in the neighborhood of the original minority class

samples. However, the duplication or synthetic creation of the minority class samples often results in overfitting.

Several combinations of sampling and boosting/ensemble learning methods have also been proposed to overcome the drawbacks associated with data sampling. Some prominent methods in this category include SMOTEBoost [50], RUSBoost [48], DataBoostIM [58], EasyEnsemble [49], BalnacedCascade [49], Chan and Stolfo's method [57], etc. For example, SMOTEBoost integrates the concept of SMOTE oversampling method into AdaBoost boosting method. It introduces synthetic samples created with SMOTE technique into boosting iteration of AdaBoost. Similarly, RUSBoost integrates random undersampling into AdaBoost. While DataBoostIM modifies the data generation technique of AdaBoost by generating synthetic samples according to the ratio of difficult-to-learn samples between classes. Minority class samples are generally more difficult to learn as compared to majority class samples. Therefore, it is expected that more synthetic minority class samples are generated which results in the decrease of imbalance in datasets.

Motivated by the good generalization performance of SVMs, researchers have also proposed several SVM based CIL methods. These methods include different error costs (DEC) [21], fuzzy support vector machines for class imbalance learning (FSVM-CIL) [25], SMOTE with different costs (SDC) [26], boosting support vector machines (BSVM) [31], granular support vector machines with repititive undersampling (GSVM-RU) [32], kernal boundary allignment (KBA) [34], multiplicative weight 'z' support vector machines (zSVM) [33], etc. The SVM algorithm is made cost-sensitive in DEC by inserting distinct misclassification cost values in the SVM objective function. FSVM-CIL is similar to DEC except that it uses FSVMs instead of SVMs to suppress the effect of noise. SDC applies SMOTE to the cost-sensitive SVM classifier, while BSVM applies boosting to the cost-sensitive SVM classifier. GSVM-RU is a granular approach that tackles the SVM class imbalance problem by implementing a guided repetitive undersampling technique. KBA adjusts the boundary of an SVM

classifier by a kernel modification method, while zSVM decreases the bias toward the majority class by increasing the weight of the positive support vectors in the SVM decision function. In general, existing SVM based CIL methods either modify the SVM learning algorithms (e.g. incorporate distinct cost values) or resample the datasets.

For a detailed review of existing CIL methods, readers can refer to [18], [20] and [40]. Note that the existing CIL methods for sequential learning generally use some form of data sampling approach such as SMOTE oversampling to alleviate the class imbalance in data streams. Cost-sensitive approach has not been explored much for tackling class imbalance in sequential learning.

1.4.2 Concept Drift Learning

Concept drift learning is a type of online sequential learning where target concept changes or drifts over time. For example, the shopping behavior of customers changes seasonally. An online predictive model should be able to track such changes and adapt accordingly. Generally speaking, there are two main types of concept drift to be detected, gradual and sudden. An effective concept drift learner should be able to respond to both sudden and gradual drifts and also recognizes old recurring contexts for efficient learning.

One of the common approaches to learn from non-stationary distributions is to adapt the base learner. Various modifications of decision trees, C4.5 in particular, have been proposed to dynamically adapt to new training data which contradicts the old concept. The trees are grown by taking the change in concept into consideration. Concept adapting very fast decision tree (CVFDT) [80] is one of the earlier methods proposed for concept drift learning. It uses a fixed sliding window of instances which

is essentially a time-based window. Later some methods were proposed with concept-based window. In these methods, the window is updated with respect to concepts as opposed to time [1]. Another base classifier that has been modified to tackle concept drift is the k -nearest neighbors (k -NN) method. In [81], the update of k -NN method is dependent on whether concept drift is detected or not. If concept drift is not detected, new samples are added to the existing knowledge base. Otherwise, obsolete samples are removed from the knowledge base.

Another approach, independent of the base learner, is to select only some recent training samples, corresponding to the current concept, or weighting the new samples higher compared to old samples. A simplest form is to merely maintain a window with a fixed number of newest samples. An obvious drawback of this method is that the appropriate window size is not known in most of the cases. A larger window may not detect the concept drift promptly while a smaller window may not have sufficient information. Various methods have been proposed to adapt the window size in order to fit the current concept. FLORA3 [82] proposed by Kubat and Widmer, uses a variable size window. The window is shrunk or grown based on how the algorithm is responding to the current concept. For poor predictive performance, the window is shrunk aggressively. If the coverage is high and performance is good, the size stays the same. Otherwise the size is increased by 1. Bifet and Gevalda [14] divide the window into sub-windows representing the same distribution. The means of two sub-windows are compared and the concept drift is assumed to have been detected if they are distinct enough. In order to reduce the computational complexity of the method, a logarithmic memory is used. In [83] two windows, a small window and a medium window, are used. The new samples are stored in small window while the medium window is updated depending on the detection of concept drift. Once the drift is detected, the medium window captures the new concept.

A similar approach for concept drift learning is the weighting technique. The weighting assigns relative importance to the samples. In windowing technique,

samples are either in the window or not while weighting provide more price control over how samples are used to update the classification model. Adaptive weighted k -NN is one such method [84]. The weighting technique is particularly useful in gradual concept drift learning since the old samples are not completely removed rather they are assigned lower weight compared to the new samples. In this way, the new samples contribute more to the classification model.

Several systems have been proposed that use ensembles of classifiers on sequential chunks of training samples for large-scale data streams. Ensemble methods are also used for concept drift learning due to their robust generalization performance. Streaming ensemble algorithm (SEA) [85], one of the earliest methods, trains multiple classifiers with non-overlapping windows. If the number of classifiers is less than the maximum ensemble size, a classifier is added with the new instance. Otherwise the worst performing classifier is removed from the classifier. Wang [86] weight each classifier during the voting process. A classifier is weighted inversely to the expected error. Dynamic weighted majority (DWM) [87] is a state-of-the-art ensemble method for concept drift learning with balanced datasets. In DWM, voting weights are decreased proportional to the error rate of the classifier. If the ensemble misclassifies the new sample, a new classifier is added in the ensemble with the weight of 1. The classifiers which misclassify this instance have their weights reduced. If the weight of any classifier falls below a predefined threshold, it is removed from the ensemble. After each time step, the classifier weights are normalized so that the maximum weight is always 1.

Learn⁺⁺[5] and its variants [69] [17], proposed by Polikar, are also becoming popular for concept drift learning. Learn⁺⁺ family of methods is an ensemble learning framework which is loosely based on AdaBoost. A new classifier (weak) is trained with each arriving chunk of data and added to the ensemble. The weak learners are then voted together using a weighted average according to the current normalized error of classifier. Learn⁺⁺.NSE [69] uses weighted voting which is based on time-adjusted

accuracy on current and past environments so that it gives more weight to classifiers performing better on the current environment, irrespective of the age of the classifier. This type of weighting is particularly useful in recurring environments when the old concepts become relevant again. Since Learn⁺⁺.NSE does not discard any classifier so the computational complexity can be an issue. Several attempts were made at pruning without significantly compromising the ability to learn the recurring concepts effectively. In order to leverage the prior knowledge of recurring concepts, previously learnt information should be retained. Adaptive Classifiers-Ensemble systems (ACE) [70] uses an online classifier trained on the current concept (short-term memory) and many batch classifiers for retaining old concepts (long-term memory). Classifiers outputs are combined using weighted majority voting. It uses a long-term buffer to store the most recent samples. A new batch classifier is trained whenever concept drift is detected or the buffer is full. A drawback associated with most of the existing ensemble methods is that classifiers are trained with fixed window size which may not match well with the drifts in concepts. Concept drift committee (CDC) uses a series of incremental classifiers. If a classifier has seen enough samples, it is considered ‘mature’ and is used for classification. If its performance falls below a threshold, it is removed from the ensemble and a new classifier is trained with the current samples.

For a detailed review of existing concept drift learning methods, readers can refer to [1] and [15].

Chapter 2

WEIGHTED ONLINE SEQUENTIAL EXTREME LEARNING MACHINE FOR BINARY CLASS IMBALANCE LEARNING WITHOUT CONCEPT DRIFT

2.1 Introduction

Most of the existing sequential learning methods for class imbalance use data sampling approach. A limitation of data sampling approach is that they learn data in chunks and cannot be directly applied to one-by-one learning easily without accessing the old data. Thus, update of the classification model may be delayed until a complete data chunk is received. This may not be desired in some applications. For example, in classification of emails as relevant or irrelevant to a user, the classification model should be immediately updated as soon as the user identifies/tags an important email [65]. Another problem with data sampling methods is that learning cannot be carried out efficiently with highly imbalanced data streams when the chunk size is smaller than the imbalance ratio. This is because in such a case, a minority sample may arrive after

several chunks comprising of only majority class samples and for most data sampling methods to work, each chunk has to contain some minority samples. Some of the data sampling methods [61]-[63] can overcome the above problem by relying on previously learnt data which however, may not be always accessible, particularly when the data size is huge [2]. Moreover, the training time for data sampling methods may be long as some extra learning cost is required for data analysis and processing. This may be another limitation as computational efficiency is crucial for online learning.

In this chapter, we propose a weighted online sequential extreme learning machine (WOS-ELM) algorithm for binary class imbalance learning (CIL) without concept drift. The main contributions of this chapter are enumerated as follows:

1. The proposed WOS-ELM algorithm can tackle the CIL problem in both chunk-by-chunk and one-by-one learning modes. To the best of our knowledge, this is the first online sequential learning algorithm for the CIL problem with the capability of both chunk-by-chunk and one-by-one learning. Since the proposed method works for any chunk size, a chunk may or may not contain minority class samples.
2. A computationally efficient weight selection procedure for alleviating the class imbalance problem is proposed. The weight tuning is performed to optimize a more appropriate measure for the CIL problem called G_{mean} .
3. Extensive empirical comparison is performed between WOS-ELM and data sampling methods for chunk-by-chunk learning, and also between WOS-ELM and the online total error rate method (OTER) for one-by-one learning. Statistical significance test is also carried out to check whether the new method is statistically better than the methods in comparison in terms of G_{mean} .

This chapter is organized as follows. Section 2.2 discusses the preliminaries. Section 2.3 presents the details of the new online sequential learning method for class imbalance. This is followed by experiments for validating the performance of the proposed framework in Section 2.4. Finally, Section 2.5 concludes the chapter.

2.2 Preliminaries

In this section, we first briefly review the theory of batch ELM [44] and online sequential ELM [8]. Then we discuss the total error rate (*TER*) method, a classification error based objective function [88] [89], and its online version referred to as the online total error rate (*OTER*) method [68].

2.2.1 ELM and Online Sequential ELM Methods

Extreme learning machine was proposed in [44] which essentially provides a single step least squares error (LSE) estimate solution for the single-hidden-layer feedforward network (SLFN). With ELM there is no need to tune the hidden layer of the SLFN as in traditional gradient based algorithms. Independent of the training data, ELM randomly assigns computational nodes in the hidden layer while the output weights connecting the hidden layer and the output layer are determined analytically.

Consider a q class training dataset $\{x_i, y_i\}$, $i = 1, \dots, N$ and $y_i \in R^q$. $x_i \in R^d$ is a d -dimensional data point. The SLFN output with L hidden nodes is given by

$$o_i = \sum_{j=1}^L \beta_j G(a_j, b_j, x_i) , \quad i = 1, \dots, N \quad (2.1)$$

where a_j and b_j , $j = 1, \dots, L$ are the j th hidden node's learning parameters assigned randomly without considering the input data. $\beta_j \in R^q$ is the output weight vector connecting the j th hidden node to the output nodes and $G(x)$ can be any infinitely

differentiable activation function such as sigmoidal function or radial basis function in the hidden layer.

The N equations in (2.1) can be written in a compact form as below

$$O = H\beta \quad (2.2)$$

$$\text{where } H = \begin{bmatrix} G(a_1, b_1, x_1) & \dots & G(a_L, b_L, x_1) \\ \vdots & \dots & \vdots \\ G(a_1, b_1, x_N) & \dots & G(a_L, b_L, x_N) \end{bmatrix}_{N \times L}$$

which is called the hidden layer output matrix, H_{ij} represents the j th hidden node output corresponding to the input x_i . $\beta = [\beta_1, \beta_2, \dots, \beta_L]^T$ and $O = [o_1, o_2, \dots, o_N]^T$.

In order to find the output weight matrix β which minimizes the cost function $\|O - Y\|$, a LSE solution of (2.2) is obtained as below

$$\beta = H^\dagger Y = (H^T H)^{-1} H^T Y \quad (2.3)$$

where $Y = [y_1, y_2, \dots, y_N]^T$ and $H^\dagger$ is the Moore-Penrose generalized inverse of matrix H . This closed form single step LSE solution is referred to as basic extreme learning machine [44].

An online sequential version of batch ELM referred to as online sequential extreme learning machine (OS-ELM) [8] has been proposed for the SLFN. OS-ELM achieves better generalization performance than the previous algorithms proposed for SLFN [9] [10] at a much faster learning speed. It can learn data one-by-one as well as chunk-by-chunk (with a fixed or varying chunk length). Only the newly arrived samples are learnt at any given time so the samples that have already been used in the

learning procedure can be discarded. The online sequential learning in OS-ELM consists of two steps.

Step1: Initialization

In this step, a small portion of training data $D_0 = \{x_i, y_i\}$, $i = 1, \dots, N_0$ with $N_0 \in N$ is considered for initializing the learning. The initial output weight matrix is calculated according to the ELM algorithm by randomly assigning the input weights a_j and bias b_j , $j = 1, \dots, L$ as follows

$$\beta^{(0)} = P_0 H_0^T Y_0 \quad (2.4)$$

where $P_0 = (H_0^T H_0)^{-1}$ and H_0 is the initial hidden layer output matrix.

It is recommended that the number of initial training samples should be greater than or equal to the number of hidden neurons. With this setting the generalization performance of online sequential ELM reaches that of the batch ELM.

Step 2: Sequential Learning

Upon the arrival of new set of observations $D_{k+1} = \{x_i, y_i\}$, $i = (\sum_{l=0}^k N_l) + 1, \dots, \sum_{l=0}^{k+1} N_l$, i.e., the $(k+1)th$ chunk of data, first compute the partial hidden layer output matrix H_{k+1} . N_{k+1} is the number of samples in the $(k+1)th$ chunk. Then by using the output weight update equation shown below, calculate the output weight matrix $\beta^{(k+1)}$ with $Y_{k+1} = [y_{(\sum_{l=0}^k N_l)+1}, \dots, y_{\sum_{l=0}^{k+1} N_l}]^T$.

$$\begin{aligned} P_{k+1} &= P_k - P_k H_{k+1}^T (I + H_{k+1} P_k H_{k+1}^T)^{-1} H_{k+1} P_k \\ \beta^{(k+1)} &= \beta^{(k)} + P_{k+1} H_{k+1}^T (Y_{k+1} - H_{k+1} \beta^{(k)}) \end{aligned} \quad (2.5)$$

The above equation is analogous to the recursive least squares algorithm. Each time a new chunk of data arrives, the output weight matrix is updated according to (2.5). Note that the general update equation (2.5) is for both chunk-by-chunk and one-by-one learning since the one-by-one learning can be considered as a special case of chunk-by-chunk learning with chunk size equal to 1.

2.2.2 Total Error Rate Methods

In machine learning applications, least square error has been used as an objective function due to its simplicity and tractability in optimization. LSE has shown good generalization performance, particularly for regression applications [44]. For classification applications, there exists a mismatch between the minimization of cost function in LSE and the actual classification objective. Recently, a classification error based objective function is proposed [88] [89] which, under quadratic approximation, is deterministically solvable. The proposed objective function essentially minimizes the total error rate (TER) or maximizes the overall accuracy. Accuracy (Acc), a commonly used evaluation criterion of a standard classification system, is defined as

$$Acc = \frac{tp + tn}{tp + fn + tn + fp} \quad (2.6)$$

where

tp = correctly classified positive samples

tn = correctly classified negative samples

fn = misclassified positive samples

fp = misclassified negative samples

Also refer to the confusion matrix defined in Table 2-1 for the above quantities

Table 2-1 Confusion matrix

	predicted positive	predicted negative
actual positive	tp	fn
actual negative	fp	tn

Let

$$tp + fn = s^+ = \text{number of positive class samples}$$

$$tn + fp = s^- = \text{number of negative class samples}$$

$$TPR = tp / s^+ = \text{true positive rate}$$

$$TNR = tn / s^- = \text{true negative rate}$$

$$FPR = fp / s^- = \text{false positive rate}$$

$$FNR = fn / s^+ = \text{false negative rate}$$

$$s = s^- / s^+$$

Now (2.6) becomes

$$Acc = \frac{TPR + sTNR}{1 + s} \quad (2.7)$$

In order to relate Acc with TER , s is set to 1 (the relative importance of TPR and TNR is the same) and also the normalization factor is ignored [81]. Thus, (2.7) becomes

$$\begin{aligned}
 Acc &= TPR + TNR \\
 &= \frac{tp}{s^+} + \frac{tn}{s^-} \\
 &= 1 - TER
 \end{aligned} \quad (2.8)$$

where

$$TER = FPR + FNR \quad (2.9)$$

Under quadratic approximation, solution for direct minimization of TER was found to be related to that of weighted least squares method with class-specific weight setting corresponding to the sizes of the two classes (s^- , s^+). The output weight matrix β that minimizes the TER is found to be [81]

$$\beta = (H^T W H)^{-1} H^T W Y \quad (2.10)$$

where

$$W = W_- + W_+ \text{ and } W_- = \text{diag}(\frac{1}{s^-}, \dots, \frac{1}{s^-}, 0, \dots, 0) \quad W_+ = \text{diag}(0, \dots, 0, \frac{1}{s^+}, \dots, \frac{1}{s^+})$$

Note that the elements of H and Y are ordered according to the samples in the two classes, i.e., all the negative class samples come first, followed by the positive class samples.

It can be observed that (2.10) has a similar structure as (2.3), except for the existence of a weighting matrix W . In this chapter, we call this weighted least squares solution as weighted extreme learning machine (WELM). By setting $s^+ = s^- = 1$, W becomes an identity matrix and equation (2.10) reduces to (2.3).

The main reason for the superior performance of the total error rate method over least square error classifier is its ability to cater for difference in the positive and negative class sizes through the weighting matrix.

Recently, an approximated online version of TER, i.e., online total error rate (OTER) is proposed for one-by-one learning. Similar to online sequential ELM, learning in OTER consists of two steps; initialization and sequential learning. The output weight update equation is given as

$$P_{k+1} = (\mu_{k+1})^{-1}P_k - \frac{(\mu_{k+1})^{-1}P_k h_{k+1} h_{k+1}^T (\mu_{k+1})^{-1}P_k}{w_{k+1}^{-1} + h_{k+1}^T (\mu_{k+1})^{-1}P_k h_{k+1}}$$

$$\beta^{(k+1)} = \beta^{(k)} + w_{k+1}P_{k+1}h_{k+1}(y_{k+1} - h_{k+1}^T \beta^{(k)}) \quad (2.11)$$

where

$$w_{k+1} = \begin{cases} w_{k+1}^- = \frac{1}{s_{k+1}^-} & \text{if } y_{k+1} = -1 \\ w_{k+1}^+ = \frac{1}{s_{k+1}^+} & \text{if } y_{k+1} = +1 \end{cases}$$

with

s_{k+1}^- = no. of negative class samples received up till $(k + 1)$ th iteration

s_{k+1}^+ = no. of positive class samples received up till $(k + 1)$ th iteration

and stacking weight update

$$\mu_{k+1} = \begin{cases} \frac{1}{2}(1 + \frac{w_{k+1}^-}{w_k^-}) & \text{if } y_{k+1} = -1 \\ \frac{1}{2}(1 + \frac{w_{k+1}^+}{w_k^+}) & \text{if } y_{k+1} = +1 \end{cases}$$

In order to minimize total error rate, the weights associated with both the old and the new samples should correspond to the current class sizes. μ_{k+1} provides an approximate rescaling of the weights associated with the old samples.

2.3 Proposed Method

In this section, we propose a weighted online sequential extreme learning machine (WOS-ELM) algorithm for class imbalance learning. WOS-ELM can learn data in both chunk-by-chunk and one-by-one mode. In WOS-ELM, the weights for

alleviating the class imbalance problem are tuned in a computationally efficient manner. Before formally proposing WOS-ELM, we briefly discuss the effect of weights tuning in weighted extreme learning machine (WELM).

2.3.1 Weight Selection in Weighted ELM for Class Imbalance Learning

Weighted ELM can be considered as a cost-sensitive algorithm for CIL since it assigns higher weight to the minority class samples, i.e., $\frac{1}{s^+} > \frac{1}{s^-}$. This particular setting ($w^- = \frac{1}{s^-}$ and $w^+ = \frac{1}{s^+}$) in (2.8) assigns equal weight to the positive class and the negative class accuracies (TPR and TNR). One can tune the weights of WELM to optimize a suitable measure for CIL.

It should be first pointed out that the overall accuracy or total error rate may not be an appropriate evaluation criterion for highly imbalanced classes. Traditional classifiers like ELM can achieve very high overall accuracy by obtaining very high TNR at the expense of very low TPR . This is not desirable in applications such as medical diagnosis of a rare disease. In this application, correct classification of the minority class samples is usually more important than that of the majority class samples. For example, a dataset may contain only 5 percent of minority class samples and 95 percent of majority class samples. If the classifier classifies all the samples as belonging to the majority class, the overall accuracy of 95 percent can be obtained but the accuracy for the minority class will be 0 percent. In such a case, although a very low total error rate (or very high overall accuracy) is obtained, the important minority class is ignored completely. To overcome this problem, G_{mean} is adopted as a measure for the classification performance in this chapter and it is defined below:

$$G_{mean} = \sqrt{TPR \times TNR}$$

In fact, G_{mean} is a popular measure for the CIL problem [18] [32] [26] [25] [90]. It is used when the combined performance of the two classes needs to be judged by a single metric. It measures both TPR and TNR in a balanced way. Unlike the overall accuracy, a high G_{mean} value requires both TPR and TNR to be high and the difference between TPR and TNR to be low. For highly imbalanced datasets, a high overall accuracy can always be achieved by classifying all the samples into the majority class and ignoring the minority class samples completely. On the other hand, G_{mean} cannot be high if the accuracy of one of the two classes is very low. Consider the above simple example again, G_{mean} will become zero when all the samples are classified as in the majority class.

Depending on the applications, other evaluation measures such as area under receiver operating characteristic curve (*auROC*), *F-measure*, and *Adjusted G_{mean}* (AG_{mean}) etc. can be optimized instead of G_{mean} e.g. in some bioinformatics applications where a high sensitivity of change detection in TNR as compared to that in TPR is required, the *Adjusted G_{mean}* or AG_{mean} can be more meaningful than G_{mean} [22]-[24]. However, our work is closely related to the total error rate method which may have the short-coming of ignoring TPR in order to maximize the overall accuracy. Thus, in this work we aim at increasing TPR and TNR in a balanced way, i.e., considering both being equally important. This can be achieved by G_{mean} optimization with efficient weight tuning which will be discussed shortly. In our future work, we shall consider formulating efficient weight tuning for the optimization of other CIL evaluation measures as well.

For the minimization of total error rate, weight tuning is proposed [89] as suggested in Table 2-2. In most of the two class datasets used in [88] [89], the ratio of the two class sizes (imbalance ratio) is between one and two, and the highest imbalance ratio is 3.2, i.e., the datasets are either balanced or slightly imbalanced.

Table 2-2 Weight setting for TER

z	$(\frac{1}{w^-}, \frac{1}{w^+})$	z	$(\frac{1}{w^-}, \frac{1}{w^+})$
0	(1,1)	5	$(0.5s^-, s^+)$
1	(s^-, s^+)	6	$(s^-, 2s^+)$
2	$(s^-, 1)$	7	$(2s^-, s^+)$
3	$(1, s^+)$	8	$(s^-, 0.2s^+)$
4	$(s^-, 0.5s^+)$	9	$(0.2s^-, s^+)$

We have applied the weight setting suggested in [89], also shown in Table 2-2. We found that the best weight setting from the table may not lead to the highest G_{mean} . The weight setting corresponding to $Z=1$ assigns equal weight to TPR and TNR , i.e., which maximizes both accuracies with equal importance. Thus, for obtaining a higher G_{mean} for a class imbalanced dataset, we may need to tune near this particular setting. From [88] [89], we selected the dataset with the highest imbalance ratio (3.2) and three datasets with imbalance ratio between 1 and 2. G_{mean} is maximized using the weight settings in Table 2-2. A finer tuning is then performed with an interval of 0.1 between the weight setting corresponding to $Z=1$ and its closest weight settings corresponding to $Z=6$ and $Z=7$. The weight settings used for fine tuning are (s^-, s^+) , $(1.1s^-, s^+)$, $(1.2s^-, s^+)$, ..., $(2s^-, s^+)$, $(s^-, 1.1s^+)$, $(s^-, 1.2s^+)$, ..., $(s^-, 2s^+)$. 50 runs of 10 fold cross-validation is performed with the average number of hidden neurons reported in [89]. On 3 out of 4 datasets, the highest G_{mean} is obtained using fine tuning.

We also evaluated the performance of weighted ELM on 15 real-world imbalanced datasets. These datasets will later be used for online sequential learning. On eight out of fifteen datasets, higher G_{mean} values than those obtained by $Z=1$ are observed using fine tuning. On four datasets, the improvement is higher than 1% while on the remaining four datasets the improvement is less than 1%. Though the improvement is small, it cannot be ignored. On these 15 imbalanced datasets, weight settings from $Z=6$ and $Z=7$ do not obtain higher G_{mean} than $Z=1$. Note that $Z=0$

represents original ELM which obtained lower G_{mean} than $Z=1$ for all these highly imbalanced datasets. From the results, we recognize the importance of tuning the weights of WELM in the vicinity of weight settings corresponding to $Z=1$.

Note that an exhaustive weight tuning is performed in [89] for minimization of total error rate, i.e., all the ten weights settings are always tested. In contrast, in our proposed method, when G_{mean} is maximized, the result of $Z=1$ are considered first in order to test the next appropriate setting. This efficient weight tuning is adopted for online sequential learning in the next sub-section.

2.3.2 Weighted Online Sequential Extreme Learning Machine

After recognizing the importance of tuning weights in Weighted ELM, we now discuss an efficient sub-optimal weight selection method for alleviating the class imbalance problem in online learning. The recent online total error rate method [68] uses fixed weights aiming at the minimization of total error rate. In the WOS-ELM to be described shortly, weight tuning is adopted to optimize the G_{mean} which is an appropriate CIL evaluation criterion as discussed previously. To compromise classification performance and computational efficiency, the fine-tuning suggested in the previous sub-section is further simplified.

We start with $Z=1$ (see Table 2-2) as it maximizes TPR and TNR with equal weight, i.e., the accuracies of both classes are maximized without a bias. By observing the TPR and TNR obtained, further weight setting is considered in an attempt to achieve a higher G_{mean} . In general, G_{mean} is higher when both accuracies are equal. Hence, if TPR and TNR are already close using $Z=1$, there is no need to tune the weights further. Now if TPR is greater than TNR , we will try to increase TNR by tuning the weights in the range between (s^-, s^+) and $(s^-, 2s^+)$. For sub-optimal tuning to keep the computational cost low, we only select $(s^-, 1.2s^+)$, $(s^-, 1.5s^+)$,

$(s^-, 1.8s^+)$, $(s^-, 2s^+)$ to increase TNR . Similarly, if TNR is greater than TPR , we only select $(1.2s^-, s^+)$, $(1.5s^-, s^+)$, $(1.8s^-, s^+)$, $(2s^-, s^+)$ to increase TPR . For most of the imbalanced datasets we have tested, the above suggested weight tuning is sufficient.

We now describe the WOS-ELM in details. Following (2.10), the weighted least square solution corresponding to the initialization chunk n_0 is given by

$$\beta^{(0)} = K_0^{-1} H_0^T W_0 Y_0 \quad (2.12)$$

$$K_0 = H_0^T W_0 H_0 \quad (2.13)$$

where W_0 is constructed according to the size of the two classes in D_0 . The starting weights are given as $w_0^- = \frac{1}{s_0^-}$ and $w_0^+ = \frac{1}{s_0^+}$, where s_0^- and s_0^+ represent number of negative class samples and number of positive class samples in the chunk respectively. w_0^- is the weight associated with all the majority class samples and w_0^+ is the weight associated with all the minority class samples in this chunk. Alternatively, to simplify the weight tuning, w_0^- can be set as 1 and w_0^+ can be set as the current imbalance ratio i.e. $\frac{s_0^-}{s_0^+}$. By observing TPR and TNR obtained using the validation set, w_0^+ is increased or decreased for optimization as discussed above, e.g. to increase TPR , $1.2w_0^+$, $1.5w_0^+$, $1.8w_0^+$, $2w_0^+$ can be tuned as an attempt to increase G_{mean} .

With the arrival of the next chunk of data, the weighted least square solution is given by

$$\beta^{(1)} = K_1^{-1} \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}^T \begin{bmatrix} W_0 & 0 \\ 0 & W_1 \end{bmatrix} \begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} \quad (2.14)$$

$$K_1 = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}^T \begin{bmatrix} W_0 & 0 \\ 0 & W_1 \end{bmatrix} \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}$$

where H_1, Y_1 correspond to the second chunk of data D_1 . The starting weights in W_1 are given as $w_1^- = 1$ and $w_1^+ = \frac{s_1^-}{s_1^+}$, where w_1^+ represents the overall imbalance ratio and s_1^- and s_1^+ represent the numbers of negative class samples and positive class samples up till time step ‘1’, respectively. As in the first chunk, w_1^+ is also tuned for a possible higher G_{mean} .

The purpose of W_0 and W_1 is to assign higher weight to the minority class samples. Let us assume that the imbalance ratio is the same in both chunks, i.e., $w_0^+ = w_1^+ = w^+$. It is possible that the weight associated with the minority class samples in the first chunk that maximizes G_{mean} is $1.2 w^+$ while in the second chunk it is $1.5 w^+$. For G_{mean} optimization, we can either allow different minority class weights in different chunks or use the same weight in all the chunks. For using the same weight, a rescaling of the old weights has to be performed. For an exact rescaling, an inversion of the summation of three matrices is required which is mathematically difficult, if not impossible to achieve (see [68]). In OTER, an approximate rescaling is proposed. It reduces three terms to two terms such that the matrix inversion lemma can be utilized. In OTER, rescaling is required since a common weight setting corresponding to the class sizes needs to be associated with all the samples. In WOS-ELM, we use different weights in different chunks to maximize G_{mean} . Thus, instead of performing approximate weights rescaling of the previously learnt samples, we allow different minority class weights in different chunks. With this in mind, we are ready to derive the online sequential algorithm in the following.

For sequential learning $\beta^{(1)}$ should be expressed as a function of K_1, H_1, Y_1 and not as a function of old dataset K_0, H_0, Y_0 .

$$\begin{aligned} K_1 &= \begin{bmatrix} H_0^T & H_1^T \end{bmatrix} \begin{bmatrix} W_0 & 0 \\ 0 & W_1 \end{bmatrix} \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} \\ &= H_0^T W_0 H_0 + H_1^T W_1 H_1 = K_0 + H_1^T W_1 H_1 \end{aligned}$$

Similarly,

$$\begin{aligned}
\begin{bmatrix} H_0 \\ H_1 \end{bmatrix}^T \begin{bmatrix} W_0 & 0 \\ 0 & W_1 \end{bmatrix} \begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} &= H_0^T W_0 Y_0 + H_1^T W_1 Y_1 \\
&= K_0 K_0^{-1} H_0^T W_0 Y_0 + H_1^T W_1 Y_1 \\
&= K_0 \beta^{(0)} + H_1^T W_1 Y_1 \\
&= (K_1 - H_1^T W_1 H_1) \beta^{(0)} + H_1^T W_1 Y_1 \\
&= K_1 \beta^{(0)} - H_1^T W_1 H_1 \beta^{(0)} + H_1^T W_1 Y_1
\end{aligned}$$

Now,

$$\begin{aligned}
\beta^{(1)} &= K_1^{-1} \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}^T \begin{bmatrix} W_0 & 0 \\ 0 & W_1 \end{bmatrix} \begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} \\
&= K_1^{-1} (K_1 \beta^{(0)} - H_1^T W_1 H_1 \beta^{(0)} + H_1^T W_1 Y_1) \\
&= \beta^{(0)} + K_1^{-1} H_1^T W_1 (Y_1 - H_1 \beta^{(0)})
\end{aligned}$$

The above arguments can be generalized as follows

$$K_{k+1} = K_k + H_{k+1}^T W_{k+1} H_{k+1}$$

$$\beta^{(k+1)} = \beta^{(k)} + K_{k+1}^{-1} H_{k+1}^T W_{k+1} (Y_{k+1} - H_{k+1} \beta^{(k)})$$

$$W_{k+1} = \text{diag}([w_{k+1} \dots w_{k+1}]) \in \mathbb{R}^{(\sum_{l=0}^{k+1} N_l) \times (\sum_{l=0}^{k+1} N_l)} \text{ with } w_{k+1} \in \{w_{k+1}^-, w_{k+1}^+\}$$

We used K_{k+1}^{-1} instead of K_{k+1} for the computation of $\beta^{(k+1)}$. The modified update expression for K_{k+1}^{-1} using matrix inversion lemma is given by

$$\begin{aligned}
K_{k+1}^{-1} &= (K_k + H_{k+1}^T W_{k+1} H_{k+1})^{-1} \\
&= K_k^{-1} - K_k^{-1} H_{k+1}^T (W_{k+1}^{-1} + H_{k+1} K_k^{-1} H_{k+1}^T)^{-1} H_{k+1} K_k^{-1}
\end{aligned}$$

Let $P_{k+1} = K_{k+1}^{-1}$, the update equation for WOS-ELM become

$$\begin{aligned}
P_{k+1} &= P_k - P_k H_{k+1}^T (W_{k+1}^{-1} + H_{k+1} P_k H_{k+1}^T)^{-1} H_{k+1} P_k \\
\beta^{(k+1)} &= \beta^{(k)} + P_{k+1} H_{k+1}^T W_{k+1} (Y_{k+1} - H_{k+1} \beta^{(k)})
\end{aligned} \tag{2.15}$$

Every time a new sample or a new chunk of samples arrives, W_{k+1} is selected to maximize G_{mean} .

Since w_{k+1}^- is always set to 1, no tuning is required upon the arrival of majority class samples in one-by-one learning. Similarly, no tuning is required if the arriving chunk does not contain any minority class sample. The WOS-ELM method is summarized in Algorithm 2-1 (on the next page). As in the online sequential ELM algorithm, the general update equation (2.15) works for both chunk-by-chunk and one-by-one learning since the one-by-one learning can be considered as a special case of chunk-by-chunk learning with chunk size equal to 1.

In summary, the significance of the new method is that it can tackle the CIL problem in chunk-by-chunk as well as one-by-one sequential learning mode. Unlike most data sampling methods, each arriving sample can update the classification model independently without any delay. The previously learnt minority class samples need not be stored in the system. It can also efficiently alleviate the CIL problem when the imbalance ratio is higher than the chunk size, i.e., minority class samples may not be present in most of the chunks.

In the newly proposed method, weights are tuned in a computationally efficient manner for alleviating the class imbalance problem. OTER is originally proposed for one-by-one learning and uses fixed weight setting for total error rate minimization while WOS-ELM covers both chunk-by-chunk and one-by-one learning and tune weights for G_{mean} optimization.

Note that WOS-ELM assumes the class concepts do not change over time [91]. The focus of WOS-ELM in this work is on the class imbalance problem in online learning.

Algorithm 2-1: Weighted online sequential extreme learning machine (WOS-ELM) algorithm

Input: $\{x_i, y_i\}, x_i \in R^d$ and $y_i \in R^q, i = 1, \dots, N$.

Output: $\beta^{(k+1)}$.

Initialization:

Initial data set: $D_0 = \{x_i, y_i\}, x_i \in R^d$ and $y_i \in R^q, i = 1, \dots, N_0$ with $N_0 \in N$.

1. Set hidden nodes L using cross-validation ($L \leq N_0$).

2. Random hidden node parameters: $a_j \in R^d$ and $b_j \in R^1, j = 1, \dots, L$.

3. Initial hidden layer output matrix $H_0 = \begin{bmatrix} G(a_1, b_1, x_1) & \dots & G(a_L, b_L, x_1) \\ \vdots & \dots & \vdots \\ G(a_1, b_1, x_{N_0}) & \dots & G(a_L, b_L, x_{N_0}) \end{bmatrix}_{N_0 \times L}$

4. Initial diagonal weighting matrix W_0 with $w_0^- = 1$ and $w_0^+ = \frac{s_0^-}{s_0^+}$ (i.e. imbalance ratio of D_0)

$$W_0 = \text{diag}([w_0 \dots w_0]) \in R^{N_0 \times N_0} \text{ with } w_0 \in \{w_0^-, w_0^+\}$$

5. Inverse of initial $K_0 \rightarrow (K_0)^{-1} = (H_0^T W_0 H_0)^{-1}$.

6. Estimate initial solution $\beta^{(0)}$ using (2.12).

7. Find G_{mean}, TPR and TNR using $\beta^{(0)}$ on the validation set.

if $TPR = TNR$ **then** %% We consider $TPR = TNR$ if $|TPR - TNR| \leq 1\%$ %%

 Keep $(w_0^-, w_0^+) = (1, \frac{s_0^-}{s_0^+})$.

else if $\{TPR > TNR\}$ **then**

 Tune $w_0^+ \rightarrow \frac{w_0^+}{1.2}, \frac{w_0^+}{1.5}, \frac{w_0^+}{1.8}, \text{ and } \frac{w_0^+}{2}$.

 Select $\beta^{(0)}$ and $(K_0)^{-1}$ yielding highest G_{mean} .

else

 Tune $w_0^+ \rightarrow 1.2w_0^+, 1.5w_0^+, 1.8w_0^+ \text{ and } 2w_0^+$.

 Select $\beta^{(0)}$ and $(K_0)^{-1}$ yielding highest G_{mean} .

end if

Sequential Learning:

New data chunk: $D_{k+1} = \{x_i, y_i\}, i = (\sum_{l=0}^k N_l) + 1, \dots, \sum_{l=0}^{k+1} N_l$.

1. Calculate the hidden layer output matrix H_{k+1}

2. Calculate the diagonal weighting matrix W_{k+1} :

$$W_{k+1} = \text{diag}([w_{k+1} \dots w_{k+1}]) \in R^{(\sum_{l=0}^{k+1} N_l) \times (\sum_{l=0}^{k+1} N_l)} \text{ with } w_{k+1} \in \{w_{k+1}^-, w_{k+1}^+\}$$

$$w_{k+1}^- = 1, w_{k+1}^+ = \frac{s_{k+1}^-}{s_{k+1}^+}$$

3. Find P_{k+1} and New estimate $\beta^{(k+1)}$ using (2.15)

4. Find G_{mean}, TPR and TNR using $\beta^{(k+1)}$ on the validation set.

if $TPR = TNR$ **then** %% We consider $TPR = TNR$ if $|TPR - TNR| \leq 1\%$ %%

 Keep $(w_{k+1}^-, w_{k+1}^+) = (1, \frac{s_{k+1}^-}{s_{k+1}^+})$

else if $\{TPR > TNR\}$ **then**

 Tune $w_{k+1}^+ \rightarrow \frac{w_{k+1}^+}{1.2}, \frac{w_{k+1}^+}{1.5}, \frac{w_{k+1}^+}{1.8}, \text{ and } \frac{w_{k+1}^+}{2}$

 Select $\beta^{(k+1)}$ and P_{k+1} yielding highest G_{mean} .

else

 Tune $w_{k+1}^+ \rightarrow 1.2w_{k+1}^+, 1.5w_{k+1}^+, 1.8w_{k+1}^+ \text{ and } 2w_{k+1}^+$.

 Select $\beta^{(k+1)}$ and P_{k+1} yielding highest G_{mean} .

end if

2.4 Experiments

2.4.1 Datasets

In this section, 15 real-world imbalanced datasets have been used. 14 are selected from the UCI repository [92] and the oil dataset is provided by Dr. Holte [93]. Table 2-3 shows the details of the datasets. Each dataset has been partitioned into a training set, a validation set and a testing set with approximately the same imbalance ratio. Using these datasets we have covered various sizes, imbalanced ratios and number of features. Some of the datasets, including haberman, transfusion, phoneme, oil and pima have only two classes. For abalone9-18, we selected class 18 and 9 as the minority and the majority class respectively. For the rest of the datasets, a single class is chosen as the minority class and the rest of them are combined to make up the majority class. These datasets have been frequently used for the performance evaluation of various CIL algorithms in batch mode.

Table 2-3 Details of the datasets

Dataset	# of Attributes	Imbalance ratio	# of Training samples	# of Testing samples	# of validation samples
page-block5	10	46.59	3473	1000	1000
abalone15	8	39.55	2677	1000	500
yeast5	8	28.09	1034	300	150
car3	6	24.04	1228	300	200
oil	49	21.85	637	200	100
abalone9-18	8	16.4	531	100	100
balance2	4	11.75	385	150	90
satimage4	36	9.27	5835	600	600
mf-morph10	6	9	1500	300	200
mf-zernike10	47	9	1500	300	200
cmc2	9	3.42	973	300	200
transfusion	4	3.2	548	200	100
haberman	3	2.77	246	60	40
phoneme	5	2.4	4204	600	600
pima	8	1.86	668	100	100

2.4.2 Experimental Settings

All the experiments are performed in Matlab by modifying ELM [44] and OS-ELM [8] packages. These experiments are carried out in a PC with 2.66 GHz CPU and 4 GB RAM.

For each dataset, the number of hidden neurons is gradually increased and the optimal number is then selected by performing 50 runs of 10-fold cross validation on the training dataset. In all the experiments, sigmoid function is used as the activation function. Threshold ' τ ' is set to 0 and bias ' η ' is set to 1 as in [89]. All the input data is normalized to be within [-1,1] interval. Due to random settings in the ELM hidden node, all the experiments are repeated 50 times with randomly shuffling the training, the validation and the testing samples. For data sampling methods, the entire procedure is repeated five times, i.e., in total 250 runs are performed for oversampling and undersampling methods. We have used SMOTE [51] as the oversampling method and Informative Undersampling [47] as the undersampling method. Informative undersampling removes noisy and borderline samples before removing more informative samples. For SMOTE, the value of k is set to 5 where k is the number of nearest neighbors calculated for a given original minority class sample. The neighbors also belong to the same minority class. If the chunk contains fewer minority class samples, the value of k is decreased accordingly. Whenever a comparison is performed, the number of hidden neurons, random weights and bias, activation function, initialization set and training, validation and testing samples are kept the same. The size of the initialization set for sequential learning and the number of hidden neurons for each dataset are recorded in Table 2-4. As recommended, the size of the initialization set in online sequential learning is greater than the number of hidden neurons [8].

Table 2-4 Number of hidden neurons and size of initialization set

Dataset	# of Hidden Neurons	Initialization Set Size
page-block5	36	500
abalone15	19	500
yeast5	44	300
car3	190	300
oil	66	200
abalone9-18	26	100
balance2	40	90
satimage4	130	300
mf-morph10	21	300
mf-zernike10	149	300
cmc2	100	200
transfusion	14	100
haberman	9	120
phoneme	170	300
pima	40	200

2.4.3 Classification Results

2.4.3.1 Chunk-by-Chunk Learning

We first compare the performance of WOS-ELM, SMOTE Oversampling with OS-ELM (OS-ELM- SMOTE) and Informative Undersampling with OS-ELM (OS-ELM-UNDER) on all the 15 datasets in chunk-by-chunk mode. The reason for implementing SMOTE and informative undersampling with OS-ELM for online learning is that these two methods are commonly used data sampling techniques for tackling the CIL problem. SMOTE has recently shown good performance in online learning [2]. Also, these methods can be implemented without accessing the previously learnt data. The results obtained by OS-ELM are not shown since it obtains lower G_{mean} than the other three methods for all the datasets. Two different chunk sizes are considered for each dataset. The size is big enough such that it has at least few minority class samples, as required by the data sampling methods. Also, the size is not too big so that we have at least few chunks for sequential learning. For sampling

methods, the chunk size represents the number of original samples (before sampling). G_{mean} values and the training times for the two chunk sizes are reported in Table 2-5 and Table 2-6. The imbalance ratios in all the chunks are similar to that of the overall imbalance ratio. For data sampling methods, various minority class distributions e.g. 35, 40, 45, 50, 55, 60, and 65 are tried for the highest G_{mean} using the validation set in each chunk. Similarly, in WOS-ELM weights are tuned in each chunk for the highest G_{mean} using the same validation set. For a fair comparison, undersampling/SMOTE percentages and weights settings are optimized under identical settings. The highest G_{mean} values obtained on the testing sets are shown in bold face. The training times of WOS-ELM are also shown in bold face. As can be observed from the tables, among the three CIL methods, OS-ELM-UNDER always produce the lowest G_{mean} . While WOS-ELM and OS-ELM-SMOTE produce quite close G_{mean} values with WOS-ELM winning on most datasets. WOS-ELM obtains similar G_{mean} values with different chunk sizes. We also used a chunk size of 10 for three highly imbalanced datasets, i.e., page-block5, abalone15 and yeast5 so that a minority class sample arrives after few chunks comprising of only majority class samples. e.g. with page-block5, after every 4 or 5 chunks of all the majority class samples, a chunk arrives with only one minority class sample. Same as in the case of one-by-one learning, the weights are tuned only when the chunk contains at least one minority class sample since majority class samples are assigned a weight of 1. The G_{mean} values obtained in this case are similar to the bigger chunk sizes. WOS-ELM does not require minority class sample(s) to be present in every chunk. A comparison of G_{mean} values obtained by WOS-ELM and OS-ELM-SMOTE is shown in Figure 2-1.

WOS-ELM has the lowest training time among the three CIL methods for both chunk sizes. The data sampling methods in comparison use sophisticated sample removal/creation procedures in every chunk which increase the training time. Before the training data is fed to the OS-ELM algorithm, it has to be analyzed and processed

e.g. in SMOTE, k -NN rule is used to create synthetic samples. In WOS-ELM, there is no extra cost for data processing.

2.4.3.2 One-by-One Learning

Unlike OS-ELM-SMOTE and OS-ELM-UNDER methods, WOS-ELM can learn one sample at a time also. In WOS-ELM, each sample can update the model without any delay. Its performance is compared with the online total error rate (OTER) method [68]. OTER learns one sample at a time and use fixed weights to minimize the total error rate. OTER is an effective online learning method, especially if the class sizes are different. As can be observed from Table 2-7, on 13 out of 15 datasets, WOS-ELM achieves higher G_{mean} than OTER. A comparison of G_{mean} values obtained by WOS-ELM and OTER is shown in Figure 2-2.

Training time of WOS-ELM is higher than OTER due to the tuning of weights upon the arrival of minority class samples. Since only one minority sample is learnt at a time, the tuning introduces a very small delay in each model update. No tuning is required upon the arrival of majority class samples since the weight is fixed at 1.

For chunk-by-chunk learning (see Table 2-5 and 2-6), no significant difference between WOS-ELM and OS-ELM-SOMTE in terms of generalization performance is observed. However, in terms of computational time WOS-ELM is preferred over OS-ELM-SOMTE. WOS-ELM clearly outperforms OS-ELM-UNDER on all the datasets used. For one-by-one learning (see Table 2-7), the importance of weight tuning is validated. WOS-ELM obtains higher G_{mean} than OTER on almost all the datasets. OTER tries to optimize total error rate using fixed weights which may not be optimal for imbalanced datasets, while WOS-ELM tune weights to optimize G_{mean} , an appropriate CIL performance measure.

2.4.4 Statistical Significance Test

In order to analyze whether WOS-ELM is statistically significant than the other online sequential algorithms in comparison, we used Friedman test [94] for both

chunk-by-chunk and one-by-one learning mode. For chunk-by-chunk mode, since three CIL methods are compared, a post-hoc analysis is also applied.

2.4.4.1 Chunk-by-Chunk mode

WOS-ELM, OS-ELM-SMOTE and OS-ELM-UNDER are evaluated on 15 datasets with two different chunk sizes. Thus, a total of thirty test results are accumulated using each of the online methods. For significance testing, average ranks among the classifiers are calculated. For each dataset, the best performing algorithm is ranked as 1, the second best as 2, and so on. If classifiers produce same result, an average rank is assigned to each of them. We used *Matlab* function '*friedman*' for hypothesis testing. If p -value is lower than a specific confidence level α , null hypothesis is rejected (i.e. all the classifiers are equivalent and any differences among their average ranks are random). In our case, the null hypothesis is rejected at $\alpha = 0.05$ since the p -value for all G_{mean} is 6.6532×10^{-11} . Next a post-hoc Nemenyi test is adopted to observe the differences among the three classifiers. At a certain confidence level, a pair of classifiers is considered significantly different if the rank difference between them is larger than the critical difference (CD). A CD of 0.605 is obtained at $p = 0.05$. The average ranks of WOS-ELM, OS-ELM-SMOTE and OS-ELM-UNDER are 1.23, 1.83 and 2.96 respectively. The rank difference between WOS-ELM and OS-ELM-SMOTE is marginally lower (0.6) than the CD (0.605) so they are considered statistically similar. OS-ELM-UNDER is found to be statistically different (inferior) from the other two methods. Figure 2-3 visualizes the Nemenyi post-hoc test at $p = 0.05$. Statistically similar methods are grouped with horizontal bar [94]. The mean ranks over the datasets for each method are shown in descending rank order on the horizontal axis. Since OS-ELM-SMOTE also performs better than OS-ELM-UNDER on almost all the datasets, the statistical significance test is unable to detect the difference between OS-ELM-SMOTE and WOS-ELM, despite WOS-ELM winning on most of the datasets.

2.4.4.2 One-by-One mode

In this mode, 15 test results are accumulated from both WOS-ELM and OTER. At a confidence level of 0.05, null hypothesis is rejected since obtained p -value is 0.0045. Thus, WOS-ELM is statistically better than OTER at a confidence level of 0.05.

2.4.5 Summary of Experimental Results and Discussion

WOS-ELM is shown to have comparable or better verification performance than existing online sequential algorithms. It outperforms OS-ELM-UNDER on all the datasets. As observed from Figure 2-1, in chunk-by-chunk mode, WOS-ELM and OS-ELM-SMOTE obtain close results with WOS-ELM winning on most of the datasets. The training time of WOS-ELM is found to be lower than the competing data sampling methods. In one-by-one mode, WOS-ELM obtains better performance than OTER on most of the datasets, as shown in Figure 2-2. This improvement is due to adopting the weight tuning instead of using the fixed weights by OTER.

The statistical significance test shows that WOS-ELM is better than OS-ELM-UNDER and similar to OS-ELM-SMOTE in terms of G_{mean} at $\alpha = 0.05$. WOS-ELM is also found to be statistically different from (superior to) the OTER method in terms of G_{mean} at $\alpha = 0.05$.

WOS-ELM is a flexible CIL algorithm that can learn data one-by-one or chunk-by-chunk with any chunk size. Unlike data sampling methods, each sample can update the model without depending on other samples from the same or the different class. It does not involve any extra data processing cost such as finding k -NN or performing clustering. Thus, it is generally faster than the competing data sampling approaches.

Table 2-5 Classification results comparison for chunk-by-chunk learning

Dataset	chunk size	Algorithm	G_{mean}	Training time(s)
page-block5	100	WOS-ELM	91.7	13.482
		OS-ELM-SMOTE	91.5	21.241
		OS-ELM-UNDER	52.7	18.325
	500	WOS-ELM	91.7	8.594
		OS-ELM-SMOTE	91.6	39.64
		OS-ELM-UNDER	50.5	27.74
abalone15	100	WOS-ELM	70.9	5.058
		OS-ELM-SMOTE	69.8	8.081
		OS-ELM-UNDER	70.5	9.846
	500	WOS-ELM	70.9	5.55
		OS-ELM-SMOTE	70.8	10.488
		OS-ELM-UNDER	61	6.507
yeast5	75	WOS-ELM	80.4	1.911
		OS-ELM-SMOTE	79.2	2.486
		OS-ELM-UNDER	52.5	3.27
	150	WOS-ELM	80.2	1.412
		OS-ELM-SMOTE	78.6	2.277
		OS-ELM-UNDER	52.2	2.655
car3	50	WOS-ELM	95.1	3.878
		OS-ELM-SMOTE	93.4	8.257
		OS-ELM-UNDER	52	4.759
	100	WOS-ELM	95	2.367
		OS-ELM-SMOTE	94.1	5.652
		OS-ELM-UNDER	53.4	3.734
oil	50	WOS-ELM	79.9	1.519
		OS-ELM-SMOTE	80.6	5.217
		OS-ELM-UNDER	54.7	2.968
	100	WOS-ELM	79.5	0.949
		OS-ELM-SMOTE	80.4	2.751
		OS-ELM-UNDER	59.6	2.539
abalone9-18	50	WOS-ELM	84.2	0.912
		OS-ELM-SMOTE	77.9	3.454
		OS-ELM-UNDER	65.3	1.834
	100	WOS-ELM	84.3	0.854
		OS-ELM-SMOTE	77.1	1.046
		OS-ELM-UNDER	63.1	1.479
balance2	30	WOS-ELM	67.6	0.812
		OS-ELM-SMOTE	63.5	1.687
		OS-ELM-UNDER	48.7	1.515
	45	WOS-ELM	67.7	0.638
		OS-ELM-SMOTE	63.1	1.462
		OS-ELM-UNDER	48.7	1.391

Table 2-6 Classification results comparison for chunk-by-chunk learning

Dataset	chunk size	Algorithm	G_{mean}	Training time(s)
satimage4	50	WOS-ELM	85.7	24.486
		OS-ELM-SMOTE	85.6	33.364
		OS-ELM-UNDER	57.2	29.966
	300	WOS-ELM	86.2	11.599
		OS-ELM-SMOTE	85.7	25.456
		OS-ELM-UNDER	56.4	18.093
mf-morph10	50	WOS-ELM	92.3	2.147
		OS-ELM-SMOTE	92.1	10.064
		OS-ELM-UNDER	86.7	5.158
	200	WOS-ELM	92.3	1.613
		OS-ELM-SMOTE	92.1	2.894
		OS-ELM-UNDER	81.8	2.798
mf-zernike10	50	WOS-ELM	87.8	5.664
		OS-ELM-SMOTE	88.4	16.924
		OS-ELM-UNDER	56.1	8.047
	200	WOS-ELM	87.5	3.366
		OS-ELM-SMOTE	88.3	5.049
		OS-ELM-UNDER	55.3	6.077
cmc2	50	WOS-ELM	66.1	2.756
		OS-ELM-SMOTE	65.6	3.939
		OS-ELM-UNDER	50.2	4.559
	100	WOS-ELM	66	1.869
		OS-ELM-SMOTE	65.8	3.434
		OS-ELM-UNDER	50.6	3.991
transfusion	20	WOS-ELM	67.9	0.306
		OS-ELM-SMOTE	66.9	1.33
		OS-ELM-UNDER	66.7	0.759
	50	WOS-ELM	67.9	0.554
		OS-ELM-SMOTE	67.1	0.932
		OS-ELM-UNDER	66.5	1.142
haberman	20	WOS-ELM	63.7	0.131
		OS-ELM-SMOTE	63.9	0.607
		OS-ELM-UNDER	60.5	0.355
	60	WOS-ELM	63.6	0.207
		OS-ELM-SMOTE	63.5	0.476
		OS-ELM-UNDER	60.5	0.338
phoneme	50	WOS-ELM	84.1	12.007
		OS-ELM-SMOTE	83.8	19.09
		OS-ELM-UNDER	81.8	16.7
	300	WOS-ELM	84.2	9.432
		OS-ELM-SMOTE	83.8	12.322
		OS-ELM-UNDER	81.2	10.32
pima	50	WOS-ELM	75.6	0.305
		OS-ELM-SMOTE	76.5	0.741
		OS-ELM-UNDER	73.4	0.490
	100	WOS-ELM	76.1	0.209
		OS-ELM-SMOTE	76.4	0.532
		OS-ELM-UNDER	73.6	0.442

Table 2-7 Classification results comparison for one-by-one learning

Dataset	Algorithm	G_{mean}	Training time(s)
page-block5	WOS-ELM	91.7	52.224
	OTER	91.5	24.351
abalone15	WOS-ELM	70.8	5.854
	OTER	69.9	2.972
yeast5	WOS-ELM	80.2	10.853
	OTER	79.1	7.85
car3	WOS-ELM	94.8	27.7
	OTER	93.1	20.014
oil	WOS-ELM	79.6	7.268
	OTER	78.1	5.9
abalone9-18	WOS-ELM	84.4	5.636
	OTER	82.8	3.127
balance2	WOS-ELM	67.6	3.447
	OTER	66.8	2.103
satimage4	WOS-ELM	85.7	197
	OTER	85.3	92.676
mf-morph10	WOS-ELM	92.3	5.298
	OTER	92.1	1.417
mf-zernike10	WOS-ELM	87.7	5.664
	OTER	86.5	1.592
cmc2	WOS-ELM	66	35.671
	OTER	66.2	11.529
transfusion	WOS-ELM	67.9	1.071
	OTER	67.5	0.57
haberman	WOS-ELM	63.7	0.558
	OTER	63.4	0.339
phoneme	WOS-ELM	84.3	239.843
	OTER	84.2	79.512
pima	WOS-ELM	75.6	12.833
	OTER	76	2.661

Figure 2-1: G_{mean} comparison between WOS-ELM and OS-ELM-SMOTE

**Average minority class distribution in OS-ELM-SMOTE*

Figure 2-2: G_{mean} comparison between WOS-ELM and OTER

Figure 2-3: Visualization of Nemenyi post-hoc test. The mean rank for each method over all the datasets is plotted on x -aix. Connected method are not significantly different at $p=0.05$

2.5 Conclusion

In this chapter, an online sequential method for class imbalance learning has been proposed. This method is referred to as weighted online sequential extreme learning machine (WOS-ELM). The new method is different from the typically used data sampling methods in that it can tackle the class imbalance problem in both chunk-by-chunk and one-by-one mode. Moreover, in the proposed WOS-ELM, previously learnt data is not required to be stored in the system. Even the old minority class samples can be discarded. Each arriving sample can update the model without any delay. Thus, the proposed method is better suited for real-time online applications. The computational time of WOS-ELM is also lower than those of the competing data sampling methods. Different from the fixed weight setting in the OTER method for total error rate minimization, the weights in WOS-ELM are tuned for G_{mean} optimization. An appropriate weight setting for CIL is selected in a computationally efficient manner. Faster training speed and comparable or better classification performance of the WOS-ELM algorithm than competing data sampling methods have been verified on 15 real-world imbalanced datasets. The proposed WOS-ELM is also seen to outperform OTER on most of the datasets in terms of G_{mean} .

Chapter 3

ENSEMBLE OF SUBSET ONLINE SEQUENTIAL EXTREME LEARNING MACHINE FOR BINARY CLASS IMBALANCE AND CONCEPT DRIFT

3.1 Introduction

In Chapter 2, WOS-ELM has been shown to effectively tackle the class imbalance problem in both chunk-by-chunk and one-by-one learning. However, the WOS-ELM method proposed in Chapter 2 covers only stationary environments and may not be appropriate for concept drift learning. Moreover, OS-ELM [8] related methods, with random hidden node parameters, may not always adapt well to the new data [95]. Thus, ensemble methods [95]-[97] are generally preferred over single OS-ELM methods [8] [68].

In this chapter, a computationally efficient framework, referred to as ensemble of subset online sequential extreme learning machine (ESOS-ELM), is proposed for binary class imbalance learning from a concept-drifting data stream. The proposed

framework comprises a main ensemble representing short-term memory, an information storage module representing long-term memory and a change detection mechanism to promptly detect concept drifts. In ESOS-ELM, a minority class sample is processed by I classifiers (I is the imbalance ratio) while a majority class sample is processed by a single classifier. The majority class samples are processed in a round robin fashion, i.e., the first majority class sample is processed by the first classifier, the second sample by the second classifier and so on. In this way, classifiers in the ensemble are trained with balanced subsets from the original imbalanced dataset.

Ensemble learning methods are widely used in concept drift learning. Dynamic weighted majority (DWM) [87] is a state-of-the-art ensemble method for concept drift learning with balanced datasets. In DWM, voting weights are decreased proportional to the error rate of the classifier. ESOS-ELM also uses dynamic weighted majority voting for concept drift learning. For tackling the class imbalance problem, ESOS-ELM processes incoming samples in a way that each OS-ELM network is trained with approximately equal number of minority and majority class samples. Unlike DWM, voting weights are updated proportional to an appropriate performance measure for CIL. In recurring environments [3] [69] [70], DWM may not be able to leverage the prior knowledge since old concepts are usually forgotten. To avoid this problem, we propose a novel information storage mechanism, using ELM theory, which is efficient both in terms of memory and computation. A change detection mechanism is also employed in the learning framework to promptly react to both the sudden and gradual drifts. Note that the proposed framework tackles class imbalance and concept drift problems in both the one-by-one and chunk-by-chunk modes.

The new framework achieves better performance than that of $\text{Learn}^{++}.\text{NIE}(gm)$ on most of the datasets used in this chapter. ESOS-ELM achieves this performance with fewer hypotheses than in $\text{Learn}^{++}.\text{NIE}(gm)$. The new method also obtained better performance than DWM in recurring environments. This superiority is due to the ELM-Store module which helps leverage the prior knowledge of old concepts. ESOS-

ELM is also applied on benchmark imbalanced datasets without concept drift. ESOS-ELM outperforms WOS-ELM, OTER and SMOTE for all the 15 imbalanced datasets used in Chapter 2.

This chapter is organized as follows. Section 3.2 presents the details of the ESOS-ELM method. This is followed by experiments for validating the performance of the proposed framework in Section 3.3. Finally, Section 3.4 concludes the chapter.

3.2 Proposed Method

In this section, an ensemble of subset online sequential extreme learning machine (ESOS-ELM) is proposed for class imbalance learning from drifting data stream. As shown in Figure 3-1, the proposed ESOS-ELM method consists of three blocks, the main ensemble block, the ELM-Store block and the change detector block. These blocks are discussed in details as follows.

Figure 3-1: ESOS-ELM for non-stationary recurring environments

3.2.1 The Main Ensemble

Similar to the ensemble of OS-ELM (EOS-ELM) method [95], we initialize several OS-ELM networks in ESOS-ELM, with different random hidden node parameters and the same number of hidden neurons. Some OS-ELM networks adapt faster and better to new data than others. The ensemble output generally provides more

stable solution than that of a single OS-ELM network [95]. EOS-ELM is only for balanced datasets while ESOS-ELM can overcome the class imbalance problem. In the new method, each minority class sample is processed by I classifiers (I is the imbalance ratio) while the majority class samples are processed in a round robin fashion, i.e., first majority class sample is processed by the first classifier, the second sample by the second classifier and so on. Classifiers in ESOS-ELM are trained with approximately equal number of majority and minority class samples. Instead of training every OS-LEM network with all the samples, they are trained with balanced subsets of the data stream.

Depending on whether the imbalance ratio is moderate or high, ESOS-ELM works slightly differently. In this chapter, an imbalance ratio that is equal to or smaller than the ensemble size is considered moderate while an imbalance ratio greater than the ensemble size is considered high. ESOS-ELM maintains two sorted lists of classifiers in ascending order, the first list is sorted with respect to the number of minority class samples processed by each classifier while the second list with respect to the number of majority class samples. If the imbalance ratio is smaller than the ensemble size, the incoming minority class sample(s) is processed by the top I classifiers in the first sorted list. While the majority class samples are distributed in a round robin fashion, starting with the top classifier in the second sorted list. Assume that the imbalance ratio is 5. For one-by-one learning, the minority class sample is processed by top 5 classifiers in the first sorted list while the majority class sample is processed by the top classifier in the second sorted list (see Figure 3-2). The two sorted lists are updated after each time step.

For chunk-by-chunk learning, suppose that the new chunk contains 50 minority class samples and 250 majority class samples. All the minority class samples are processed by the top 5 classifiers in the first sorted list, while 50 majority class samples are processed by 5 classifiers in a round robin fashion, i.e., the first 50 samples are processed by the first classifier in the second sorted list, the next 50 samples by the

second classifier and so on. Even though the original dataset is imbalanced, classifiers in the ensemble are trained with similar number of majority and minority class samples.

If the imbalance ratio is high, i.e., greater than the ensemble size, the arriving minority class sample(s) is processed by all the classifiers. The majority class samples can be processed similarly to the moderate imbalance ratio case. However, for high imbalance ratios e.g. 100, classifiers are still trained with imbalanced subsets with a lower imbalance ratio of 10, if the ensemble size is 10. In such cases, we only select the majority class samples for training which are misclassified by the ensemble.

Dynamic weighted majority (DWM) [87] is a state-of-the-art ensemble method for concept drift learning. However, it has some limitations when used for imbalanced data streams. Firstly, every classifier in the ensemble will be biased towards the majority class since they are trained with imbalanced dataset. Secondly, voting weights are updated with the assumption that errors made by the two classes are equal, i.e., DWM does not consider imbalance in the dataset for its voting weight update. In ESOS-ELM, we also use dynamic weighted majority voting for concept drift learning. Existing classifiers in ESOS-ELM are updated with new data according to the balanced subset approach discussed earlier and shown in Figure 3-2. The output matrices are updated using (2.5). If the current ensemble misclassifies a sample, a new classifier is added using (2.10) to make sure it is not biased towards the majority class. Classifiers can be added and removed at every time step or after every 'p' time steps. For imbalanced datasets, the overall accuracy or total error rate is not an appropriate evaluation measure. If the voting weight is updated considering error from both classes as equal, the ensemble may not be able to promptly detect changes in the minority class. Therefore, voting weights should also be updated differently than the weights in original DWM. In ESOS-ELM, classifier voting weight is multiplied by its G_{mean} value on the current environment. As discussed in Chapter 2, G_{mean} is a commonly used evaluation criterion for CIL which measures the accuracy of both the classes in a

balanced way [32] [26] [25]. The weights are normalized after each learning step so that the highest weight is always 1.

DWM has a potential of adding large number of classifiers, particularly for difficult classification tasks [87]. With constrained computational and memory resources, a limit can be set on the number of classifiers. If the limit is reached, classifier with the lowest weight is replaced with a new one. It was pointed out that placing a limit on the number of classifiers may not have appreciable performance impact [90]. In ESOS-ELM, we also set a limit of 15 classifiers on the ensemble size. However, old concepts which are not relevant now can be forgotten since classifiers trained with old concepts may be removed over time. With cyclic or recurring environments [3] [69] [70], i.e., when old class concepts recur after a long period, DWM usually starts learning concepts from the beginning since it no longer has access to the forgotten information. Prior knowledge of recurring concepts is particularly useful in class imbalance learning since the minority class samples can be rare. Using extreme learning machine theory, we propose a computationally efficient information storage scheme to retain old information. We call this storage scheme as ELM-Store which is discussed as follows.

Figure 3-2: One-by-one sequential learning in ESOS-ELM for imbalanced data streams

3.2.2 ELM-Store for Saving Old Information

In order to leverage the prior knowledge of recurring concepts, previously learnt information should be retained in an efficient way. Adaptive Classifiers-Ensemble systems (ACE) [70] uses an online classifier trained on the current concept (short-term memory) and many batch classifiers for retaining old concepts (long-term memory). Classifiers outputs are combined using weighted majority voting. It uses a long-term buffer to store the most recent samples. A new batch classifier is trained whenever concept drift is detected or the buffer is full. This approach to storing prior information may not be very efficient in terms of both memory and computation. If the long-term buffer is too big, we require a big memory to store old samples. The size of memory may be a constraint in various applications. It also violates the single-pass requirement of incremental learners. If the buffer size is too small, the number of stored hypotheses or batch classifiers grows very fast as ACE does not perform any pruning. In this case, the learning framework becomes computationally very expensive. Using ELM theory, we propose an efficient way to retain old information using batch classifiers. These batch classifiers constitute the long-term memory while the main ensemble, described in the previous subsection represents the current or the short-term memory. Weighted ELM (WELM) is used as the batch classifier for CIL [46]. The information acquired from training samples is accumulated independent of any long-term buffer. A batch classifier is only trained and stored whenever ensemble performance falls below a performance threshold which usually reflects concept drift. Thus, the number of batch classifiers does not grow rapidly. If any stored hypothesis performs better than the current ensemble, indicating possible recurring concept, it is introduced in the ensemble with a weight of 1. We observe from the experiments that stored hypotheses obtaining G_{mean} greater than 95% of the main ensemble G_{mean} help improve the ensemble performance. This is in contrast to DWM which introduces new classifiers trained on the current data and does not make use of the prior knowledge. We hope to

train and store just one (or at most a few) hypothesis per concept, which has the exact performance of a batch learner (e.g. WELM), while computational time and memory requirement are the same or lower than that of an incremental learner (e.g. WOS-ELM). Consider the output weight equation of ELM in (2.3).

$$\beta = H^\dagger Y = (H^T H)^{-1} H^T Y \quad (3.1)$$

At time step '0', let

$$K_0 = H_0^T H_0$$

where H_0 is the initial hidden layer output matrix calculated using initialization set D_0 .

At the subsequent time step

$$K_1 = K_0 + H_1^T H_1$$

where H_1 is the hidden layer output matrix calculated with new data arriving at time step '1'

Generalizing to time step ' k ' ($k = 1, 2, \dots$)

$$K_k = K_{k-1} + H_k^T H_k$$

Similarly,

$$Q_0 = H_0^T Y_0$$

Then

$$Q_1 = Q_0 + H_1^T Y_1$$

$$\vdots$$

$$Q_k = Q_{k-1} + H_k^T Y_k$$

Thus (3.1) can be written as

$$\beta = K_k^{-1} Q_k$$

where $K_k \in R^{L \times L}$, $Q_k \in R^{L \times q}$, L is the number of hidden neurons and q represents the number of classes. In this chapter, we only consider two class problems, i.e., $q = 2$. Note that K_k and Q_k act as buffers for storing information from training samples similar to the long-term buffer in ACE. However, different from ACE's buffer, the sizes of K_k and Q_k are fixed and dependent on the number of hidden neurons rather than the number of samples. Splitting K_k and Q_k according to positive and negative classes

$$K_k = K_k^+ + K_k^-$$

$$K_k^+ = K_{k-1}^+ + (H_k^+)^T H_k^+$$

$$K_k^- = K_{k-1}^- + (H_k^-)^T H_k^-$$

$$Q_k = Q_k^+ + Q_k^-$$

$$Q_k^+ = Q_{k-1}^+ + (H_k^+)^T Y_k^+$$

$$Q_k^- = Q_{k-1}^- + (H_k^-)^T Y_k^-$$

For CIL, we use WELM (2.10) instead of ELM as below:

$$\beta = (H^T W H)^{-1} H^T W Y \quad (3.2)$$

where $W = W_- + W_+$ and $W_- = \text{diag}(\frac{1}{s_k}, \dots, \frac{1}{s_k}, 0, \dots, 0)$

$W_+ = \text{diag}(0, \dots, 0, \frac{1}{s_k^+}, \dots, \frac{1}{s_k^+})$

s_k^- and s_k^+ are negative and positive sizes respectively. Thus K_k and Q_k matrices for WELM becomes

$$K_k = \frac{1}{s_k^+} K_k^+ + \frac{1}{s_k^-} K_k^-$$

$$Q_k = \frac{1}{s_k^+} Q_k^+ + \frac{1}{s_k^-} Q_k^- \quad (3.3)$$

K_k^+ and $K_k^- \in R^{L \times L}$, Q_k^+ and $Q_k^- \in R^{L \times 2}$. K_k^+, K_k^-, Q_k^+ and Q_k^- are collectively referred to as temporary matrices and they represent the ELM-Store.

When a concept drift is detected, a batch classifier (WELM) using (3.2) needs to be trained. Thus, we need to constantly update K_k^+, K_k^-, Q_k^+ and Q_k^- , whenever new samples arrive. These temporary matrices have the dimensions dependent on the number of hidden neurons but not on the number of samples, i.e., even after one million samples, the sizes of these temporary matrices still stay the same. Thus, we can store the information from all the samples using temporary matrices without any long-term buffer or extra memory. Also, the number of WELMs or stored hypotheses increases proportionally with the number of times the concept drift is detected and not with the number of arriving samples. For example, if the concept drift occurs after one million samples belonging to the same concept, a single WELM (sudden drift) or a few

WELMs (gradual drift) are trained, with incrementally updating temporary matrices. For such huge data streams, a large number of batch classifiers will be trained and stored in ACE.

In ELM, matrix inverse is usually the most expensive part in terms of computation. It is only performed once, i.e., when a concept drift is detected. Then, K_k^+, K_k^-, Q_k^+ and Q_k^- are updated in an incremental way. Samples are discarded once the matrices are updated thus still meeting the single-pass requirement. Whenever a new sample or a chunk of samples arrives, at most four matrix additions and multiplications are performed in ELM-Store. Once the concept drift is detected and new β is calculated using (3.2), K_k^+, K_k^-, Q_k^+ and Q_k^- are reset and initialized with the best hidden node parameters in the main ensemble. The counters s_k^- and s_k^+ are also reset. With the ELM-Store technique, we manage to retain information of past concepts without occupying considerable memory space. In short, ELM-Store has the unique feature of storing information with which classifiers can be trained later having

- Computation and memory requirements of incremental learners (updating K_k^+, K_k^-, Q_k^+ and Q_k^-)
- Performance of batch learners ($\beta = K_k^{-1}Q_k$)

3.2.3 Change Detector

In general, there are two types of concept drifts to be detected, sudden drifts and gradual drifts. Sudden drifts are usually easier to detect as compared to gradual ones. In the proposed method, if performance of the main ensemble falls below certain threshold ‘ τ ’, a new hypothesis (WELM) is trained using ELM-Store. ‘ τ ’ is set to 90% of the G_{mean} value from the last evaluation. To make the ensemble react rapidly to sudden drifts, all the existing classifiers stop voting. In order to avoid false alarms,

these classifiers are still kept in the ensemble and continuously updated with new data. Any old classifier yielding better G_{mean} than any of the new classifiers can start voting again. Note that the new batch WELM can also enter the main ensemble in case of false alarm.

Gradual drift is generally not very apparent since the performance degrades slowly. It can also be mistaken as noise. In this chapter, gradual drift is detected using statistical decision theory similar to [98]. Other change detection algorithms can also be used. DWM can handle gradual drifts quite effectively. In ESOS-ELM, the gradual drift detection method does not change the way ensemble works. It is only used to train WELM using ELM-Store. In stationary conditions, a performance evaluation criterion G remains nearly constant. It is assumed that G follows Gaussian distribution. If the concept drifts gradually, G will move from one Gaussian to another Gaussian slowly. Hypothesis testing is used to detect this type of change. In hypothesis testing, there is a null hypothesis $\mathcal{H}_0$ and an alternate hypothesis $\mathcal{H}_1$. $\mathcal{H}_0$ corresponds to no change while $\mathcal{H}_1$ corresponds to a change. Suppose G_i is the G_{mean} of the current evaluation i and that the conditional probability density functions (*pdfs*) of G_i under $\mathcal{H}_0$ and $\mathcal{H}_1$ are $p(G_i|\mathcal{H}_0)$ and $p(G_i|\mathcal{H}_1)$ respectively. Once $p(G_i|\mathcal{H}_0)$ and $p(G_i|\mathcal{H}_1)$ are known, a likelihood ratio test can be performed as below

$$L(G_i) = \begin{cases} \mathcal{H}_0, & \frac{p(G_i|\mathcal{H}_1)}{p(G_i|\mathcal{H}_0)} < \tau \\ \mathcal{H}_1, & \frac{p(G_i|\mathcal{H}_1)}{p(G_i|\mathcal{H}_0)} \geq \tau \end{cases} \quad (3.4)$$

$L(G_i)$ greater than ' τ ' corresponds to alternate hypothesis $\mathcal{H}_1$ which means change is detected. Otherwise there is no change detected. We split the sequence of G_{mean} values obtained from past evaluations into two parts. The first part represents the null hypothesis $\mathcal{H}_0$ with mean μ_0 , variance σ_0^2 and *pdf* $p(G_i|\mathcal{H}_0)$. The second part

represents the alternate hypothesis $\mathcal{H}_1$ with mean μ_1 , variance σ_1^2 and *pdf* $p(G_i|\mathcal{H}_1)$. $p(G_i|\mathcal{H}_0)$ and $p(G_i|\mathcal{H}_1)$ can be estimated assuming Gaussian *pdf* as in (3.5) and (3.6)

$$p(G_i|\mathcal{H}_0) = \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left(-\frac{(G_i-\mu_0)^2}{2\sigma_0^2}\right) \quad (3.5)$$

$$p(G_i|\mathcal{H}_1) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{(G_i-\mu_1)^2}{2\sigma_1^2}\right) \quad (3.6)$$

Once $p(G_i|\mathcal{H}_0)$ and $p(G_i|\mathcal{H}_1)$ are estimated, the likelihood ratio test using (3.4) is performed. A smaller value of ' τ ' can detect small changes but number of batch classifiers can grow very fast. If ' τ ' is too high to detect a change, the algorithm will be the same as DWM, i.e., ELM-Store has no contribution. G_{mean} values that are obtained by the new classifier (added at the time step when last change is detected) over subsequent time steps, form the sequence for calculating $p(G_i|\mathcal{H}_0)$ and $p(G_i|\mathcal{H}_1)$. The test using (3.4) is performed at every alternate evaluation step, starting with at least 6 previous evaluations. If there is no change, the next test is performed with additional G_{mean} values from the next two evaluations, i.e., a sequence of past 8 G_{mean} values from the same classifier.

The main ensemble, ELM-Store and drift detection mechanism are combined and referred to as ESOS-ELM which is summarized in Algorithm 3-1 and also shown in Figure 3-1. The samples are classified using weighted majority voting in the main ensemble. ESOS-ELM can promptly respond to both sudden and gradual drifts. The significance of ESOS-ELM is that it can tackle the class imbalance problem of concept-drifting data streams in both chunk-by-chunk and one-by-one learning. It provides a computationally and memory efficient alternative to the current state-of-the-art chunk-by-chunk learning methods. ESOS-ELM maintains fewer hypotheses to leverage prior knowledge of recurring concepts. Unlike various class imbalance methods, old minority class samples need not be stored.

It is of relevance to point out that ESOS-ELM can also be effectively applied to class imbalance problems without concept drift. WOS-ELM and OTER have been proposed to tackle the class imbalance problem in online sequential learning applications in stationary environments. WOS-ELM and OTER methods are modified versions of OS-ELM, thus, may not always adapt well to the new data due to the non-optimal hidden node parameters problem. Since ESOS-ELM is an ensemble method, it can provide a more stable solution. In the experiments section, we will use ESOS-ELM to sequentially learn the benchmark imbalanced datasets and compare its results with those of the recently proposed methods.

Algorithm 3-1: ESOS-ELM training for moderate imbalance ratiosGiven: $M \rightarrow$ main ensemble size $D_0 = D_0^+ + D_0^- \rightarrow$ initialization set size $x_i \rightarrow$ training sample $y_i \rightarrow$ target label, $i \rightarrow 1$ to N $p \rightarrow$ interval between classifier addition, removal, weight update, and change detection test $\tau \rightarrow$ threshold for change detectionOutput:hypotheses $\rightarrow \beta_u$ hypotheses weight $\rightarrow w_u$, $u \rightarrow 1$ to M Initialization $ir_0 = \text{floor}(n_0^-/n_0^+)$, $c = \text{floor}(n_0^-/M)$ % ir_0 is the imbalance ratio in the initialization chunk% $a=1$, $b=c$, $d=1$, $e=c$, $\text{count}=0$ **for** $u \rightarrow 1, \dots, M$ initialize OS-ELM using (2.4) $\rightarrow \beta_u(x_a^-, \dots, x_b^-, x_d^+, \dots, x_e^+)$ $w_u \rightarrow 1$ $a=a+c$; $b=b+c$ $\text{count}=\text{count}+1$ **if** $\text{count}==ir_0$ $d=d+c$; $e=e+c$; $\text{count}=0$ **end****end**initialize ELM-store(K_k^+ , K_k^- , Q_k^+ and Q_k^-)Online Sequential LearningIndex_p=sort_ascend($\beta_u, +$) %sort classifiers with respect to positive class samples%Index_n=sort_ascend($\beta_u, -$) %sort classifiers with respect to negative class samples% $O \rightarrow$ Ensemble_Classify(x_i)**if** (change_detection_test(τ) == 1) %change detected%

train WELM using (3.3)

re-initialize ELM-Store

else

update ELM-Store

end**if** $y_i == '+'$ **for** $u \rightarrow 1, \dots, I$ %minority class sample processed by I classifiers, $I \rightarrow \text{floor}(\text{imb. ratio})$ % $o \rightarrow$ classify($\beta_{\text{Index}_p(u)}, x_i$) weight_update(o, y_i, p) update OS-ELM using (2.5) $\rightarrow \beta_{\text{Index}_p(u)}(x_i)$ **end****else** $o \rightarrow$ classify($\beta_{\text{Index}_n(1)}, x_i$) weight_update(o, y_i, p) update OS-ELM using (2.5) $\rightarrow \beta_{\text{Index}_n(1)}(x_i)$ %majority class sample processed by single

classifier%

end**if** ($O \sim y_i$ and $i \bmod p = 0$) $M \rightarrow M+1$ initialize β_M using (2.10) $w_M \rightarrow 1$ **End**

Note: For high imbalance ratios, the algorithm is slightly different. A minority class sample is processed by all the classifiers while a majority class sample is processed by the top classifier in the negative sorted list, only if it is misclassified by the ensemble.

3.3 Experiments

The performance of ESOS-ELM is first evaluated for class imbalance learning with concept drift in Section 3.3.1. Later it is also evaluated for class imbalance learning without concept drift in Section 3.3.2.

3.3.1 Classification Results on Benchmark Imbalanced Datasets with Concept Drift

In order to show the effectiveness of ESOS-ELM in changing environments, four non-stationary datasets are used. For all the datasets, main ensemble is initialized with 10 classifiers and an upper limit of 15 is set on the ensemble size. ESOS-ELM has the distinction of learning in both one-by-one and chunk-by-chunk mode over existing methods. Hence, it is first compared with WOS-ELM for class imbalance learning in one-by-one mode. This comparison will show the effectiveness of the new CIL method over WOS-ELM in drifting datasets. Then it is compared with DWM approach to show the effectiveness of ELM-Store in recurring environments. Since DWM is originally for balanced classes, classifiers in DWM are trained in the same way as the classifiers in the main ensemble of ESOS-ELM (Fig. 3-2). This modified version of DWM is referred to as DWM-CIL. Note that we train DWM-CIL identical to ESOS-ELM for the initial part of the dataset and then observe difference between the two in the later part when concepts recur. ESOS-ELM can make use of old knowledge through ELM-Store while DWM-CIL does not have such mechanism. Finally, in chunk-by-chunk mode, the new method is compared with Learn⁺⁺.NIE which is a state-of-the-art CIL method in non-stationary recurring environments. Details of datasets and classification results are given below.

The STAGGER Concepts:

The STAGGER concepts [99] is one of the widely used dataset for performance evaluation of classifiers in the presence of concept drift. The feature space is described by three attributes: size {small, medium, large}, color {red, green, blue} and shape {square, circular, triangle}. There are three data sources as below

- **Target Concept 1 : size = small AND color = red**
- **Target Concept 2 : color = green OR shape = circular**
- **Target Concept 3 : size = medium OR size = large**

We randomly generated 1200 instances belonging to one of the 3 concepts. The first 400 samples belong to Concept 1, the next 400 to Concept 2, and the last 400 to Concept 3. This is an example of sudden concept drift and class 1 is the target class. For Concept 1, imbalance ratio between the two classes is 7 while the remaining two concepts are fairly balanced. This is a very good example of varying imbalance ratios, i.e. initially the classes are imbalanced but in subsequent concepts they become balanced. We repeated the same procedure to generate 1200 more instances in order to simulate recurring environment. In order to evaluate classifiers performance, 2400 instances are randomly generated for testing. Samples are learnt one-by-one. ‘p’ in this example is set to 40, i.e., new classifier can be added and old classifier with the lowest weight can be removed after every 40 samples. The number of hidden neurons is gradually increased and the one yielding the best overall G_{mean} is selected. The best number of hidden neurons is found to be 10. A limitation of the new method in one-by-one learning is that few samples should be available for initializing a new classifier using (2.10). It is preferred to have this number equals to or greater than the number of hidden neurons. In order to meet the single-pass requirement, ELM-Store is used to incrementally store the information of recent most samples.

If the ensemble G_{mean} of the current environment is less than 90% of the last evaluation, all the old classifiers stop voting. This acts as a performance threshold. These classifiers are still updated with new data. Any old classifier achieving higher G_{mean} than any new classifier can start voting again. In this way, the new method can react quickly to the sudden drift. A batch classifier (WELM) is trained whenever overall G_{mean} of the ensemble falls below the performance threshold. This classifier can be added to the ensemble in case of false alarm since any stored WELM with a higher G_{mean} than that of the ensemble is added to the ensemble with a weight of 1. Figures 3-3 and 3-4 compare performance of ESOS-ELM with those of WOS-ELM and DWM-CIL respectively. The average G_{mean} values over the entire dataset are reported in Table 3-1. G_{mean} values in the table are average of 5 trials. ESOS-ELM clearly outperforms WOS-ELM. The performance improvement over DWM-CIL (see time steps 30,40 and 50 in Fig. 3-4) is due the fact that ESOS-ELM can make use of old knowledge of recurring concepts, using ELM-Store, which is completely or partially forgotten by DWM-CIL method. This improvement is achieved with storing 5 extra hypotheses (WELM).

Figure 3-3: ESOS-ELM and WOS-ELM comparison on STAGGER Concepts

Figure 3-4: ESOS-ELM and DWM-CIL comparison on STAGGER Concepts

The SEA Concepts:

The synthetic problem proposed by Street and Kim [85] has become a benchmark in the concept drift problem. The dataset consist of three attributes, two of them are relevant while the third one is noise. Values of the two relevant attributes (x_1 and x_2) are uniformly distributed between 0 and 10. The samples belong to class 1 if $x_1 + x_2 \leq b$ and class 2 otherwise, where b is the threshold. Similar to [3] threshold is varied from 4 to 7 twice to simulate recurring environment. Concept drift occurs after every 50,000 samples. This is also an example of sudden concept drift. In total, there are 200,000 samples for training and 200,000 for testing. Originally SEA does not have class imbalance problem so the target class is undersampled. The imbalance ratio is varied between 4 and 10. The class imbalance problem is overcome in one-by-one learning mode as shown in Figure 3-2. The best number of hidden neurons is found to be 120. ‘p’ is set to 1000.

Whenever change is detected, a batch WELM is trained using ELM-Store. The performance threshold is set to 90% of G_{mean} from the last evaluation. Figures 3-5 and 3-6 compare ESOS-ELM with WOS-ELM and DWM-CIL respectively. The average G_{mean} values over the entire dataset are reported in Table 3-1. ESOS-ELM performs better than WOS-ELM. ESOS-ELM also performs slightly better than DWM-CIL due

to ELM-Store. The table shows average G_{mean} values over entire period while the difference between ESOS-ELM and DWM-CIL is only in the second half of the entire period. The average G_{mean} values in the second half (when concepts re-occur) obtained by ESOS-ELM and DWM-CIL are 84.3 and 83.8 respectively. Whenever concepts drift in the second half, ESOS-ELM responds faster than DWM-CIL (see time steps 100-110 and 150-160 in Fig. 3-6) using ELM-Store. This gain is achieved by storing only 3 extra hypotheses (WELM).

Figure 3-5: ESOS-ELM and WOS-ELM comparison on SEA Concepts

Figure 3-6: ESOS-ELM and DWM-CIL comparison on SEA Concepts

Rotating Spiral Dataset:

Rotating spiral is a recently introduced synthetic dataset [3] [100] consisting of four rotating spirals, two for each class. The spirals rotate over 600,000 instances with environment recurring every 300,000 instances, i.e., two complete rotations. Entire feature space is considered in the experiments. Rotating spirals change the decision boundary continuously thus simulating gradual drift. The best number of hidden neurons is found to be 120. Originally both classes have equal number of samples so class 1 is undersampled to achieve an imbalance ratio of 10. Samples are learnt one-by-one and ‘p’ is set to 2000.

Hypothesis testing is used for detecting gradual drift. In this example, G_{mean} values obtained by the new classifier (added at the time step when change is detected) over subsequent time steps form the sequence for calculating $p(G_i|\mathcal{H}_0)$ and $p(G_i|\mathcal{H}_1)$. A likelihood ratio test using (3.4) is then performed with ‘ τ ’ set to 1.5. The test is performed at every alternate time step starting with 6 previous evaluations. If there is no change, the next test is performed with additional G_{mean} values from the next two evaluations, i.e., a sequence of past 8 G_{mean} values. For this dataset, changes are detected mostly when sequence size is 10 or 12. Through likelihood ratio test, 15 batch WELMs are trained using ELM-Store over one complete rotation of spirals. Figures 3-7 and 3-8 compare performance of ESOS-ELM with those of WOS-ELM and DWM-CIL respectively. The average G_{mean} values over entire dataset are reported in Table 3-1. The average G_{mean} values obtained by ESOS-ELM and DWM-CIL after the concepts re-occur are 80.8 and 80.3 respectively. WOS-ELM performs poorly on this gradually drifting dataset.

Figure 3-7: ESOS-ELM and WOS-ELM comparison on Rotating Spiral

Figure 3-8: ESOS-ELM and DWM-CIL comparison on Rotating Spiral

Electricity Pricing Domain (elect2):

The ‘elect2’ was first used by Harris [101] and has already become a real-world benchmark dataset for concept drift learning. This dataset was acquired from electricity supplier from New South Wales (NSW), Australia. It consists of 45,312 instances, each with 5 attributes. The first attribute is the day of week (1-7), followed by the period of day (1- 48). The remaining three attributes represents NSW electricity demand, Victoria electricity demand, and the amount of electricity scheduled for transfer between the two states. The classification goal is to predict whether the NSW price will be higher than Victoria’s price. The dataset contains natural concept drift. However, the original dataset has an imbalance ratio of less than 2 so we undersampled the minority class (class 1) such that imbalance ratio between the two classes is around 13.

Samples are presented one at a time. Instead of evaluating the learner's performance with each arriving sample, we evaluate whenever a minority class sample arrives, with 50 subsequent samples forming the validation set. Fig. 3-9 shows the performance comparison between ESOS-ELM and WOS-ELM. The average G_{mean} values over the entire dataset are reported in Table 3-1. ESOS-ELM and DWM-CIL obtained similar results. Concept drift and high imbalance make classification difficult for this dataset. G_{mean} values obtained using WOS-ELM fall to zero several times. The fluctuation in G_{mean} values obtained using ESOS-ELM is due to the fact that validation set contains very few minority class samples. Correct and wrong predictions of these samples obtain significantly different G_{mean} values. A bigger validation set can have many more minority class samples. However, it is a real-world online learning problem with high class imbalance in which minority class samples may not be available at each time step.

Figure 3-9: ESOS-ELM and WOS-ELM comparison on elect2 dataset

ESOS-ELM obtained significantly better performance than WOS-ELM on STAGGER concepts and Rotating Spiral datasets while the improvement is minor for the remaining two datasets. Clearly, WOS-ELM is not stable for non-stationary

environments and need of an ELM related CIL method for non-stationary environments is justified. In recurring environments, ESOS-ELM responds faster to change in concepts than DWM-CIL.

Table 3-1 Classification results comparison for non-stationary environments (one-by-one)

Dataset	Algorithm	G_{mean}
STAGGER	DWM-CIL	97.2
	WOS-ELM	67.1
	ESOS-ELM	98.1
SEA	DWM-CIL	83.8
	WOS-ELM	82.6
	ESOS-ELM	84.0
Spiral	DWM-CIL	80.2
	WOS-ELM	52.8
	ESOS-ELM	80.5
Elec2	DWM-CIL	70.5
	WOS-ELM	65.6
	ESOS-ELM	70.5

For chunk-by-chunk case, ESOS-ELM is compared with a recently proposed Learn⁺⁺.NIE [3] method. Learn⁺⁺.NIE has shown superior generalization performance to earlier methods in recurring environments. Similar to ESOS-ELM, Learn⁺⁺.NIE does not store old minority class samples. For a fair comparison, we implemented Learn⁺⁺.NIE with ELM as the base classifier. In order to tackle the class imbalance problem, Learn⁺⁺.NIE trains a sub-ensemble of ELM for each data chunk. The number of classifiers in every sub-ensemble is set to 5. The numbers of hidden neurons for ELM are same as in ESOS-ELM except for Rotating Spiral dataset where 90 hidden neurons obtained the best average G_{mean} . The parameters a and b in Learn⁺⁺.NIE are set to 0.5 and 10 respectively [3]. Chunk sizes for STAGGER concepts, SEA concepts and Rotating Spiral are 40, 1000, and 2000 respectively. For elect2 dataset, chunk of 200 samples is used for training and the next 200 samples are used for testing. The same train and test procedure is continued for the remaining samples. At every time step, identical chunk of data is presented to both the methods in comparison. Learn⁺⁺.NIE exists for three different CIL measures, i.e., G_{mean} , F -

measure and *weighted recall measure*. We selected G_{mean} for determining voting weight of the sub-ensemble since ESOS-ELM also uses G_{mean} for its weight calculation. Moreover, other ELM related class imbalance learners such as WELM [46], OTER [68] and WOS-ELM have been successfully evaluated with G_{mean} as the error measure. Table 3-2 shows G_{mean} values obtained by the two methods, along with the maximum number of hypotheses maintained by them. Figures 3-10, 3-11, 3-12 and 3-13 show the performance comparison between ESOS-ELM and Learn⁺⁺.NIE.

Table 3-2 Classification results comparison for non-stationary environments (chunk-by-chunk)

Dataset	Algorithm	G_{mean}	max no. of hypotheses
STAGGER	Learn ⁺⁺ .NIE(gm)	98.6	300
	ESOS-ELM	98.3	20
SEA	Learn ⁺⁺ .NIE(gm)	82.8	1000
	ESOS-ELM	83.9	18
Spiral	Learn ⁺⁺ .NIE(gm)	78.8	1500
	ESOS-ELM	80.7	44
Elect2	Learn ⁺⁺ .NIE(gm)	58.5	340
	ESOS-ELM	61.1	15

Figure 3-10: ESOS-ELM and Learn⁺⁺.NIE(gm) comparison on STAGGER concepts

Figure 3-11: ESOS-ELM and Learn++.NIE(gm) comparison on SEA concepts

Figure 3-12: ESOS-ELM and Learn++.NIE(gm) comparison on Rotating Spiral

Figure 3-13: ESOS-ELM and Learn++.NIE(gm) comparison on elect2

For STAGGER concepts, Learn++.NIE obtained slightly better G_{mean} than ESOS-ELM. For SEA concepts, superior performance of the new method is mainly due to the fact that it responds quickly to sudden drift, particularly after time step 50 (see Fig. 3-11). Finally, ESOS-ELM maintains higher G_{mean} on Rotating Spiral and

elect2 dataset. These are concept drift learning problems with higher imbalance ratios than the previous two datasets. We observed that ESOS-ELM tackles the class imbalance problem more effectively compared to $\text{Learn}^{++}.\text{NIE}(gm)$ for these highly imbalanced datasets. Moreover, it can be observed from Tables 3-1 and 3-2, ESOS-ELM obtains similar results for both chunk-by-chunk and one-by-one learning modes. The sensitivity of number of hidden neurons on the performance of ESOS-ELM for each dataset is shown in Figure 3-14.

Figure 3-14: Parameter tuning in ESOS-ELM

ESOS-ELM is primarily proposed for imbalanced datasets. It is of interest to see how the proposed method behaves on balanced datasets. Thus, we compare it with the original DWM method on datasets having relatively balanced classes. The majority classes in STAGGER and SEA datasets are undersampled first to make the class sizes equal while the original Rotating spiral dataset already has balanced classes. All the three datasets have recurring concepts. The results are recorded in Table 3-3. ESOS-ELM obtains better performance than that of the original DWM even on balanced datasets. As explained earlier for the imbalanced case, ESOS-ELM responds faster to

the concept drifts in the second half (recurring parts) than DWM which does not leverage prior knowledge of past concepts.

Table 3-3 Classification results comparison for balanced datasets

Dataset	Algorithm	G_{mean}
STAGGER (balanced)	DWM	94.1
	EOS-ELM-CIL	94.8
SEA (balanced)	DWM	83.4
	EOS-ELM-CIL	83.7
Sprial (balanced)	DWM	80.3
	EOS-ELM-CIL	80.5

3.3.2 Classification Results on Benchmark Imbalanced Datasets without Concept Drift

The performance of ESOS-ELM is also evaluated on the same 15 real-world imbalanced datasets that were previously used for performance evaluations of WOS-ELM, OTER and OS-ELM-SMOTE in Chapter 2. The details of these datasets are shown in Table 2-3. For a fair comparison, all the experimental settings, i.e., validation set, testing set, chunk size and the number of hidden neurons are set identical to those of above CIL methods.

We first compare the performance of ESOS-ELM with those of WOS-ELM and OS-ELM-SMOTE in chunk-by-chunk mode. Same as for non-stationary datasets, we start with an ensemble size of 10. However, in some datasets the number of samples available for initialization of an individual classifier is very less than the number of hidden neurons. In such cases, a smaller ensemble size is selected. The smallest ensemble size used in this chapter is 5. If sufficient samples are available for initialization, we recommend using ensemble size of 10. For sequential learning in stationary environments, evaluation is performed once at the end rather than at each time step. Voting weights are updated using G_{mean} values obtained on the validation set. The results for chunk-by-chunk mode are recorded in Table 3-4. The new method

obtained better than or similar results to those of WOS-ELM and OS-ELM-SMOTE on all the datasets. We also compare performance of ESOS-ELM with those of WOS-ELM and OTER in one-by-one mode. The results for one-by-one learning are recorded in Table 3-5. As can be observed from the table, the new method has outperformed previous best methods in one-by-one learning. A comparison of G_{mean} values obtained by ESOS-ELM and WOS-ELM (previous best for one-by-one learning) is shown in Fig. 3-15. Note that the classification results for WOS-ELM, OS-ELM-SMOTE and OTER are taken from Chapter 2.

Friedman test is used to confirm whether improvements achieved in Tables 3-4 and 3-5 are statistically significant than the other online sequential algorithms in comparison. For both chunk-by-chunk and one-by-one learning, a post-hoc analysis is also applied since more than two CIL methods are compared.

For each dataset, the best performing algorithm is ranked as 1, the second best as 2, and so on. If classifiers produce same results, an average rank is assigned to each of them. For each classifier, mean rank over all the 15 datasets is calculated. Matlab function '*friedman*' is used for hypothesis testing. A null hypothesis that all the classifiers are equivalent and any differences among their average ranks are random is rejected, if the obtained p -value is lower than a specific confidence level α . For chunk-by-chunk case, the p -value obtained for all G_{mean} values is 3.372×10^{-6} thus a null hypothesis is rejected at $\alpha = 0.05$. In order to analyze differences among ESOS-ELM, WOS-ELM and OS-ELM-SMOTE, a post-hoc Nemenyi is adopted. The average ranks of ESOS-ELM, WOS-ELM and OS-ELM-SMOTE are 1, 2.2 and 2.8 respectively. If the rank difference between a pair of classifiers is larger than the critical difference (CD) at a certain confidence level, the two classifiers are considered statistically different. A CD of 0.855 is obtained at $p=0.05$. The rank difference between ESOS-ELM and WOS-ELM is greater than CD so they are statistically different while the rank difference between WOS-ELM and OS-ELM-SMOTE is smaller than CD so they are statistically similar. Fig. 3-16 visualizes the Nemenyi post-hoc test at $p = 0.05$. The

mean ranks over 15 datasets are shown in descending rank order on the x -axis. Statistically, similar methods, WOS-ELM and OS-ELM-SMOTE in this case are grouped with horizontal bar.

For one-by-one case, the p -value obtained for all G_{mean} values is 2.4189×10^{-6} thus null hypothesis is rejected at $\alpha = 0.05$. The average ranks of ESOS-ELM, WOS-ELM and OTER are 1, 2.866 and 2.133 respectively. The rank difference between ESOS-ELM and WOS-ELM is found to be greater than CD so they are statistically different, while the rank difference between WOS-ELM and OTER is found to be smaller than CD so they are statistically similar. Fig. 3-17 visualizes the Nemenyi post-hoc test at $p = 0.05$. Statistically, similar methods, WOS-ELM and OTER in this case are grouped with horizontal bar.

Figure 3-15: G_{mean} comparison between ESOS-ELM and WOS-ELM

Table 3-4 Classification results comparison for stationary environments (chunk-by-chunk)

Dataset	chunk size	Algorithm	G_{mean}
page-block5	500	WOS-ELM	91.7
		OS-ELM-SMOTE	91.6
		ESOS-ELM	92.4
abalone15	500	WOS-ELM	70.9
		OS-ELM-SMOTE	70.8
		ESOS-ELM	73.7
yeast5	150	WOS-ELM	80.2
		OS-ELM-SMOTE	78.6
		ESOS-ELM	82.2
car3	100	WOS-ELM	95
		OS-ELM-SMOTE	94.1
		ESOS-ELM	95.9
oil	100	WOS-ELM	79.5
		OS-ELM-SMOTE	80.4
		ESOS-ELM	83.2
abalone9-18	100	WOS-ELM	84.3
		OS-ELM-SMOTE	77.1
		ESOS-ELM	87.1
balance2	45	WOS-ELM	67.7
		OS-ELM-SMOTE	63.1
		ESOS-ELM	69.6
satimage4	300	WOS-ELM	86.2
		OS-ELM-SMOTE	85.7
		ESOS-ELM	86.3
mf-morph10	200	WOS-ELM	92.3
		OS-ELM-SMOTE	92.1
		ESOS-ELM	92.5
mf-zernike10	200	WOS-ELM	87.5
		OS-ELM-SMOTE	88.3
		ESOS-ELM	89
cmc2	100	WOS-ELM	66
		OS-ELM-SMOTE	65.8
		ESOS-ELM	66.1
transfusion	50	WOS-ELM	67.9
		OS-ELM-SMOTE	67.1
		ESOS-ELM	68.3
haberman	60	WOS-ELM	63.6
		OS-ELM-SMOTE	63.5
		ESOS-ELM	66
phoneme	300	WOS-ELM	84.2
		OS-ELM-SMOTE	83.8
		ESOS-ELM	84.3
pima	50	WOS-ELM	75.6
		OS-ELM-SMOTE	76.5
		ESOS-ELM	77.1

Table 3-5 Classification results comparison for stationary environments (one-by-one)

Dataset	Algorithm	G_{mean}
page-block5	WOS-ELM	91.7
	OTER	91.5
	ESOS-ELM	92.4
abalone15	WOS-ELM	70.8
	OTER	69.9
	ESOS-ELM	73.6
yeast5	WOS-ELM	80.2
	OTER	79.1
	ESOS-ELM	82.4
car3	WOS-ELM	94.8
	OTER	93.1
	ESOS-ELM	95.9
oil	WOS-ELM	79.6
	OTER	78.1
	ESOS-ELM	83.4
abalone9-18	WOS-ELM	84.4
	OTER	82.8
	ESOS-ELM	87.1
balance2	WOS-ELM	67.6
	OTER	66.8
	ESOS-ELM	69.5
satimage4	WOS-ELM	85.7
	OTER	85.3
	ESOS-ELM	86.3
mf-morph10	WOS-ELM	92.3
	OTER	92.1
	ESOS-ELM	92.5
mf-zernike10	WOS-ELM	87.7
	OTER	86.5
	ESOS-ELM	88.9
cmc2	WOS-ELM	66
	OTER	66.2
	ESOS-ELM	66.3
transfusion	WOS-ELM	67.9
	OTER	67.5
	ESOS-ELM	68.2
haberman	WOS-ELM	63.7
	OTER	63.4
	ESOS-ELM	66.2
phoneme	WOS-ELM	84.3
	OTER	84.2
	ESOS-ELM	84.4
pima	WOS-ELM	75.6
	OTER	76
	ESOS-ELM	77.2

Post-hoc (Nemenyi) test of G_{mean} performance for chunk-by-chunk learning

Figure 3-16: Visualization of Nemenyi post-hoc test for chunk-by-chunk learning. The mean rank for each method over all the datasets is plotted on x -axis. Connected method are not significantly different at $p=0.05$

Post-hoc (Nemenyi) test of G_{mean} performance for one-by-one learning

Figure 3-17: Visualization of Nemenyi post-hoc test for one-by-one learning. The mean rank for each method over all the datasets is plotted on x -axis. Connected method are not significantly different at $p=0.05$

As can be observed from Figures 3-16 and 3-17, ESOS-ELM is statistically superior to the previous best methods in both chunk-by-chunk and one-by-one learning

modes. In Chapter 2, WOS-ELM was found to be statistically superior to the other methods only in one-by-one but not in chunk-by-chunk.

3.3.3 Discussion on Results and Computational Complexity

ESOS-ELM obtains better performance than the recently proposed $\text{Learn}^{++}.\text{NIE}(gm)$ method on 3 out of 4 non-stationary datasets used in this chapter. From the experiments, it is observed that ESOS-ELM tackles the class imbalance problem better than $\text{Learn}^{++}.\text{NIE}(gm)$ and achieves comparable or slightly lower boost in performance in recurring environments using fewer hypotheses. The new method uses similar approach as Dynamic weighted majority (DWM) voting method for tackling concept drifts. However, using ELM-Store we are able to achieve faster response to concept drift than DWM with recurring concepts. The new method obtains similar results in both one-by-one and chunk-by-chunk learning. Moreover, we have shown that ESOS-ELM is more effective than WOS-ELM for tackling the class imbalance problem, in both stationary and non-stationary environments.

For non-stationary datasets with recurring concepts, the new method shows improvement over DWM-CIL. At each time step, ELM-Store updates four matrices which is actually same (if not less) as single OS-ELM, in terms of computational complexity. Only 5 extra hypotheses (WELM) are stored for the STAGGER concepts, 3 for the SEA concepts and 29 extra hypotheses for the Rotating Spiral dataset. These numbers are much smaller than in $\text{Learn}^{++}.\text{NIE}(gm)$ [3] which stores 5 hypotheses per chunk of data. The differences in numbers of stored hypotheses can be seen in Table 3-2. Thus, ESOS-ELM provides a computationally efficient alternative to $\text{Learn}^{++}.\text{NIE}(gm)$ in recurring environments, with the added advantage of learning samples one-by-one. Similar to $\text{Learn}^{++}.\text{NIE}(gm)$, our method does not store old minority class samples.

In this study we only used G_{mean} for determining voting weights in the ESOS-ELM method. We wish to explore ESOS-ELM with other CIL error criteria such as F-measure, area under ROC and weighted recall measure etc. as in Learn++.NIE. Most of the recent Learn++ algorithms can add new classes in the middle of the learning process. We plan to add this capability to ESOS-ELM as well. Moreover, it will be interesting to investigate if changing number of hidden neurons with changing concepts has any significant performance gain. Although, we suspect that it would come with higher computational complexity.

3.4 Conclusion

In this chapter, we have proposed an ensemble of subset online sequential extreme learning machine (ESOS-ELM) for class imbalance learning from drifting data streams. ESOS-ELM consists of a main ensemble for classification in the current imbalanced environment, an ELM-Store module for storing information of old concepts and a change detector for promptly detecting concept drifts. In ESOS-ELM, the main ensemble is trained with balanced subsets of the data stream. Similar to WOS-ELM, the new method works in both one-by-one and chunk-by-chunk learning modes. However, WOS-ELM only considers stationary environments while ESOS-ELM can tackle class imbalance problem in both stationary and non-stationary environments, i.e., with or without concept drift. For recurring environments, the prior knowledge of old concepts is retained using the ELM-Store module. ELM-Store is a computationally efficient storage scheme derived from ELM theory. In non-stationary environments, the proposed framework obtained better or comparable performance to that of Learn++.NIE(gm) using G_{mean} as the figure of merit. It is a computationally efficient alternative to Learn++.NIE(gm) since it maintains significantly lower number of hypotheses. ESOS-ELM outperforms WOS-ELM, OTER and SMOTE on all 15 imbalanced datasets used in the previous chapter.

Chapter 4

META-COGNITIVE ONLINE SEQUENTIAL EXTREME LEARNING MACHINE FOR BINARY CLASS IMBALANCE AND CONCEPT DRIFT

4.1 Introduction

ESOS-ELM and Learn⁺⁺.NIE methods are primarily proposed for data streams with imbalanced classes and drifting concepts, and may not always be as efficient as the methods specific to balanced and stationary applications. For example, Learn⁺⁺.NIE continuously adds new classifiers to the ensemble to cope with the changing environment. For stationary environments, standard OS-ELM is computationally much more efficient than Learn⁺⁺.NIE method. Moreover, ESOS-ELM may not be the preferred choice for imbalanced data streams which become balanced over time or vice versa since it does not reduce to ensemble of OS-ELM (EOS-ELM) [95] or to standard OS-ELM which are commonly used ELM [44] methods for balanced data streams. Application specific methods can be used

interchangeably while learning from data streams for better generalization performance. However, it is difficult to predict the nature of the data stream *a priori*. Thus, an automated or self-regulatory method needs to be implemented to adapt the learning according to the nature of the data stream.

Meta-cognitive learning [78][79][102][103] has been used to self-regulate the learning process. It decides what samples to learn, when to learn and how to learn. A recently proposed multiclass ELM based method, Meta-cognitive ELM (McELM) [104] obtains better generalization performance than self-regulatory resource allocation network (SRAN), ELM, SVM and recently developed meta-cognitive neural network [102]. However, there is no mechanism to tackle the concept drift problem in McELM. Although imbalance learning was discussed in [104], it is also not suitable for highly imbalanced data streams since samples from minority and majority classes are treated similarly.

Meta-cognitive learning can be utilized to tackle the class imbalance problem more effectively by considering two conditions. First, the focus is given to the minority class samples. Second, samples which are difficult to learn are treated differently from other samples of the same class. Note that McELM only considers the second condition while ESOS-ELM only considers the first. Depending on certain knowledge measures such as maximum hinge-loss error, the predicted class label and a suitable class imbalance learning (CIL) performance measure, SMOTE [51] oversampling, undersampling [47] or cost-sensitive learning [52] can be selected to tackle the class imbalance problem. Similarly, for tackling the concept drift problem, window adaptation and drift detection mechanism can be embedded in the meta-cognitive framework. Importantly, class imbalance and concept drift learning can be reduced to standard OS-ELM or McELM method for balanced and stationary data streams. In this chapter, we propose a meta-cognitive online sequential extreme learning machine (MOS-ELM) for binary class imbalance and concept drift. It uses meta-cognitive learning to select the appropriate strategy for class imbalance, concept drift and the

combined problem. Moreover, a unifying equation is proposed as the first OS-ELM output update equation that can be applied to both class imbalance and concept drift learning. It also reduces to standard OS-ELM equation for balanced and stationary data streams.

This chapter is organized as follows. Section 4.2 discusses the preliminaries. Section 4.3 presents the details of the MOS-ELM method. This is followed by experiments for validating the performance of the proposed framework in Section 4.4. Finally, Section 4.5 concludes the chapter.

4.2 Preliminaries

4.2.1 Meta-Cognitive Learning & Knowledge Measures

Figure 4-1: Nelson and Narens model of meta-cognition

Meta-cognitive or self-regulated learning algorithms are inspired by the Nelson and Narens model of human meta-cognition as shown in Figure 4-1 [78] [79]. Meta-cognition means cognition about cognition, essentially providing knowledge about knowledge. It improves and controls the cognitive process by selecting the best learning strategy among various strategies stored in the memory. A meta-cognitive framework consists of a cognitive component which constitutes the object-level processes and a meta-cognitive component which constitutes the meta-level processes.

Object-level processes are monitored and controlled by feedforward and feedback loops with meta-level processors. The flow of information from meta-cognitive component to cognitive component is called control signal and the flow in the reverse direction is called monitory signal. The meta-cognitive component has a dynamic model of the cognitive component and a self-regulatory mechanism to decide what to learn, when to learn and how to learn based on the relative knowledge of the cognitive component. The meta-cognitive component compares the information presented in the sample with that of the cognitive component using certain knowledge measures. Knowledge measures used in this work include the class label predicted by the cognitive component, the maximum hinge-loss error and a CIL performance measure, geometric mean. The knowledge measures are defined as follows

a) Predicted Class label

For a ‘ q ’ class problem, the predicted class label ‘ $\hat{c}$ ’ is obtained using the predicted output ‘ o ’ as below

$$\hat{c} = \arg \max_{j \in \{1,2,\dots,q\}} o^j$$

b) Maximum Hinge-loss Error

The hinge-loss is a loss/cost function used for training classifiers. The hinge-loss error function helps to estimate the posterior probability more accurately than the mean squared error [103] [104] [105] for classification problems. Assume that the class labels are coded as below

$$y^j = \begin{cases} 1, & \text{if } c = j \\ -1, & \text{otherwise} \end{cases} \quad j = 1, 2, \dots, q$$

where ‘ c ’ is the actual class label.

Now the hinge-loss error is defined as follows

$$e^j = \begin{cases} 0, & \text{if } y^j o^j > 1 \\ y^j - o^j, & \text{otherwise} \end{cases} \quad (4.1)$$

The maximum absolute hinge-loss error (E) is given by

$$E = \max_{j \in \{1, 2, \dots, q\}} |e^j|$$

c) *Geometric Mean or G_{mean}*

For binary class problem, G_{mean} is defined in Section 2.3.1.

4.3 Proposed Method

In this section, meta-cognition will be utilized to effectively tackle the class imbalance problem, the concept drift problem and the combined problem of imbalanced and non-stationary class distributions. An OS-ELM based output update equation is also proposed for class imbalance and non-stationary learning.

The new meta-cognitive framework is shown in Figure 4-2. The cognitive component is an ELM classifier while the meta-cognitive component has a dynamic model of the cognitive component, a change detector and a pool of learning strategies. As can be seen from Figure 4-2, a particular learning strategy is selected depending on the knowledge measures in the new meta-cognitive framework. Class imbalance learning strategies include undersampling, SMOTE oversampling and cost-sensitive weighting. For concept drift learning, adaptive windowing and change detection mechanism are used. In order to have flexible architecture in non-stationary environments, hidden layer neuron addition is also allowed during sequential learning. The details of the proposed method are given as follows.

First, a hybrid approach comprising data sampling and cost-sensitive learning is proposed for class imbalance problem in Section 4.3.1. Then, change detection and adaptive window scheme for concept drift are discussed in Section 4.3.2. In Section 4.3.3, an output update equation is derived which unifies OS-ELM based class

imbalance and concept drift learning algorithms. A single equation can be used with data streams having balanced or imbalanced, stationary or non-stationary distributions.

Figure 4-2: Meta-cognitive learning for class imbalance and concept drift

4.3.1 Hybrid Approach for Class Imbalance Learning

To effectively tackle the class imbalance problem, two conditions are considered. First, the focus is given to the minority class samples. Second, samples which are difficult to learn are treated differently from other samples of the same class. For highly imbalanced data streams, a hybrid CIL approach is proposed to focus more on minority class samples. The hybrid approach combines undersampling, oversampling and cost-sensitive learning within the meta-cognitive framework to consider both conditions. Let S_k^+ and S_k^- represent minority and majority classes respectively at time step ' k ' while s_k^+ and s_k^- are the total numbers of minority class samples and majority class samples respectively up till time step ' k '.

The synthetic minority class samples generated using SMOTE [51] oversampling help a classifier to create larger and less specific decision regions. Smaller and more specific regions may result in overfitting. However, SMOTE can become computationally expensive particularly for online learning applications. It has been shown in Chapter 2 that weighted OS-ELM (WOS-ELM), a cost sensitive approach, tackles the class imbalance problem in a computationally efficient way. It assigns higher misclassification cost to the minority class samples as compared to the majority class samples. If the maximum hinge-loss error of a minority class sample is low, the computationally efficient WOS-ELM method is selected to tackle the class imbalance problem. While for a minority class sample with high maximum hinge-loss error, SMOTE is used to create one or more synthetic minority class samples. A high misclassification cost may then be assigned to these synthetic minority class samples. The majority class is undersampled by removing the redundant samples since there are already a lot more majority class samples as compared to minority class samples. The majority and minority class samples are checked for the following criteria and learning strategies are selected accordingly.

Majority Class Sample

Undersampling

A majority class sample whose predicted label is the same as actual label and whose max hinge-loss error is less than the delete threshold E_d does not participate in learning, i.e., majority class samples matching the following criterion are deleted

$$c = \hat{c} \text{ AND } E \leq E_d \quad (4.2)$$

Following the recommendations in McELM [104], we choose E_d from the interval $[0.1, 0.2]$. In our experiments, we selected 0.1 so that more samples can

participate in the training process. Readers can refer to [104] for the selection of E_d and other error threshold values.

This sample deletion strategy prevents learning of samples with similar knowledge and avoids overfitting. It also decreases the imbalance in dataset. Note that when a sample is deleted from the majority class, the size s_k^- stays the same.

Neuron Addition

A majority class sample whose predicted label is different from the actual class label, or whose maximum hinge-loss error is greater than the neuron addition threshold E_a , as given in (4.3), is considered to have novel information.

$$c \neq \hat{c} \text{ OR } E > E_a \quad (4.3)$$

where E_a is self-regulated according to

$$E_a = \delta E_a + (1 - \delta)E$$

In the above equation δ is the slope that controls the rate of self-adaptation and is set close to 1. The interval for the initial value of neuron addition threshold, as recommended in [104], is [1.3, 1.7]. We chose 1.3 in our experiments.

A new hidden layer neuron is added to the ELM network because of the novel information. If the number of hidden neuron changes from $L-1$ to L with the arrival of new data, i.e., the number of samples increases from $N-1$ to N , the dimension of hidden layer output matrix ' H ' changes from $(N-1) \times (L-1)$ to $N \times L$. The new output matrix is obtained using (4.4)

$$\beta_k = (\tilde{H}_k^T \tilde{W}_k \tilde{H}_k)^{-1} \tilde{H}_k^T \tilde{W}_k \tilde{Y}_k \quad (4.4)$$

The above equation is used for training weighted ELM for class imbalance learning in batch mode. The weighting matrix $\tilde{W}_k$ is given as

$$\tilde{W}_k = \text{diag}([w_{k,1}w_{k,2}\dots w_{k,Q_k}])$$

$$\text{if } x_{k,i} \in S_k^+, w_{k,i} = 1/s_k^+$$

$$\text{otherwise } w_{k,i} = 1/s_k^-$$

where Q_k is the total number of samples in the window at time step ' k '. For now, we assume that window contains all the samples up till time step ' k '.

For balanced datasets, the same weight values are assigned to minority and majority class samples. In this case, weighted ELM reduces to standard ELM as in (2.3).

Parameter Update

A majority class sample matching criterion (4.5) is used to update the network output using (4.6) and (4.7)

$$c = \hat{c} \text{ AND } E \geq E_u \quad (4.5)$$

where E_u is the update threshold and is self-regulated according to

$$E_u = \delta E_u + (1 - \delta)E$$

The range for initial value of update threshold E_u as recommended in [104] is selected within [0.4, 0.7]. We choose 0.4.

$$P_{k+1} = P_k - P_k H_{k+1}^T (W_{k+1}^{-1} + H_{k+1} P_k H_{k+1}^T)^{-1} H_{k+1} P_k \quad (4.6)$$

$$\beta^{(k+1)} = \beta^{(k)} + P_{k+1} H_{k+1}^T W_{k+1} (Y_{k+1} - H_{k+1} \beta^{(k)}) \quad (4.7)$$

The above update equation corresponds to the standard WOS-ELM method. A majority class sample takes the weight value of $1/s_{k+1}^-$, i.e., $W_{k+1} = 1/s_{k+1}^-$.

For balanced dataset, (4.7) reduces to (2.5) that corresponds to standard OS-ELM.

Finally, majority class samples which do not match criteria in (4.2), (4.3), and (4.5) are reserved for future use [104].

Minority Class Sample

Neuron Addition

Neuron addition criterion is the same as for majority class samples in (4.3) and the output matrix is also obtained using (4.4).

SMOTE Oversampling

A minority class sample that satisfies the criterion in (4.5) is considered difficult samples. Thus, SMOTE oversampling is performed by selecting ‘K’ nearest neighbors of the given minority class sample in the window. Then randomly select one of the neighbors and create a synthetic sample along the line joining the given sample and the selected neighbor. Synthetic samples are created to make decision region of the minority class more general instead of more specific.

Consider Figure 4-3 where x_i is the given minority class sample, $\hat{x}_l$ is one of its ‘K’ nearest neighbors from the same class in the current window and Δ is a random number between 0 and 1. A synthetic sample x_{new} is created as below

$$x_{new} = x_i + (\hat{x}_l - x_i) * \Delta$$

Figure 4-3: SMOTE oversampling

Once the required amount of synthetic samples is created, the output matrix is updated according to WOS-ELM equation (4.7). Where the weight value associated with the minority class is $1/s_{k+1}^+$, i.e., $W_{k+1} = 1/s_{k+1}^+$.

Whenever new synthetic samples are created, s_k^+ is updated accordingly.

Parameter Update

Finally, minority class samples which do not match criteria in (4.3) and (4.5) are used for updating network output using (4.7). A minority class sample takes the weight value of $1/s_{k+1}^+$. Since minority class samples are less in number, they are never deleted.

The hybrid learning approach can be summarized in Algorithm 4-1. In Algorithm 4-1, a minority class sample is treated differently from a majority class sample. Moreover, samples of the same class are also treated differently depending on the amount of knowledge they contain. The criteria used for a majority class sample are the same as in McELM. For balanced data stream, there is no minority class so all the samples (regardless of class labels) are checked for the majority class criteria of Algorithm 4-1. W_{k+1} is the same for minority and majority class samples. In this case, (4.7) becomes the same as parameter update equation of McELM. (4.4) becomes the

neuron addition equation of McELM. Thus, it can be deduced from above that McELM is a special case of the new method when classes are balanced. The new hybrid approach can work for highly imbalanced data streams as well. It is important to point out that McELM is proposed for stationary datasets or datasets without concept drift.

So far we have discussed one-by-one or sample-by-sample learning since McELM is originally proposed for one-by-one learning only [104]. Now we extend the new method to chunk-by-chunk case as most of the non-stationary and imbalanced learning methods are proposed for chunk-by-chunk mode [1]. For chunk-by-chunk case, meta-cognition is applied slightly differently from the one-by-one case. Assuming at each time step ' k ', a new chunk of samples is received. First, the chunk is pre-processed, i.e., some samples in the chunk are deleted and some synthetic samples are created. Then, the output matrix is calculated using samples from the updated chunk, all at once. Samples in the ' k th' chunk are matched for criteria in Algorithm 4-1 and the chunk is updated as follows. For simplicity, majority class samples matching criterion (4.2) or not matching any criteria are deleted while synthetic samples are created for each minority class sample matching criterion (4.3) or (4.5). s_k^+ and s_k^- are also updated. Then, output matrix β is calculated using the updated chunk as follows.

If any sample matches criterion (4.3), one or more hidden nodes are added. The samples in the updated chunk and all the old samples which are retained in the window at time step ' k ' are used for calculating new β using (4.4). E_a is updated with the highest E value obtained among the samples belonging to this criterion.

If no sample matches criterion (4.3), the updated chunk is used for parameter update (4.7). Weighting matrix in this case is given below

$$W_k = \text{diag}([w_{k,1} w_{k,2} \dots w_{k,N_k}])$$

$$\text{if } x_{k,i} \in S_k^+, w_{k,i} = 1/s_k^+$$

$$\text{otherwise } w_{k,i} = 1/s_k^-$$

where N_k is the total number of samples in the updated ' k th' chunk. Note that W_k is only constructed using samples in the chunk received at time step ' k ' while $\tilde{W}_k$ is constructed using all the samples in the window at time step ' k '. However, the class specific weights are calculated using s_k^+ and s_k^- in both cases since they represent class sizes. E_u is updated with the highest E value obtained among the samples belonging to this criterion.

In sequential learning from data streams the number of training samples can be in millions and using (4.5) to get output matrix with complete dataset is time consuming. In this work, neuron addition is only allowed until a maximum number of hidden neurons L_{max} is reached. After that output weight matrix is only updated sequentially using (4.7).

In non-stationary environments, meta-cognitive framework should also take care of the concept drift problem along with class imbalance. In the following subsection, drift detection and adaptive window techniques are discussed for concept drift learning.

Algorithm 4-1: Hybrid approach for class imbalance learning using meta-cognition

Given sample (x_k, c)

```

If ( $c == \text{'-'}$ )           % a majority class sample

    If (criterion (4.2) = true)
        Sample Deletion: Undersampling

    Elseif (criterion (4.3) = true)
        Neuron Addition: Obtain new output matrix ' $\beta$ ' using (4.4)

    Elseif (criterion (4.5) = true)
        Parameter Update: Update output matrix ' $\beta$ ' using (4.7)

    Else
        Reserve sample for future use

    End

Else           % a minority class sample

    If (criterion (4.3) = true)
        Neuron Addition: Obtain new output matrix ' $\beta$ ' using (4.4)

    Elseif (criterion (4.5) = true)
        SMOTE: Create synthetic samples
        Update output matrix ' $\beta$ ' using (4.7)

    Else
        Parameter Update: Update output matrix ' $\beta$ ' using (4.7)

    End

End
  
```

4.3.2 Change Detection and Adaptive Window for Concept Drift Learning

In general, there are two types of concept drifts to be detected, sudden drifts and gradual drifts. Sudden drifts are usually easier to detect as compared to gradual ones. We use the same concept drift detection mechanism as in Chapter 3 for detecting both gradual and sudden drifts. If the performance obtained by the ELM network falls below certain threshold ' τ ', drift is assumed to be detected. For sudden drifts, ' τ ' is set to 90% of the G_{mean} value from the last evaluation. Gradual drift is generally not very apparent since the performance degrades slowly. It can also be mistaken as noise. In this chapter, gradual drift is detected using statistical decision theory [98]. The implementation detail can be found in Chapter 3.

In concept drift learning, the classifier is trained with samples related to the current concept only. A window can be utilized for this type of learning which contains samples from the current concept. While samples from the old concepts, not relevant now, are removed from it. In this work, an adaptive window approach is employed for concept drift learning. The size of window in terms of the number of chunks is adjusted to adapt to the changing environment. At every time step, a new chunk is added and some old chunks may or may not be removed. If concept drift is detected by the change detector, the size of window is reduced to one, i.e., it only contains the recent chunk of samples. The size of window is then increased by one chunk at each time step until a minimum window size m_{min} is reached, i.e., once concept drift is detected and all the previous chunks are removed from the window, the window size is allowed to increase until m_{min} . If the performance improves, window size increases by one chunk at every time step. Depending on the computational resources, a maximum window size can also be set. If the performance begins to deteriorate, the size of window stays the same, i.e., a new chunk is added and the oldest chunk in the current window is removed. If performance continues to deteriorate, two oldest chunks are

removed. In case of continuous performance deterioration, the number of old chunks to be removed ' n ' increases by one at each time step. For example, assume ' G_k ' represents G_{mean} at time step ' k '. If $G_k > G_{k-1}$ (performance improves), ' n ' is zero. If $G_{k+1} < G_k$ (performance starts to deteriorate), ' n ' is 1. If $G_{k+2} < G_{k+1}$ (performance continues to deteriorate), ' n ' is 2. If $G_{k+3} > G_{k+2}$, ' n ' is again 0.

Let the window size at time step ' k ' represented by ' M ' while the window size at the current time step ' $k+1$ ' by ' m '. Suppose at time step ' $k+1$ ', a new chunk is added and ' n ' old chunks which are not relevant to the current concept are removed (see Fig. 4-4). Assuming ' n ' is known for now ($n \geq 1$), the new window size is given as

$$m = M + 1 - n$$

Figure 4-4: Adaptive window (a) window at time step ' k ' (b) window at time step ' $k+1$ '

The window size is adapted as in Algorithm 4-2

Algorithm 4-2: Adaptive window scheme

Given minimum window size: m_{min}

Old window size: $M = 1$

```

For  $k=1:T$                                      %  $T$  is the total number of chunks%
  If (change_detection_test( $\tau$ ) = 1)
     $m = 1$                                        % If change detected, window only
                                                % contains the recent most chunk%
     $n = 0$ 

  Elseif ( $M < m_{min} \mid G_{k+1} > G_k$ )
    % to allow for slight variations  $G_{k+1} > 0.99 * G_k$  is used%
     $m = M + 1$                                    % Increase window size by 1%
     $n = 0$                                        % All previous chunks are kept%

  Else
     $n = n + 1$ 
    % no. of chunks to be removed increases as the
    % performance continues to deteriorate i.e.  $G_{k+1} < G_k$ %
     $m = M+1 - n$ 
    % new chunk is added & ' $n$ ' old chunks removed%

  End
   $M = m$ 
End

```

In the next sub-section, the adaptive window approach and cost-sensitive learning are integrated into OS-ELM update equation for concept drift and class imbalance learning.

4.3.3 A Unifying Equation for Class Imbalance and Concept Drift

An OS-ELM based output update equation is proposed which can be applied to both class imbalance and concept drift learning. It is the first OS-ELM equation that simultaneously targets class imbalance and non-stationary problems. It also reduces to standard OS-ELM update equation for stationary and balanced data streams.

In non-stationary environment, the network output is updated sequentially by adding new chunk of data and removing outdated chunks using adaptive window scheme discussed in Section 4.3.2. The class imbalance problem is tackled through various CIL learning approaches discussed in Section 4.3.1.

Now we are ready to derive a unified OS-ELM equation for class imbalance and concept drift learning. Let the weighted least square solution corresponding to the initialization set, containing N_0 samples, given by

$$\beta^{(0)} = K_0^{-1} H_0^T W_0 Y_0$$

where

$$K_0 = H_0^T W_0 H_0$$

K_0 is a square matrix with dimensions $L \times L$. H_0 is the hidden layer output matrix with dimensions $N_0 \times L$ and W_0 the weighting matrix with dimensions $N_0 \times N_0$. We can follow the same approach for the subsequent time step. Now the weighted least square solution is given by

$$\beta^{(1)} = K_1^{-1} \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}^T \begin{bmatrix} W_0 & 0 \\ 0 & W_1 \end{bmatrix} \begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix}$$

$$K_1 = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}^T \begin{bmatrix} W_0 & 0 \\ 0 & W_1 \end{bmatrix} \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}$$

or
$$K_1 = K_0 + H_1^T W_1 H_1$$

H_1 and W_1 are constructed with chunk 1, containing N_1 samples. The dimensions of H_1 and W_1 are $N_1 \times L$ and $N_1 \times N_1$ respectively.

It can be generalized to ' k th' chunk with respect to Figure 4-4. As can be observed from figure, ' $k-M+1$ ' or ' $k-m-n+2$ ' represents the initialization set or the oldest chunk in the window. If at time step ' k ', all the previous chunks (starting from time step ' 0 ' to time step ' k ') are retained in the window then ' $k-M+1$ ' or ' $k-m-n+2$ ' represents time step ' 0 '. For a more general case when the oldest chunk in the window may or may not belong to time step ' 0 ', $\beta^{(k)}$ is given by

$$\beta^{(k)} = K_k^{-1} \begin{bmatrix} H_{k-m-n+2} \\ \vdots \\ H_k \end{bmatrix}^T \begin{bmatrix} W_{k-m-n+2} & 0 & \dots \\ 0 & \ddots & 0 \\ \vdots & 0 & W_k \end{bmatrix} \begin{bmatrix} Y_{k-m-n+2} \\ \vdots \\ Y_k \end{bmatrix}$$

$$K_k = K_{k-1} + H_k^T W_k H_k \quad (4.8)$$

With the arrival of ' $(k+1)$ th' chunk, the weighted least square solution corresponding to the current window ' m ' (see Fig. 4-4(b)) containing chunks ' $k-m+2$ ' to ' $k+1$ ' is given as follows

$$\beta^{(k+1)} = K_{k+1}^{-1} \begin{bmatrix} H_{k-m+2} \\ \vdots \\ H_{k+1} \end{bmatrix}^T \begin{bmatrix} W_{k-m+2} & 0 & \dots \\ 0 & \ddots & 0 \\ \vdots & 0 & W_{k+1} \end{bmatrix} \begin{bmatrix} Y_{k-m+2} \\ \vdots \\ Y_{k+1} \end{bmatrix} \quad (4.9)$$

K_{k+1} is obtained as shown in (4.10). The first two terms on the RHS of equation (4.10) follow from (4.8) while the third term corresponds to chunks from the old concept. Note that K_k accumulates all the data upto chunk ' k ' but ' n ' oldest chunks

(belonging to old concept) from the previous window need to be removed.

$$K_{k+1} = K_k + H_{k+1}^T W_{k+1} H_{k+1} - \begin{bmatrix} H_{k-m-n+2} \\ \vdots \\ H_{k-m+1} \end{bmatrix}^T \begin{bmatrix} W_{k-m-n+2} & 0 & \dots \\ 0 & \ddots & 0 \\ \vdots & 0 & W_{k-m+1} \end{bmatrix} \begin{bmatrix} H_{k-m-n+2} \\ \vdots \\ H_{k-m+1} \end{bmatrix} \quad (4.10)$$

$$= K_k + \begin{bmatrix} -H_{k-m-n+2} \\ \vdots \\ -H_{k-m+1} \\ H_{k+1} \end{bmatrix}^T \begin{bmatrix} W_{k-m-n+2} & 0 & \dots & 0 \\ 0 & \ddots & 0 & \vdots \\ \vdots & 0 & W_{k-m+1} & 0 \\ 0 & \dots & 0 & W_{k+1} \end{bmatrix} \begin{bmatrix} H_{k-m-n+2} \\ \vdots \\ H_{k-m+1} \\ H_{k+1} \end{bmatrix} \quad (4.11)$$

$$= K_k + \Phi_{k+1}^T \Omega_{k+1} \theta_{k+1}$$

$$\text{or } K_k = K_{k+1} - \Phi_{k+1}^T \Omega_{k+1} \theta_{k+1} \quad (4.12)$$

where

$$\theta_{k+1} = \begin{bmatrix} H_{k-m-n+2} \\ \vdots \\ H_{k-m+1} \\ H_{k+1} \end{bmatrix}, \quad \Phi_{k+1} = \begin{bmatrix} -H_{k-m-n+2} \\ \vdots \\ -H_{k-m+1} \\ H_{k+1} \end{bmatrix} \quad \text{and}$$

$$\Omega_{k+1} = \begin{bmatrix} W_{k-m-n+2} & 0 & \dots & 0 \\ 0 & \ddots & 0 & \vdots \\ \vdots & 0 & W_{k-m+1} & 0 \\ 0 & \dots & 0 & W_{k+1} \end{bmatrix}$$

Consider (4.9) again

$$\begin{aligned} \beta^{(k+1)} &= K_{k+1}^{-1} \left\{ \begin{bmatrix} H_{k-m-n+2} \\ \vdots \\ H_k \end{bmatrix}^T \begin{bmatrix} W_{k-m-n+2} & 0 & \dots \\ 0 & \ddots & 0 \\ \vdots & 0 & W_k \end{bmatrix} \begin{bmatrix} Y_{k-m-n+2} \\ \vdots \\ Y_k \end{bmatrix} + \Phi_{k+1}^T \Omega_{k+1} \Psi_{k+1} \right\} \\ &= K_{k+1}^{-1} (K_k \beta^{(k)} + \Phi_{k+1}^T \Omega_{k+1} \Psi_{k+1}) \end{aligned} \quad (4.13)$$

where

$$\Psi_{k+1} = \begin{bmatrix} Y_{k-m-n+2} \\ \vdots \\ Y_{k-m+1} \\ Y_{k+1} \end{bmatrix}$$

using (4.12)

$$\begin{aligned} \beta^{(k+1)} &= K_{k+1}^{-1} (\{ K_{k+1} - \Phi_{k+1}^T \Omega_{k+1} \theta_{k+1} \} \beta^{(k)} + \Phi_{k+1}^T \Omega_{k+1} \Psi_{k+1}) \\ &= K_{k+1}^{-1} (K_{k+1} \beta^{(k)} - \Phi_{k+1}^T \Omega_{k+1} \theta_{k+1} \beta^{(k)} + \Phi_{k+1}^T \Omega_{k+1} \Psi_{k+1}) \quad (4.14) \\ &= \beta^{(k)} + K_{k+1}^{-1} \Phi_{k+1}^T \Omega_{k+1} (\Psi_{k+1} - \theta_{k+1} \beta^{(k)}) \end{aligned}$$

Let $P_{k+1} = K_{k+1}^{-1}$

Here we use the following matrix inversion lemma to calculate P_{k+1}

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + V^{-1}AU)^{-1}VA^{-1}$$

where A is K_k , U is Φ_{k+1}^T , C is Ω_{k+1} and V is θ_{k+1} . Thus,

$$\begin{aligned} P_{k+1} &= P_k - P_k \Phi_{k+1}^T (\Omega_{k+1}^{-1} + \theta_{k+1} P_k \Phi_{k+1}^T)^{-1} \theta_{k+1} P_k \\ \beta^{(k+1)} &= \beta^{(k)} + P_{k+1} \Phi_{k+1}^T \Omega_{k+1} \{\Psi_{k+1} - \theta_{k+1} \beta^{(k)}\} \end{aligned}$$

In the above equation, θ_{k+1} , Φ_{k+1} , Ω_{k+1} and Ψ_{k+1} are substituted with the original matrices to get (4.15). The single closed form equation in (4.15) is a unifying equation for class imbalance and concept drift learning. In this equation, ‘ m ’ is the size of current window and ‘ n ’ is the number of old chunks to be removed from the window. As can be observed from (4.15), the most recent chunk to be removed from window is ‘ $k-m+1$ ’ (represented by H_{k-m+1} with a negative sign). When ‘ n ’ is 1, the oldest chunk to be removed from the window is also ‘ $k-m+1$ ’ so only one chunk is removed. When ‘ n ’ is 2, the oldest chunk to be removed is ‘ $k-m$ ’ which is true since two chunks are removed. When ‘ n ’ is 3, the oldest chunk to be removed is ‘ $k-m-1$ ’ which is also true since three chunks (‘ $k-m-1$ ’, ‘ $k-m$ ’ and ‘ $k-m+1$ ’) are removed now. This continues for all positive integer values of ‘ n ’. When ‘ n ’ is 0, the oldest chunk to

be removed is ‘ $k-m+2$ ’ which is a contradiction since it is more recent than ‘ $k-m+1$ ’.

Hence, for $n = 0$ no chunk is removed and $\theta_{k+1}, \Phi_{k+1}, \Omega_{k+1}, \Psi_{k+1}$ simply become $H_{k+1}, H_{k+1}, W_{k+1}, Y_{k+1}$ respectively.

$$\beta^{(k+1)} = \beta^{(k)} + P_{k+1} \left[\begin{matrix} -H_{k-m-n+2} \\ \vdots \\ -H_{k-m+1} \\ H_{k+1} \end{matrix} \right]^T \left[\begin{matrix} W_{k-m-n+2} & 0 & \dots & 0 \\ 0 & \ddots & 0 & \vdots \\ \vdots & 0 & W_{k-m+1} & 0 \\ 0 & \dots & 0 & W_{k+1} \end{matrix} \right] \left\{ \left[\begin{matrix} Y_{k-m-n+2} \\ \vdots \\ Y_{k-m+1} \\ Y_{k+1} \end{matrix} \right] - \left[\begin{matrix} H_{k-m-n+2} \\ \vdots \\ H_{k-m+1} \\ H_{k+1} \end{matrix} \right] \beta^{(k)} \right\} \quad (4.15)$$

Note that (4.15) is based on the mean-squared error since OS-ELM, WOS-ELM and FOS-ELM uses mean-squared error. For hinge-loss error, the factor $\left\{ \left[\begin{matrix} Y_{k-m-n+2} \\ \vdots \\ Y_{k-m+1} \\ Y_{k+1} \end{matrix} \right] - \left[\begin{matrix} H_{k-m-n+2} \\ \vdots \\ H_{k-m+1} \\ H_{k+1} \end{matrix} \right] \beta^{(k)} \right\}$ in (4.15) is replaced with ‘ e ’ equation (4.1) as pointed out in [104]. The update equation (4.15) can simultaneously target class imbalance and concept drift as shown in Figure 4-5.

Figure 4-5: Single OS-ELM update equation for class imbalance and concept drift

It can easily be shown that (4.15) reduces to update equations corresponding to OS-ELM (also used in McELM), WOS-ELM or FOS-ELM methods under different conditions. Consider the following cases

Case 1: Relatively Balanced classes with stationary distributions

In this case, all the values in weighting matrix are the same which is equivalent to multiplying a scalar with identity matrix. Moreover, ' n ' is zero assuming performance criteria ' G_k ' in Algorithm 4-2 does not deteriorate. Under this condition, (4.15) becomes standard update equation for OS-ELM (4.15). McELM also uses (4.15) for parameter update.

Case 2: Imbalanced classes with stationary distributions

In this case, the samples from different classes takes different weight values so weighting matrix will be diagonal and ' n ' is still zero. In this case, (4.15) becomes update equation for WOS-ELM (4.7). Note that undersampling and SMOTE oversampling do not change the update equation since they are pre-processing methods.

Case 3: Balanced classes with non-stationary distributions

In this case, all the values in weighting matrix are the same which is equivalent of multiplying a scalar with identity matrix but ' n ' is not zero. If ' n ' is a fixed number, i.e., if we know *a priori* the optimal window size then (4.15) becomes update equation for FOS-ELM. However, with variable ' n ', (4.15) becomes a more intelligent version of FOS-ELM since ' n ' changes such that the window contains chunks only from the current concept.

Case 4: Imbalanced classes and non-stationary distributions

This is the case where (4.15) does not reduce to any previous method since there is no single OS-ELM based update equation that can tackle the combined problem. (4.15) is the first OS-ELM equation that can be applied to class imbalance learning from drifting data stream.

In non-stationary environments, the number of hidden neurons is allowed to change when the class concepts change. In case of detected concept drift, the algorithm allows neuron addition using (4.4), same as at the start of the learning (discussed in Section 4.3.1). The batch learning is performed only with samples in the current

window ‘ m ’. The window size is generally not very big after the drift is detected so the computational complexity doesn’t increase much. Once L_{max} is reached, the output matrix is updated sequentially using (4.15).

4.4 Experiments

This section is divided into 3 parts such that in each part MOS-ELM is compared with methods specific to a particular application. In the first part, MOS-ELM is compared with WOS-ELM and OS-ELM-SMOTE methods for imbalanced datasets. In the second part, it is compared with recently proposed FOS-ELM [71] and TOSELM [72] methods on non-stationary datasets. Then, the proposed method is compared with ESOS-ELM and Learn++.NIE methods for imbalanced datasets in non-stationary environments. The experiments are carried out in a PC with 2.66 GHz CPU and 4 GB RAM. In all the experiments, sigmoid is used as the activation function. As in the previous chapters, G_{mean} is used as the evaluation measure.

4.4.1 MOS-ELM Performance Comparison on Imbalanced Datasets

The performance of MOS-ELM is first evaluated on the same 15 real-world imbalanced datasets that were previously used for performance evaluations of WOS-ELM and OS-ELM-SMOTE. 14 datasets are taken from UCI repository [92] and the oil dataset is provided by Dr. Holte [93]. The details of these datasets can be found in Table 2-3. MOS-ELM uses meta-cognition to introduce self-regularization in CIL. It employs WOS-ELM, SMOTE and undersampling depending on the information contained in the sample. It is interesting to compare MOS-ELM with WOS-ELM and SMOTE methods which do not have any self-regulating mechanism. All the experimental settings, i.e., training set, testing set, validation set and chunk sizes are set

identical to those of above CIL methods. Chapter 2 can be referred for the number of hidden neurons and other experimental details. The L_{max} value (max. no. of hidden neurons) is set to the optimal number of hidden neurons found in Chapter 2. If needed, five neurons are added with every chunk. Table 4-1 shows results obtained by WOS-ELM, OS-ELM-SMOTE and the new method.

Table 4-1 G_{mean} obtained on imbalanced datasets

Dataset	Chunk Size	OS-ELM-SMOTE	WOS-ELM	MOS-ELM
page-block5	100	91.5	91.7	92.6
abalone15	100	69.8	70.9	72.2
yeast5	75	79.2	80.4	82.2
car3	50	93.4	95.1	96.3
oil	50	80.6	79.9	83.6
abalone9-18	50	77.9	84.2	84.6
balance2	30	63.5	67.6	68.5
satimage4	50	85.6	85.7	85.9
mf-morph10	50	92.1	92.3	92.8
mf-zernike10	50	88.4	87.8	88.9
cmc2	50	65.6	66.1	65.8
transfusion	20	66.9	67.9	69.1
haberman	20	63.9	63.7	65.2
phoneme	50	83.8	84.1	84.1
pima	50	76.5	75.6	77.8
Avg.		78.5	79.5	80.6

Friedman test [94] is used to confirm whether improvements achieved in Table 4-1 are statistically significant than the other online sequential algorithms in comparison. A post-hoc analysis is also applied since more than two CIL methods are compared.

For each dataset, the best performing algorithm is ranked as 1, the second best as 2, and so on. If classifiers produce same results, an average rank is assigned to each of them. Matlab function '*friedman*' is used for hypothesis testing. A null hypothesis that all the classifiers are equivalent and any differences among their average ranks are random is rejected, if the obtained p -value is lower than a specific confidence level α . The p -value obtained for all G_{mean} values is 2.77×10^{-5} thus a null hypothesis is rejected at $\alpha = 0.05$. In order to analyze differences among MOS-ELM, WOS-ELM and OS-ELM-SMOTE, a post-hoc Nemenyi is adopted. The average ranks of MOS-ELM, WOS-ELM and OS-ELM-SMOTE are 1.1, 2.167 and 2.67 respectively as shown in Table 4-2. A critical difference (CD) of 0.855 is obtained at $p = 0.05$ [94]. The rank difference between MOS-ELM and WOS-ELM is greater than CD so they are statistically different while the rank difference between WOS-ELM and OS-ELM-SMOTE is smaller than CD so they are statistically similar. Fig. 4-6 visualizes the Nemenyi post-hoc test at $p = 0.05$. The mean ranks over 15 datasets are shown in descending rank order on the x -axis. Statistically, similar methods, WOS-ELM and OS-ELM-SMOTE in this case are grouped with horizontal bar [94].

Table 4-2: Rank comparison among OS-ELM-SMOTE, WOS-ELM and MOS-ELM

Dataset	OS-ELM-SMOTE	WOS-ELM	MOS-ELM
page-block5	3	2	1
abalone15	3	2	1
yeast5	3	2	1
car3	3	2	1
oil	2	3	1
abalone9-18	3	2	1
balance2	3	2	1
satimage4	3	2	1
mf-morph10	3	2	1
mf-zernike10	2	3	1
cmc2	3	1	2
transfusion	3	2	1
haberman	2	3	1
phoneme	2	1.5	1.5
pima	2	3	1
Avg. rank	2.666667	2.166667	1.1

Figure 4-6: Visualization of Nemenyi post-hoc test for chunk-by-chunk learning. The mean rank for each method over all the datasets is plotted on x -axis. Connected method are not significantly different at $p=0.05$

4.4.2 MOS-ELM Performance Comparison on Non-Stationary Datasets

In this sub-section, the performance of MOS-ELM is compared with two recently proposed OS-ELM based learners for non-stationary environments. The first classifier is FOS-ELM which removes the old samples after a fixed period of time. In concept drifting datasets, fixed period may not be known *a priori*. We tried validity period of 5, 10 and 15 chunks and reported the one with best result. The minimum window size in MOS-ELM is set to 5. The second method in comparison is TOSELM. FOS-ELM and TOSELM are two OS-ELM methods very closely related to MOS-ELM. In fact, FOS-ELM is a special case of MOS-ELM when classes are balanced and window size is fixed. Note that MOS-ELM, TOSELM and FOS-ELM are all implemented as single classifiers for a fair comparison.

The STAGGER Concepts:

The STAGGER concepts [99] is one of the widely used dataset for performance evaluation of classifiers in the presence of concept drift. The details of the dataset can be found in Section 3.3.1. This is an example of sudden concept drift and class 1 is the target class. 1200 instances are used for training the classifiers and 1200 instances are randomly generated for testing.

The SEA Concepts:

The synthetic problem proposed by Street and Kim [85] has become a benchmark in the concept drift problem. The details of the dataset can be found in Section 3.3.1. This is also an example of sudden concept drift and the two classes are balanced. In total, there are 100,000 samples for training and 100,000 for testing.

Rotating Spiral Dataset:

Rotating spiral is a recently introduced synthetic dataset [3] [100] consisting of four rotating spirals, two for each class. The spirals rotate over 300,000 instances in

one complete rotation. Entire feature space is considered in the experiments. Rotating spirals change the decision boundary continuously thus simulating gradual drift.

Electricity Pricing Domain (elect2):

The ‘elect2’ was first used by Harris [101] and has now become a real-world benchmark dataset for concept drift learning. The details of the dataset can be found in Section 3.3.1. At every time step, identical chunk of data is presented to all the methods in comparison. The initial number of hidden neurons is obtained using cross-validation on the initialization set. Subsequently, if criterion (4.3) is met, 5 neurons are added with each time step and L_{max} value is set to the optimal number of hidden neurons in FOS-ELM or TOSELM (whichever is smaller). The minimum window size in MOS-ELM is set to 5 for most of the datasets. For SEA and Rotating Spiral (large datasets) a maximum window size of 12 is also set. The results are reported in Table 4-3.

Table 4-3 Classification results comparison for concept drifting datasets (balanced datasets)

Dataset	Algorithm	G_{mean}
STAGGER	TOSELM	90.4
	FOS-ELM	80.4
	MOS-ELM	95.9
SEA	TOSELM	73.9
	FOS-ELM	80.8
	MOS-ELM	83.9
Spiral	TOSELM	60.4
	FOS-ELM	79.1
	MOS-ELM	79.8
Elect2	TOSELM	59.1
	FOS-ELM	64
	MOS-ELM	65.7

(a)

(b)

(c)

(d)

Figure 4-7: Classification results comparison for concept drifting datasets (balanced datasets) (a) STAGGER (b) SEA (c) Spiral (d) Elect2

As can be seen from Table 4-3 and Figure 4-7, MOS-ELM clearly outperformed TOSELM on all the four datasets and also obtained better results than that of FOS-ELM.

4.4.3 MOS-ELM Performance Comparison on Non-Stationary Datasets with Imbalanced Classes

Most state-of-the art methods for class imbalance learning in the presence of concepts drift exist as ensemble methods. We compare MOS-ELM performance with the results obtained by two recently proposed ensemble methods namely Learn⁺⁺.NIE [3] and ESOS-ELM. We used the same imbalanced data streams with drifting concepts as in Chapter 3.

The results shown in Table 4-4 and Figure 4-8 for STAGG, SEA and Spiral consider the first half only as the comparison is not for recurring environments. For recurring environments, MOS-ELM can use the same ELM-Store technique which is used in ESOS-ELM. MOS-ELM method obtained similar or better performance than that of Learn⁺⁺.NIE and ESOS-ELM methods which are considered state-of-the-art ensemble learning methods on binary class data streams.

Table 4-4 Classification results comparison for concept drifting datasets (imbalanced datasets)

Dataset	Algorithm	G_{mean}
STAGGER	Learn ⁺⁺ .NIE	97.2
	EOS-ELM-CIL	97
	MOS-ELM	98.4
SEA	Learn ⁺⁺ .NIE	81.9
	EOS-ELM-CIL	83.8
	MOS-ELM	84
Spiral	Learn ⁺⁺ .NIE	77.9
	EOS-ELM-CIL	80.3
	MOS-ELM	79.8
Elect2	Learn ⁺⁺ .NIE	58.4
	EOS-ELM-CIL	62.3
	MOS-ELM	63.6

(a)

(b)

(c)

(d)

Figure 4-8: Classification results comparison for concept drifting datasets (Imbalanced datasets) (a) STAGGER (b) SEA (c) Spiral (d) Elect2

4.5 Discussion

In this chapter, a meta-cognitive online sequential extreme learning machine (MOS-ELM) is proposed for class imbalance and concept drift. MOS-ELM uses meta-cognition to self-regulate the learning, i.e., selects the best learning strategy depending on the nature of data stream. A hybrid technique comprising oversampling, undersampling and cost-sensitive weighting for tackling class imbalance and change detector and adaptive window technique for tackling concept drift are embedded in the meta-cognitive framework. A single OS-ELM output update equation that can be applied to both class imbalance and concept drift learning is proposed. MOS-ELM obtained better results than WOS-ELM for imbalanced datasets, TOSELM and FOS-ELM for non-stationary data streams and ESOS-ELM and Learn⁺⁺.NIE for data streams with both imbalance and concept drift.

It is appropriate at this point to compare the results obtained by three binary CIL methods proposed in Chapter 2, 3 and 4 on the benchmark imbalanced datasets without concept drift from Table 2-3 for chunk-by-chunk learning. 50 trials of 10-fold cross validation are conducted and G_{mean} is used as the evaluation measure. The performances are reported in terms of the average of G_{mean} (Mean) over 50 trials for each dataset, the standard deviation (Dev) and the smallest value (Min) with respect to G_{mean} . The average Mean, Dev and Min values over all the 15 datasets are also reported. It can be observed from the table that ESOS-ELM achieves the best overall performance. The performance of MOS-ELM is similar to that of ESOS-ELM. It is important to point out that MOS-ELM achieves this with a single classifier while ESOS-ELM with an ensemble of 10 or more classifiers. As discussed earlier, WOS-ELM cannot work for concept drift learning while MOS-ELM produces better result than ESOS-ELM for the combined problem of class imbalance and concept drift.

Table 4-5 Classification results in terms of Mean, Dev and Min values using G_{mean}

		WOS-ELM			ESOS-ELM			MOS-ELM		
Dataset	Chunk Size	Dev	Min	Mean	Dev	Min	Mean	Dev	Min	Mean
page-block5	100	3	82.8	91.7	2.8	84.2	92.4	3.1	83.8	92.6
abalone15	100	4.4	61.7	70.9	2.6	66.8	73.6	3.2	64.6	72.2
yeast5	75	2.7	74.8	80.4	2.5	76.3	82.2	2.5	77.7	82.2
car3	50	2.8	91.7	95.1	1.4	93.3	96.1	2.2	93.7	96.3
oil	50	5.2	68.5	79.9	4.7	72.8	83.2	4.4	75.4	83.6
abalone9-18	50	6.6	64.7	84.2	6.5	68.5	86.4	6.2	67.9	84.6
balance2	30	6.8	49.2	67.6	4.4	59.1	69.2	6.3	56.2	68.5
satimage4	50	2.3	82.6	85.7	1.1	83.9	85.9	1.2	83.4	85.9
mf-morph10	50	1.5	87.9	92.3	1.9	89.2	92.5	1.9	89.4	92.8
mf-zernike10	50	2.8	80.5	87.8	2.4	84.4	89	2.6	82.8	88.9
cmc2	50	2.9	55.6	66.1	1.5	62.3	66.1	2.5	59.7	65.8
transfusion	20	3	60.3	67.9	2.8	63.7	68.5	2.8	65.1	69.1
haberman	20	6.7	52.8	63.7	5.7	56.5	66	6.1	55.7	65.2
phoneme	50	0.6	82.8	84.1	0.5	83.4	84.3	0.6	82.9	84.1
pima	50	3.8	63.7	75.6	2.4	69.1	77.5	2.8	66.8	77.8
Avg.		3.6	70.6	79.5	2.8	74.2	80.8	3.2	73.6	80.6

Chapter 5

IMBALANCE PROBLEM IN MULTICLASS SEQUENTIAL LEARNING

5.1 Introduction

5.1.1 Multiclass Imbalance Learning with Stationary Datasets

In the past decade, numerous methods have been proposed to tackle the class imbalance problem in binary class datasets. Some efforts have been made to extend these methods to multiclass classification problem, i.e., when there is more than one minority or majority classes. Class imbalance learning methods proposed for binary class datasets are either less effective or sometimes perform negatively for multiclass case [106]. Recently, some systematic studies of class imbalance learning have been performed to specifically target the multiclass imbalance problem [30] [73]. In multiclass datasets, defining the imbalance ratio and identification of minority and majority classes are not as straight forward as for the two class datasets. For example, there can be more than one minority classes in the dataset and one minority class is

more difficult to be separated with the majority class as compared to others. Decomposition methods such as one-vs-all (OVA) and one-vs-one (OVO) which divide multiclass classification problem into several binary class problems are also applied to CIL. In this way, multiclass imbalance problem is decomposed into several binary class imbalance problems. The decomposition methods address the multiclass part while data sampling methods such as undersampling and SMOTE oversampling address the class imbalance part [76]. Similar approaches have been proposed by Liao [77] and Chen [74] which makes use of decomposition methods in combination with some binary class imbalance learning methods. Even though the decomposition methods proposed for multiclass imbalance problem simplify the problem, the ambiguity still remains since binary classifiers are trained without knowledge of full dataset [75].

Recently, weighted extreme learning machine (WELM) [46] is proposed as a multiclass method to tackle the class imbalance problem in batch learning. The output layer in WELM has multiple nodes whose number depends on the number of classes. It implements a special type of cost-sensitive learning in which the cost is automatically generated from the class information. The class specific weights are set inversely proportional to the respective class sizes such that after weighting the classes are believed to be re-balanced.

When it comes to sequential learning, multiclass imbalance problem is considered as an uncharted territory. Very recently McELM [104] method is proposed for multiclass sequential learning and shows some potential for imbalance learning. However, McELM has not been evaluated on highly imbalanced multiclass datasets.

5.1.2 Multiclass Imbalance Learning with Concept Drift

Most of the class imbalance learning methods in non-stationary environments are proposed for binary class datasets, i.e., datasets with one minority class and one majority class. For example the method proposed in [61] uses a bagging framework which selects all the minority class samples and a subset of the majority class samples to train the base classifiers. Thus, inherently it is for binary class problems. The selectively recursive approach (SERA) [62], proposed by Chen and He, is an improvement over the uncorrelated bagging method [61] since it selects only the relevant minority class samples instead of all the previous minority class samples. SERA also exists for a single majority and single minority class dataset. Learn⁺⁺ for Concept Drift with SMOTE (Learn⁺⁺.CDS) [3] uses SMOTE to create synthetic minority class samples in order to reduce the imbalance and then apply Learn⁺⁺.NSE method to tackle the concept drift. Learn⁺⁺.CDS is primarily evaluated for binary class datasets. Learn⁺⁺ for Nonstationary and Imbalanced Environments (Learn⁺⁺.NIE) [3] optimizes the classifier's performance on both minority and majority class data. Instead of oversampling of minority class using SMOTE, this method generates a sub-ensemble of classifiers, each trained with all the minority class samples and a randomly sampled subset of majority class samples. Again Learn⁺⁺.NIE is also evaluated on binary class datasets since the main idea of creating the sub-ensemble is specifically for two classes. Ensemble of subset online sequential extreme learning machine (ESOS-ELM) is proposed for class imbalance and concept drift in Chapter 3. In ESOS-ELM, each minority class sample is processed by I classifiers (I is the imbalance ratio) while the majority class samples are processed in a round robin fashion, i.e., first majority class sample is processed by the first classifier, the second sample by the second classifier and so on. Classifiers in ESOS-ELM are trained with approximately equal number of majority and minority class samples.

Instead of training every OS-ELM network with all the samples, they are trained with balanced subsets of the data stream. ESOS-ELM assumes that there is only one minority and one majority class, i.e., it is also proposed primarily for binary class datasets, thus, it may not be suitable for multiclass problem.

All the methods discussed above are considered state-of-the-art when it comes to tackling the class imbalance problem in non-stationary environments. To the best of our knowledge, the combined problem of class imbalance and concept drift for multiple minority or majority classes has not been discussed in the literature. Multiclass imbalance problem in the context of sequential learning is relatively a very new field and when it comes to drifting data streams, hardly any research has been done.

In this chapter, we will first extend the MOS-ELM method proposed in Chapter 4 for multiclass imbalance problem and compare its results with that of McELM for stationary datasets. Later, we will implement MOS-ELM as the first method to tackle the combined problem of class imbalance and concept drift in multiclass data streams.

5.2 Proposed Method

In multiclass datasets, the classes have varying degrees of imbalance among them. For the sake of simplicity, if the number of samples in a class is greater than or equal to the average number of samples per class, it is considered as a majority class. Otherwise, it is considered as a minority class. Let S_k^j represents class j at time step ' k ' and s_k^j is the total number of samples in class j up till time step ' k ', where $j=1, \dots, q$.

This segregation of minority and majority classes is useful in selecting the appropriate CIL strategy in MOS-ELM. SMOTE oversampling may be performed for samples from any of the minority classes while samples from any of the majority

classes may be undersampled. McELM becomes a special case of MOS-ELM when classes are balanced with no concept drift. The MOS-ELM learning strategies for multiclass case are used in a similar way as for the two class problem but with multiple minority and majority classes.

Majority Class Sample

Undersampling

A majority class sample whose predicted label is the same as actual class label and max hinge-loss error is less than the delete threshold E_d does not participate in learning, i.e., a majority class sample matching the criterion in (5.1), is deleted. Note that when a sample is deleted from class j , s_k^j stays the same.

$$c = \hat{c} \text{ AND } E \leq E_d \quad (5.1)$$

Neuron Addition

A majority class sample whose predicted label is different from the actual class label, or max hinge-loss error is greater than the neuron addition threshold E_a , as given in (5.2), is considered to have novel information.

$$c \neq \hat{c} \text{ OR } E > E_a \quad (5.2)$$

A new hidden layer neuron is added to the ELM network because of this novel information using (5.3)

$$\beta_k = (\tilde{H}_k^T \tilde{W}_k \tilde{H}_k)^{-1} \tilde{H}_k^T \tilde{W}_k \tilde{Y}_k \quad (5.3)$$

The weighting matrix $\tilde{W}_k$ is given as

$$\tilde{W}_k = \text{diag}([w_{k,1} w_{k,2} \dots w_{k,Q_k}])$$

$$\text{if } x_{k,i} \in S_k^j, w_{k,i} = 1/s_k^j$$

where Q_k is the total number of samples in the window at time step ' k '.

Parameter Update

A majority class sample matching criterion (5.4) is used to update the network output using WOS-ELM update equations in (4.6) and (4.7)

$$c = \hat{c} \text{ AND } E \geq E_u \quad (5.4)$$

A sample from class j takes the weight value of $1/s_{k+1}^j$, i.e., $W_{k+1} = 1/s_{k+1}^j$, where $j=1, \dots, q$.

Finally, majority class samples which do not match criteria in (5.1), (5.2), and (5.4) are reserved for future use.

Minority Class Sample

Neuron Addition

Neuron addition criterion is the same as for majority class sample in (5.2) and the output matrix is also obtained using (5.3).

SMOTE Oversampling

A minority class sample that satisfies the criterion in (5.4) is considered difficult sample. Thus, SMOTE oversampling is performed by selecting ' K ' nearest neighbors of the given minority class in the current window.

Once the required amount of synthetic samples is created, the output matrix is updated according to (4.7). Where the weight value associated with sample from class j is $1/s_{k+1}^j$, i.e., $W_{k+1} = 1/s_{k+1}^j$ where $j=1, \dots, q$.

Whenever new synthetic samples are created for class j , s_k^j is updated accordingly.

Parameter Update

Finally, a minority class sample which does not match criteria in (5.2) and (5.4) is used for updating network output using (4.7). A sample from class j takes weight value of $1/s_{k+1}^j$. Since minority class samples are less in number, they are never deleted.

Note that the error threshold values in (5.1), (5.2) and (5.4) are the same as for binary class case.

Non-Stationary Case

For non-stationary learning, the class imbalance is alleviated using separate learning strategies for minority and majority classes, same as discussed above. The adaptive window along with drift detection mechanism takes care of the concept drift problem. An extended version of G_{mean} suitable for multiclass case is used as the evaluation measure in the drift detection and adaptive window schemes. The extended version also measures the per class accuracies in a balanced way. For multiclass datasets, it is given as follows

$$G_{mean} = \left(\prod_{j=1}^q \left(\frac{s^{jj}}{s^j} \right) \right)^{\frac{1}{q}}$$

where q is the number of classes, s^j is the number of samples in class j , and s^{jj} is the number of correctly classified samples in class j . For a high G_{mean} value, all accuracies should be high.

5.3 Experiments

The performance of MOS-ELM is first evaluated on 8 real-world imbalanced datasets with more than two classes. Later, a recently introduced synthetic moving Gaussian dataset is used for performance evaluation in concept drifting environment. Two different versions of moving Gaussian dataset are used to simulate both single and multiple minority classes.

The eight stationary datasets are taken from UCI repository [92]. The details of datasets are shown in Table 5-1 and it can be observed that different number of classes, features and class sizes are considered. The performance of MOS-ELM is compared with McELM which also uses meta-cognition to introduce self-regularization in the training and has the ability to solve multiclass problems. McELM has not been evaluated on highly imbalanced datasets. 50 runs of n-fold cross validation is performed where ‘n’ is the same for both the methods in comparison. In Table 5-2, G_{mean} values obtained by both MOS-ELM and McELM on 8 imbalanced datasets are reported.

Table 5-3 shows the number of hidden neurons required for both the methods.

Table 5-1 Details of multiclass imbalanced datasets

Dataset	# of Attributes	#of Classes	Class Distribution	size
Balance	4	3	49: 288: 288	625
Contraceptive	9	3	629: 333: 511	1473
Glass	9	6	70:76:17:13:9:29	214
Satimage	36	6	1533: 703: 1358: 626: 707: 1508	6435
Dermatology	34	6	112: 61: 72: 49: 52: 20	366
Ecoli	6	5	143: 77: 35: 20: 52	327
Yeast	8	9	463:429:244:163:51:44:35:30:20	1479
Page-Block	10	5	4913: 329: 28: 88: 115	5473

Table 5-2 G_{mean} values obtained by MOS-ELM and McELM

Dataset	MOS-ELM	McELM
Balance	91.1	68
Contraceptive	56.4	54.1
Glass	84.6	79.5
Satimage	87.2	84.3
Dermatology	82.2	76.6
Ecoli	90.3	90.3
Yeast	52.4	0
Page-Block	92.3	72.6

Table 5-3 Number of hidden neurons in MOS-ELM and McELM

Dataset	MOS-ELM	McELM
Balance	52	56
Contraceptive	115	367
Glass	36	58
Satimage	190	309
Dermatology	46	77
Ecoli	40	33
Yeast	80	251
Page-Block	45	114

As can be observed from Table 5-2, McELM performance is not reliable for highly imbalanced datasets. For example, McELM compromises one of the minority classes in Yeast dataset thus resulting in zero G_{mean} .

Now, the performance of MOS-ELM is evaluated on two multiclass imbalanced data streams with drifting concepts. The explanation of the two datasets is given below

Drifting Gaussians dataset with one minority class (Guassian1)

Recently, a synthetic dataset called drifting gaussian dataset has been introduced for concept drift learning [3]. It consists of 4 classes representing 4 different

combinations of Gaussian components. All the classes are gradually drifting over time 't' according to the parametric equations shown in Table 5-4. Each class is a combination of two Gaussians whose variances and means are given in the table. In the original dataset, class 1, 3 and 4 were combined to form the majority class and class 2 was the minority class. In this study, we are interested in multiclass imbalance problem so we haven't combined the classes and at each time step, 50 samples for class 1, 3 and 4 are obtained while 5 samples for class 2, which is the only minority class, are obtained. Samples are received over a period of 100 time steps. Readers can refer to [3] for detailed explanation on how to generate this dataset. This imbalanced datasets has only been used previously in binary classification problem for the performance evaluation of Learn⁺⁺.NIE and Learn⁺⁺.CDS in [3].

Drifting Gaussians dataset with two minority classes(Gaussian2)

In this datasets, 50 samples are received for class 1 and class 4 at each time step. While 5 samples are received for class 2 and class 3. This is a more challenging version of the first dataset since now there are 2 minority classes. The samples are generated according to the Gaussian components in Table 5-4.

Table 5-4 Drifting Gaussians dataset parametric equations

	t=0 to t=1/3				t=1/3 to t=2/3				t=2/3 to t=1			
Class	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y
1	1	1+6t	2	5	1	3	2	5	1	8-9(t-1/3)	8-9(t-1/3)	8-9(t-1/3)
2	1	1	8	5	1	1	8-9(t-1/3)	5	1	1	8-9(t-1/3)	8-9(t-1/3)
3	3-6t	1	5	8	1	1	5+9(t-1/3)	8	1	1	8	8
4	3-6t	1	5	2	1	1	5+9(t-1/3)	2	1	1	8	2

As discussed earlier, current state-of-the-art methods for class imbalance learning from drifting data streams including Learn⁺⁺.NIE, Learn⁺⁺.CDS and ESOS-ELM exist for binary class datasets. We compared the performance of MOS-ELM on the two multiclass datasets with McELM and WOS-ELM as these are the two online learning methods that can work for multiclass problems. McELM inherently works for multiclass problems but its performance has not been evaluated in drifting and highly imbalanced environments. WOS-ELM was originally proposed for binary classification problems, however its weighting matrix is extended to multiclass case, just as for MOS-ELM in Section 5.2.

Table 5-5 G_{mean} values obtained on multiclass drifting datasets

Dataset	MOS-ELM	WOS-ELM	McELM
Gaussian1	92.3	74.6	0.08
Gaussian2	92	68.9	1.2

The optimal number of hidden neurons for WOS-ELM was found to be 40 for Gaussian1 and 20 for Gaussian2. These values are set as L_{max} for MOS-ELM. As can be observed from Table 5-5 and Figures 5-1, 5-2 and 5-3, McELM is not suitable for high imbalance data streams with drifting concepts. It performs even worse than WOS-ELM. There is no mechanism in McELM that can tackle high imbalance ratios since samples from majority and minority classes are treated equally. Also, McELM can only work if the class concepts are not drifting. On the other hand, WOS-ELM can tackle the high imbalance problem but it cannot cope well with drifting concepts so the performance is quite low compared to MOS-ELM on both the datasets.

Figure 5-1: MOS-ELM and WOS-ELM comparison on Gaussian1 dataset

Figure 5-2: MOS-ELM and McELM comparison on Gaussian1 dataset

Figure 5-3: MOS-ELM and WOS-ELM comparison on Gaussian2 dataset

5.4 Discussion

As can be observed from Table 5-2, MOS-ELM clearly outperforms McELM on all the multiclass imbalanced datasets. On Balance, Dermatology, Yeast and Page-Block datasets, the difference in G_{mean} is very large. The reason behind superior performance of MOS-ELM is that it has an inherent mechanism to treat minority class and majority class samples separately. MOS-ELM not only emphasizes on difficult to learn samples but also focuses particularly on minority class samples. On the other hand, McELM only emphasize on difficult to learn samples without considering the level of imbalance in datasets. In the case of Yeast dataset, one of the minority class accuracies is zero thus G_{mean} is able to detect that. In contrast, methods based on overall accuracy or even average accuracy are unable to detect that. From Table 5-3, it can be observed that MOS-ELM achieves superior performance to the McELM method. In the context of sequential learning, MOS-ELM is the only method with the ability to tackle the class imbalance problem in multiclass data streams which no other method has achieved till date.

Chapter 6

CONCLUSIONS AND FUTURE WORKS

6.1 Conclusions

In this thesis, we have presented the research work on two problem domains in sequential learning: class imbalance and concept drift. Three new extreme learning machines (ELM) based sequential classifiers are proposed in this thesis to tackle the class imbalance problem with and without concept drift for binary as well as multiclass datasets.

In Chapter 2, we have proposed an online sequential method for binary class imbalance learning without concept drift. This method is referred to as weighted online sequential extreme learning machine (WOS-ELM). WOS-ELM is different from the typically used data sampling methods in that it can tackle the class imbalance problem in both chunk-by-chunk and one-by-one modes. Each arriving sample can update the

model without any delay. Faster training speed and comparable or better classification performance of the WOS-ELM algorithm than competing data sampling methods have been verified on 15 real-world imbalanced datasets. The proposed WOS-ELM is also seen to outperform OTER on most of the datasets in terms of G_{mean} .

WOS-ELM only considers stationary environments while the method proposed in Chapter 3 can tackle the binary class imbalance problem in both stationary and non-stationary environments, i.e., with and without concept drift. This new method is referred to as ensemble of subset online sequential extreme learning machine (ESOS-ELM). It consists of a main ensemble for classification in the current imbalanced environment, an ELM-Store module for storing information of old concepts and a change detector for promptly detecting concept drifts. An ELM-Store technique, based on ELM theory, is also proposed to retain the prior knowledge of old concepts in recurring environments. ESOS-ELM achieves better performance than that of Learn⁺⁺.NIE(*gm*) using fewer hypotheses. ESOS-ELM also outperforms WOS-ELM, OTER and SMOTE on all 15 imbalanced datasets used in Chapter 2.

WOS-ELM and EOS-ELM proposed in Chapters 2 and 3 are application specific methods. Meta-cognitive online sequential extreme learning machine (MOS-ELM) is proposed in Chapter 4 to self-regulate the learning according to the nature of data stream. A single OS-ELM update equation which can be used for both binary class imbalance and concept drift learning is also proposed. For balanced and stationary data streams, it reduces to standard OS-ELM equation. The performance of MOS-ELM is evaluated under different conditions and compared with methods specific to these conditions. On most of the datasets, MOS-ELM outperforms the competing methods.

Chapters 2, 3 and 4 discuss the class imbalance problem only in binary datasets. In Chapter 5, MOS-ELM is extended to multiclass data streams. In the context of sequential learning, MOS-ELM is the first method to alleviate the imbalance problem in multiclass data streams. MOS-ELM is more versatile than WOS-ELM,

ESOS-ELM and other existing sequential learning methods since it can be applied to different applications including binary class and multiclass, balanced and imbalanced, and stationary and non-stationary distributions. The relationships among the proposed methods and their applicability can be summarized in Figure 6.1.

Figure 6-1: Proposed methods and target applications

6.2 Future Works

Future work can be carried as mentioned below.

6.2.1 Addition of New Classes in Non-Stationary Learning

In non-stationary learning, a new class can be added in the data streams at a later point of time. For example, the classification model can be initially trained to classify the samples into two classes but several time steps later a third class is added due to non-stationary nature of the data stream. A mechanism has to be implemented to detect that a new class has been added. The detection of the new class can become more challenging if the new class is a minority class. The incoming samples from majority class may overshadow the samples of the new minority class. A classifier

trained with fixed number of classes may treat the new minority class as noise. All the methods proposed in this thesis assume that the number of classes do not change over time. MOS-ELM, proposed in Chapter 4, has potential to detect the addition or even removal of classes since it uses meta-cognition. In future, we wish to add class addition/removal capability to MOS-ELM.

6.2.2 Class Imbalance Problem for Semi-supervised Learning

In future we also wish to explore the challenges faced in semi-supervised learning when the dataset is imbalanced. In semi-supervised learning, labels of some of the samples are not known. The information from the unlabeled samples is used to modify the hypothesis learned by the labeled samples alone. In the context of class imbalance learning, it is still an open problem how we use the unlabeled information given that it is coming from skewed class distributions. Similarly, another open problem is how we know if the unlabeled data belongs to balanced or imbalanced class distributions. Semi-supervised class imbalance learning is an uncharted territory and these questions need to be addressed.

6.2.3 Online Sequential Learning using Deep Architectures

In this thesis, the online sequential learning methods proposed for class imbalance and concept drift applications are based on single hidden layer feedforward neural networks (SLFN). In fact, there is no deep sequential learning method in extreme learning machine domain for class imbalance and concept drift to date. Deep learning methods have shown promising results particularly in automatic speech recognition and image classification tasks with big datasets. These methods model high-level abstractions in data by using multiple non-linear transformations. It would

be interesting to come up with an online sequential extreme learning method using deep architectures and also apply it to class imbalance and concept drift problems.

6.2.4 Optimization based OS-ELM for Class Imbalance and Concept Drift

In future, we wish to propose online sequential learning methods which are derived from optimization based extreme learning machine. The online sequential learning methods proposed in this thesis are based on basic extreme learning machine with random features. It would be interesting to see how the optimization based online sequential extreme learning method performs in class imbalance and concept-drifting environments. It is expected to have better generalization performance than the methods proposed in this thesis. However, the performance gain may come at a cost of higher computational complexity.

AUTHOR'S PUBLICATIONS

Journal Papers

Bilal Mirza, Zhiping Lin, Kar-Ann Toh, "Weighted Online Sequential Extreme Learning Machine for Class Imbalance Learning," *Neural Processing Letters*, 38(3): 465-486, 2013.

Bilal Mirza, Zhiping Lin, Nan Liu, "Ensemble of Subset Online Sequential Extreme Learning Machine for Class Imbalance and Concept Drift," *Neurocomputing*, 149(A): 316-329, 2015.

Nan Liu, Zhi Xiong Koh, E. Chua, L. Tan, Zhiping Lin, Bilal Mirza, and M. Ong, "Risk Scoring for Prediction of Acute Cardiac Complications From Imbalanced Clinical Data," *IEEE Journal of Health and Bioinformatics*, 2014.
DOI 10.1109/JBHI.2014.2303481

Bilal Mirza, Zhiping Lin, "Meta-Cognitive Online Sequential Extreme Learning Machine for Imbalance and Non-stationary Data streams," submitted to *IEEE Transactions on Knowledge and Data Engineering*.

Conference Papers

Bilal Mirza, Zhiping Lin, "One-vs-All for Class Imbalance Learning," *In Proceedings of 9th International Conference on Information, Communications and Signal Processing (ICICS)*, Tainan, Taiwan, 10-13 December 2013.
DOI 10.1109/ICICS.2013.6782785

Bilal Mirza, Zhiping Lin, Jiuwen Cao, Xiaoping Lai, "Voting based Weighted Online Sequential Extreme Learning Machine for Imbalance Multi-Class Classification," *IEEE International Symposium on Circuits and Systems (ISCAS)*, Lisbon, Portugal, 2015. Accepted.

BIBLIOGRAPHY

- [1] T. R. Hoens, R. Polikar, N. V. Chawla, "Learning from Streaming Data with Concept Drift and Imbalance: An Overview," *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 89-101, 2012.
- [2] G. Ditzler, R. Polikar, N. V. Chawla, "An Incremental Learning Algorithm for Non-Stationary Environments and Class Imbalance," *Int. Conf. on Pattern Recognition*, pp. 2997-3000, 2010.
- [3] G. Ditzler, R. Polikar, "Incremental Learning of Concept Drift from Streaming Imbalanced Data," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 10, pp. 2283-2301, 2012.
- [4] Moreno-Torres, G. Jose, F. Herrera, "A Preliminary Study on Overlapping and Data Fracture in Imbalanced Domains by Means of Genetic Programming-Based Feature Extraction," *IEEE 10th Int. Conf. on Intelligent Systems Design and Applications (ISDA)*, pp. 501-506, 2010.
- [5] R. Polikar, L. Upda, S. S. Upda, V. Honavar, "Learn++: An Incremental Learning Algorithm for Supervised Neural Networks," *IEEE Trans. Syst. Man Cybern. C, Human Syst.*, vol. 31, no. 4, pp. 497-508, 2001.
- [6] A. Shilton, M. Palaniswami, D. Ralph, A. C. Tsoi, "Incremental Training of Support Vector Machines," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 114-131, 2005.
- [7] M. Muhlbaier, R. Polikar, "Multiple Classifiers Based Incremental Learning Algorithm for Learning Nonstationary Environments," *Int. Conf. on Machine Learning and Cybern.*, pp. 3618-3623, 2007.
- [8] N. Y. Lian, G. B. Huang, P. Saratchandran, N. Sundararajan, "A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1411-1423, 2006.
- [9] G. B. Huang, P. Saratchandran, N. Sundararajan, "An Efficient Sequential Learning Algorithm for Growing and Pruning RBF (GAP-RBF) Networks," *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 34, no. 6, pp. 2284-2292, 2004.
- [10] G. B. Huang, P. Saratchandran, N. Sundararajan, "A Generalized Growing and Pruning RBF (GGAP-RBF) Neural Network for Function Approximation," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 57-67, 2005.
- [11] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla, F. Herrera, "A Unifying View on Dataset Shift in Classification," *Pattern Recognition*, vol. 45, no. 1, pp. 521-530, 2012.
- [12] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, R. Gavalda, "New Ensemble Methods for Evolving Data Streams," *15th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pp. 139-148, 2009.
- [13] A. Bifet, R. Gavalda, "Adaptive Learning from Evolving Data Streams," *Advances in Intelligent Data Analysis VIII Springer Berlin Heidelberg*, pp. 249-260, 2009.
- [14] A. Bifet, R. Gavalda, "Learning from Time-Changing Data with Adaptive Windowing," *SDM*, vol. 7, pp. 443-448, 2007.
- [15] J. Gama, P. Medas, G. Castillo, P. Rodrigues, "Learning with Drift Detection," *Advances in Artificial Intelligence-SBIA*, pp. 286-295, 2004.
- [16] M. Karnick, M. D. Muhlbaier, R. Polikar, "Incremental Learning in Non-stationary Environments with Concept Drift using a Multiple Classifier based Approach," *19th IEEE Int. Conf. on Pattern Recognition*, December 2008.
- [17] R. Elwell, R. Polikar, "Incremental Learning in Nonstationary Environments with Controlled Forgetting," *IEEE Int. Joint Conf. on Neural Netw.*, pp. 771-778, 2009.
- [18] H. He, E. A. Garcia, "Learning from Imbalanced Data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263-1284, 2009.
- [19] G. Weiss, "Mining with Rarity: A Unifying Framework," *ACM SIGKDD Explor. Newslett.*, vol. 6, no. 1, pp. 7-19, 2004.
- [20] N. Japkowicz, S. Stephen, "The Class Imbalance Problem: A Systematic Study," *Intelligent Data Analysis*, vol. 6, no. 5, pp. 429-449, 2002.

- [21] K. Veropoulos, C. Campbell, N. Cristianini, "Controlling the Sensitivity of Support Vector Machines," *Int. Joint Conf. on Artif. Intell.*, pp. 55–60, Stockholm, Sweden 1999.
- [22] R. Batuwita, V. Palade, "AGm: A New Performance Measure for Class Imbalance Learning. Application to Bioinformatics Problems," *8th IEEE Int. Conf. on Machine Learning and Applications*, pp. 545-550, 2009.
- [23] R. Batuwita, V. Palade, "Adjusted Geometric-Mean: A Novel Performance Measure for Imbalanced Bioinformatics Datasets Learning," *Journal of Bioinformatics and Computational Biology*, vol. 10, no. 4, 2012.
- [24] R. Batuwita, V. Palade, "Class Imbalance Learning Methods for Support Vector Machines," *Imbalanced Learning: Foundations, Algorithms, and Applications*, Haibo He and Yunqian Ma (Eds.), Wiley, (book chapter in press), 2012.
- [25] R. Batuwita, V. Palade, "FSVM-CIL: Fuzzy Support Vector Machines for Class Imbalance Learning," *IEEE Trans. Fuzzy Syst.*, vol. 18, no. 3, pp. 558-571, 2010.
- [26] J. R. Akbani, S. Kwek, N. Japkowicz, "Applying Support Vector Machines to Imbalanced Datasets," *15th Eur. Conf. on Machine Learning*, pp. 39-50, 2004.
- [27] N. García-Pedrajas, J. Pérez-Rodríguez, A. de Haro-García, "OligoIS: Scalable Instance Selection for Class-Imbalanced Data Sets," *IEEE Trans. Cybern.*, vol. 43, no. 1, pp. 332-346, 2013.
- [28] V. López, A. Fernández, S. García, V. Palade, F. Herrera, "An Insight into Classification with Imbalanced Data: Empirical Results and Current Trends on Using Data Intrinsic Characteristics," *Information Sciences*, vol. 250, pp. 113-141, 2013.
- [29] U. Bhowan, M. Johnston, M. Zhang, X. Yao, "Evolving Diverse Ensembles Using Genetic Programming for Classification with Unbalanced Data," *IEEE Trans. Evolutionary Computation*, vol. 17, no. 3, pp. 368-386, 2013.
- [30] M. Lin, K. Tang, X. Yao, "Dynamic Sampling Approach to Training Neural Networks for Multiclass Imbalance Classification," *IEEE Trans. Neural Netw. and Learning Syst.*, vol. 24, no. 4, pp. 647-660, 2013.
- [31] B. X. Wang, N. Japkowicz, "Boosting Support Vector Machines for Imbalanced Data sets," *Knowl. Inf. Syst.*, vol. 25, no. 1, pp. 1-20, 2010.
- [32] Y. Tang, Y. Zhang, N. V. Chawla, S. Krasser, "SVMs Modeling for Highly Imbalanced Classification," *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 39, no. 1, pp. 281-288, 2009.
- [33] T. Imam, K. Ting, J. Kamruzzaman, "z-SVM: An SVM for Improved Classification of Imbalanced data," *19th Aust. Joint Conf. AI*, pp. 264-273, 2006.
- [34] G. Wu, E. Y. Chang, "KBA: Kernel Boundary Alignment Considering Imbalanced Data Distribution," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 6, pp. 786-795, 2005.
- [35] J. Wang, J. You, Q. Li, Y. Xu, "Extract Minimum Positive and Maximum Negative Features for Imbalanced Binary Classification," *Pattern Recognition*, vol. 45, no. 3, pp. 1136-1145, 2012.
- [36] W. J. Lin, J. J. Chen, "Class-Imbalanced Classifiers for High-Dimensional Data," *Briefings in Bioinformatics*, vol. 41, no. 1, pp. 13-26, 2012.
- [37] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, F. Herrera, "A Review on Ensembles for the Class Imbalance Problem: Bagging, Boosting, and Hybrid Based Approaches," *IEEE Trans. Syst. Man Cybern. C, Applications and Reviews*, vol. 42, no. 4, pp. 463-484, 2012.
- [38] A. Orriols-Puig, E. Bernadó-Mansilla, "Evolutionary Rule-Based Systems for Imbalanced Data Sets," *Soft Computing*, vol. 13, no. 3, pp. 213-225, 2009.
- [39] M. Wasikowski, X. W. Chen, "Combating the Small Sample Class Imbalance Problem Using Feature Selection," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1388-1400, 2010.
- [40] Y. Sun, A. K. Wong, M. S. Kamel, "Classification of Imbalanced Data: A Review," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 23, no. 4, pp. 687-719, 2009.
- [41] J. V. Hulse, T. M. Khoshgoftaar, A. Napolitano, "An Empirical Comparison of Repetitive Undersampling Techniques," *IEEE Int. Conf. on Information Reuse & Integration*, pp.

- 29-34, 2009.
- [42] C. Cortes, V. Vapnik, "Support Vector Networks," *Machine Learning*, vol. 20, pp. 273-297, 1995.
 - [43] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Mining Knowl. Discovery*, vol. 2, pp. 121-167, 1998.
 - [44] G. B. Huang, Q. Y. Zhu, C. K. Siew, "Extreme Learning Machine: Theory and Applications," *Neurocomputing*, vol. 70, pp. 489-501, 2006.
 - [45] G. B. Huang, H. Zhou, X. Ding, R. Zhang, "Extreme Learning Machine for Regression and Multiclass Classification," *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 42, no. 2, pp. 513-529, 2012.
 - [46] W. Zong, G. B. Huang, Y. Chen, "Weighted Extreme Learning Machine for Imbalance Learning," *Neurocomputing*, vol. 101, pp. 229-242, 2013.
 - [47] M. Kubat, S. Matwin, "Addressing the Curse of Imbalanced Training Sets: One-Sided Selection," *Int. Conf. on Machine Learning*, pp. 179-186, 1997.
 - [48] C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse, A. Napolitano, "RUSBoost: A Hybrid Approach to Alleviating Class Imbalance," *IEEE Trans. Syst. Man Cybern. A, Human Syst.*, vol. 40, no. 1, pp. 185-197, 2010.
 - [49] X. Y. Liu, J. Wu, Z. H. Zhou, "Exploratory Under-Sampling for Class-Imbalance Learning," *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 39, no. 2, pp. 539-550, 2009.
 - [50] N. V. Chawla, A. Lazarevic, L. O. Hall, K. Bowyer, "SMOTEBoost: Improving Prediction of the Minority Class in Boosting," *Proc. Principles Know. Discov. Databases*, pp. 107-119, 2003.
 - [51] N. V. Chawla, K. W. Bowyer, L. O. Hall, W.P. Kegelmeyer, "SMOTE: Synthetic Minority Over-Sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002.
 - [52] C. X. Ling, V. S. Sheng, "Cost-Sensitive Learning and the Class Imbalance Problem," *Encyclopedia of Machine Learning (C. Sammut, ed.), Springe*, 2008.
 - [53] M. A. Maloof, "Learning When Data Sets Are Imbalanced and When Costs Are Unequal and Unknown," *Int. Conf. on Machine Learning, Workshop Learning from Imbalanced Data Sets II*, 2003.
 - [54] Y. Sun, M. S. Kamel, A. K. C. Wong, Y. Wang, "Cost-Sensitive Boosting for Classification of Imbalanced Data," *Pattern Recognition*, vol. 40, no. 12, pp. 3358-3378, 2007.
 - [55] W. Fan, S. J. Stolfo, J. Zhang, P.K. Chan, "AdaCost: Misclassification Cost-Sensitive Boosting," *Int. Conf. on Machine Learning*, pp. 97-105, 1999.
 - [56] K. M. Ting, "An Empirical Study of MetaCost Using Boosting Algorithm," *11th Eur. Conf. on Machine Learning*, pp. 413-425, 2000.
 - [57] P. K. Chan, S. J. Stolfo, "Toward Scalable Learning with Non-uniform Class and Cost Distributions: A Case Study in Credit Card Fraud Detection," *4th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pp. 164-168, 1998.
 - [58] H. Guo, H. Viktor, "Learning from Imbalanced Data sets with Boosting and Data Generation: the DataBoost IM approach," *ACM SIGKDD Explor. Newslett.*, vol. 6, no. 1, pp. 30-39, 2004.
 - [59] P. Viola, M. Jones, "Fast and Robust Classification Using Asymmetric AdaBoost and a Detector Cascade," *Neural Inf. Process. Syst.*, pp. 1311-1318, 2001.
 - [60] G. Ditzler, M. Muhlbaier, R. Polikar, "Incremental Learning of New Classes in Unbalanced Datasets: Learn++. UDNC," *MCS Lecture Notes in Computer Science*, vol. 5997, pp. 33-42, 2010.
 - [61] J. Gao, W. Fan, J. Han, P. S. Yu, "A General Framework for Mining Concept-drifting Streams with Skewed Distribution," *SIAM Int. Conf. on Data Mining*, vol. 7, 2007.
 - [62] S. Chen, H. He, "SERA: Selectively Recursive Approach Towards Nonstationary Imbalanced Stream Data Mining," *Int. Joint Conf. on Neural Netw.*, pp. 522-529, 2009.
 - [63] S. Chen, H. He, "Towards Incremental Learning of Nonstationary Imbalanced Data Stream: A Multiple Selectively Recursive Approach," *Evolving Systems*, vol. 2, no. 1, pp.

- 35-50, 2011.
- [64] Y. Wang, Y. Zhang, Y. Wang, "Mining Data Streams with Skewed Distribution by Static Classifier Ensemble," *Studies in Computational Intelligence*, vol. 214, pp. 65-71, 2009.
 - [65] H. Nguyen, E. Cooper, K. Kamei, "Online Learning from Imbalanced Data Streams," *Int. Conf. of Soft Computing and Pattern Recognition (SoCPaR)*, pp. 347-352, 2011.
 - [66] R. Lichtenwalter, N. V. Chawla, "Adaptive Methods for Classification in Arbitrarily Imbalanced and Drifting Data Streams," *In: New Frontiers in Applied Data Mining, Lecture Notes in Computer Science*, vol. 5669, pp. 53-75, 2010.
 - [67] T. R. Hoens, N. V. Chawla, "Learning in Non-stationary Environments with Class Imbalance," *Int. Conf. on Knowledge Discovery and Data Mining*, pp. 168-176, 2012.
 - [68] Y. Kim, K. A. Toh, A. B. J. Teoh, H. L. Eng, W.Y. Yau, "An Online Learning Network for Biometric Scores Fusion," *Neurocomputing*, vol. 102, pp. 65-77, 2013.
 - [69] R. Elwell, R. Polikar, "Incremental Learning of Concept Drift in Nonstationary Environments," *IEEE Trans. Neural Netw.*, vol. 22, no. 10, pp. 1517-1531, 2011.
 - [70] K. Nishida, K. Yamauchi, T. Omori, "ACE: Adaptive Classifiers-Ensemble System for Concept-Drifting Environments," *Multiple Classifier Systems*, pp. 176-185, 2005.
 - [71] J. Zhao, Z. Wang, D. S. Park, "Online Sequential Extreme Learning Machine with Forgetting Mechanism," *Neurocomputing*, vol. 87, pp. 79-89, 2012.
 - [72] Y. Gu, J. Liu, Y. Chen, X. Jiang, H. Yu, "TOSELM: Timeliness Online Sequential Extreme Learning Machine," *Neurocomputing*, vol. 128, pp. 119-127, 2014.
 - [73] S. Wang, X. Yao, "Multiclass Imbalance Problems: Analysis and Potential Solutions," *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 42, no. 4, pp. 1119-1130, 2012.
 - [74] K. Chen, B. L. Lu, J. T. Kwok, "Efficient Classification of Multi-label and Imbalanced Data Using min-max Modular Classifiers," *Int. Joint Conf. on Neural Netw.*, pp. 1770-1775, 2006.
 - [75] A. C. Tan, D. Gilbert, Y. Deville, "Multi-Class Protein Fold Classification Using a New Ensemble Machine Learning Approach," *Genome Inf.*, vol. 14, pp. 206-217, 2003.
 - [76] X. M. Zhao, X. Li, L. Chen, K. Aihara, "Protein Classification with Imbalanced Data," *Proteins: Structure, Function, and Bioinformatics*, vol. 70, no. 4, pp. 1125-1132, 2008.
 - [77] T. W. Liao, "Classification of Weld Flaws with Imbalanced Class Data," *Expert Syst. Appl.*, vol. 35, no. 3, pp. 1041-1052, 2008.
 - [78] T. Nelson, L. Narens, "Metacognition: core readings, ch. Metamemory: a theoretical framework and new findings," *Allyn and Bacon: Boston, T. O. Nelson (ed.) ed.*, pp. 9-24, 1980.
 - [79] R. M. Isaacson, F. Fujita, "Metacognitive Knowledge Monitoring and Self-Regulated Learning: Academic Success and Reflections on Learning," *Journal of Scholarship of Teaching and Learning*, vol. 6, no. 1, pp. 39-55, 2006.
 - [80] G. Hulten, L. Spencer, P. Domingos, "Mining Time-Changing Data Streams," *7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pp. 97-106, 2001.
 - [81] C. Alippi, M. Roveri, "Just-in-time Adaptive Classifiers in Non-Stationary Conditions," *IEEE Int.l Joint Conf. on Neural Netw.*, pp. 1014-1019, August 2007.
 - [82] G. Widmer, M. Kubat, "Learning in the Presence of Concept Drift and Hidden Contexts," *Machine learning*, vol. 23, no. 1, pp. 69-101, 1996.
 - [83] M. Lazarescu, S. Venkatesh, H. Bui, "Using Multiple Windows to Track Concept Drift," *Intelligent data analysis*, vol. 8, no. 1, pp. 29-59, 2004.
 - [84] C. Alippi, G. Boracchi, M. Roveri, "Just in time Classifiers: Managing the Slow Drift Case," *IEEE Int. Joint Conf. on Neural Netw.*, pp. 114-120, 2009.
 - [85] W. N. Street, Y. Kim, "A Streaming Ensemble Algorithm (SEA) for Large-scale Classification," *7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pp. 377-382, 2001.
 - [86] H. Wang, W. Fan, P. S. Yu, J. Han, "Mining Concept-Drifting Data Streams Using Ensemble Classifiers," *9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pp. 226-235, August 2003.
 - [87] J. Z. Kolter, M. A. Maloof, "Dynamic Weighted Majority: An Ensemble Method for

- Drifting Concepts," *The Journal of Machine Learning Research*, vol. 8, pp. 2755-2790, 2007.
- [88] K. A. Toh, H. L. Eng, "Between Classification-Error Approximation and Weighted Least-Squares Learning," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 30, no. 4, pp. 658-669, 2008.
 - [89] K. A. Toh, "Deterministic Neural Classification," *Neural Comput.*, vol. 20, no. 6, pp. 1565-1595, 2008.
 - [90] S. Wang, X. Yao, "Relationships Between Diversity of Classification Ensembles and Single-Class Performance Measures," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 1, pp. 206-219, 2013.
 - [91] G. Ditzler, R. Polikar, N. V. Chawla, "An Incremental Learning Algorithm for Non-Stationary Environments and Class Imbalance," *Int. Conf. on Pattern Recognition*, pp. 2997-3000, 2010.
 - [92] A. Frank, A. Asuncion. University of California, Irvine, School of Information and Computer Sciences: Machine Learning Repository. [Online]. <http://archive.ics.uci.edu/ml>
 - [93] M. Kubat, R. C. Holte, S. Matwin, "Machine Learning for the Detection of Oil Spills in Satellite Radar Images," *Machine Learning*, vol. 30, pp. 195-215, 1998.
 - [94] J. Demsar, "Statistical Comparisons of Classifiers Over Multiple Data Sets," *Journal of Machine Learning Research*, vol. 7, pp. 1-30, 2006.
 - [95] Y. Lan, Y. C. Soh, G. B. Huang, "Ensemble of Online Sequential Extreme Learning Machine," *Neurocomputing*, vol. 72, pp. 3391-3395, 2009.
 - [96] J. Cao, Z. Lin, G. B. Huang, N. Liu, "Voting Based Extreme Learning Machine," *Information Sciences*, vol. 185, no. 1, pp. 66-77, 2012.
 - [97] J. Cao, Z. Lin, G. B. Huang, "Voting Base Online Sequential Extreme Learning Machine for Multi-Class Classification," *IEEE ISCAS 2013*, pp. 2327 - 2330, 2013.
 - [98] F. Chu, C. Zaniolo, "Fast and Light Boosting for Adaptive Mining of Data Streams," *Advances in Knowledge Discovery and Data Mining, LNCS*, vol. 3056, pp. 282-292, 2004.
 - [99] J. C. Schlimmer, R. H. Granger, "Beyond Incremental Processing: Tracking Concept Drift," *5th National Conf. on Artif. Intell.*, vol. 1, pp. 502-507, 1986.
 - [100] G. Ditzler, R. Polikar. Benchmark Datasets for Evaluating Concept Drift/Imbalanced Data Algorithms. [Online]. http://users.rowan.edu/~polikar/research/NIE_data
 - [101] M. Harries, New South Wales. (2013, Apr.) Splice-2 Comparative Evaluation: Electricity Pricing. [Online]. <ftp://ftp.cse.unsw.edu.au/pub/doc/papers/UNSW/9905.pdf>
 - [102] G. S. Babu, S. Suresh, "Meta-Cognitive Neural Network for Classification Problems in a Sequential Learning Framework," *Neurocomputing*, vol. 81, pp. 86-96, 2012.
 - [103] G. S. Babu, S. Suresh, "Sequential Projection-Based Metacognitive Learning in a Radial Basis Function Network for Classification Problems," *IEEE Trans. Neural Netw. and Learning Syst.*, vol. 24, no. 2, pp. 194-206, 2013.
 - [104] R. Savitha, S. Suresh, H. J. Kim, "A Meta-Cognitive Learning Algorithm for an Extreme Learning Machine Classifier," *Cognitive Computation*, vol. 6, no. 2, pp. 253-263, 2014.
 - [105] S. Suresh, N. Sundararajan, P. Saratchandran, "Risk-Sensitive Loss Functions for Sparse Multi-Category Classification Problems," *Information Sciences*, vol. 178, no. 12, pp. 2621-2638, 2008.
 - [106] Z. H. Zhou, X. Y. Liu, "Training Cost-Sensitive Neural Networks with Methods Addressing the Class Imbalance Problem," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 1, pp. 63-77, 2006.