M.Sc. Engg. Thesis

Label Based Ensemble Framework for Multi-label Data Stream Classification with Recurring and Novel Class Detection

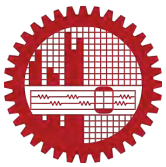
by

Sajjadur Rahman

Submitted to

Department of Computer Science and Engineering

in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science and Engineering



Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology (BUET)

Dhaka 1000

June 2014

Dedicated to my loving parents

Author's Contact

Sajjadur Rahman

Lecturer

Department of Computer Science & Engineering

Bangladesh University of Engineering & Technology (BUET).

Email: sunnysajjad@cse.buet.ac.bd

The thesis titled "Label-Based Ensemble Framework for Multi-label Data Stream Classification with Recurring and Novel Class Detection", submitted by Sajjadur Rahman, Roll No. **0412052022P**, Session April 2012, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfilment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on June 02, 2014.

Board of Examiners

1. _____

Dr. Md. Monirul Islam — Chairman (Supervisor)
Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

2. _____

Dr. Mohammad Mahfuzul Islam — Member (Ex-Officio)
Head and Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

3. _____

Dr. Md. Monirul Islam — Member
Assistant Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

4. _____

Dr. A. B. M. Alim Al Islam — Member
Assistant Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

5. _____

Dr. Chowdhury Mofizur Rahman — Member (External)
Professor
Department of Computer Science and Engineering
United International University, Dhaka.

Candidate's Declaration

This is hereby declared that the work titled "Label-Based Ensemble Framework for Multi-label Data Stream Classification with Recurring and Novel Class Detection" is the outcome of research carried out by me under the supervision of Prof. Dr. Md. Monirul Islam, in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka-1000. It is also declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Sajjadur Rahman

Candidate

Acknowledgment

First of all, I would like to thank my supervisor, Prof. Dr. Md. Monirul Islam for introducing me to the fascinating field of data stream mining. Without his continuous supervision, guidance and valuable advice, it would have been impossible to complete the thesis. I am especially grateful to him for allowing me to chose my course of actions with freedom, for enlightening me during the moments of frustration, for his patience throughout the entire research work and for his moral support to the extra-curricular activities that I accomplished aside from the thesis. His patient hearing of my ideas, critical analysis of my observations and detecting flaws (and amending thereby) in my thinking and writing have made this thesis a success.

I would also want to thank the members of my thesis committee for their valuable suggestions. I thank Dr. Mohammad Mahfuzul Islam, Dr. Md. Monirul Islam, Dr. A. B. M. Alim Al Islam and specially the external member Dr. Chowdhury Mofizur Rahman.

In this regard, I remain ever grateful to my beloved parents, who always exists as sources of inspiration behind every success of mine I have ever made.

Abstract

Of late, the advent of online social media has led to the inception of a new form of data stream called multi-label data stream, where each stream record carries multiple class labels and requires a classifier to associate multiple categories to each record. Data streams present several challenges that has to be dealt with by any stream classification model. Concept drifting, infinite length with finite memory and processing time are the challenges that have been addressed by the existing multi-label data stream classification models in literature. In real world applications that generate data streams, the amount of labeled data is usually very scarce compared to the entire stream. Moreover, with the ever changing nature of Internet and social media, the emergence new class of data in the stream is a common phenomenon. This phenomenon is known as concept evolution. When this emergence occurs periodically for some classes of data, it is called class recurrence. None of the existing methodologies address any of the issues of scarcity of labeled data, concept evolution and class recurrence.

This thesis proposes a layered ensemble based classification framework ($LEAD$) for multi-label data streams. The primary component of our $LEAD$ framework is a two layer ensemble architecture. The top layer of the ensemble architecture reflects the most recent concept of the data stream whereas the bottom layer represents the older concepts of the stream. As a result, the bottom layer enables $LEAD$ to classify recurrent class instances. Moreover, the layered approach also helps to differentiate between recurrent and novel class instances which significantly reduces the false alarm rate of novel class instance identification. $LEAD$ deploys a fuzzy novel class detection technique to identify the emergence of novel concept(s) in the stream. The problem of limited amount of labeled data is handled by a deferred classification mechanism. This mechanism allows more labeled data to appear in the stream that may help the development of a more informed classifier. Experimental results show clearly that $LEAD$ exhibits better performance than the baseline methods.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Data stream is a large, continuous and high speed sequence of data. The data stream model is motivated by emerging application involving massive data sets, such as, online social media (Twitter, Facebook, blogs etc.), telephone records, large sets of web pages, multimedia data and so on. Being an effective tool of data mining, classification in streaming environment has attracted special interest from researchers. Infinite stream with finite processing resources (time and memory), one pass learning, concept drift [1], concept evolution [2] and class recurrence [2] are the issues most frequently encountered by the data stream classification models. Training a classifier with limited amount of labeled data is another challenge in the data stream scenario [3]. In a real world streaming environment, a huge volume of data appears at high speed where majority of the data is unlabeled. The challenge for algorithm designers is to perform meaningful computation with these restrictions.

Data stream classification is the machine learning technique of associating class label(s) to an instance of the stream. Single label data stream classification is a common learning problem where the goal is to classify each instance to a unique class label from a set of disjoint class labels L. Depending on the total number of disjoint classes in L, the problem can be identified as binary classification (when $|L| = 2$) or multi-class classification (when $|L| > 2$) problem. Unlike the single label data stream classification problems, multi-label classification allows the instances to be associated with more than one class label. With the proliferation of online social media, many real world applications generate data streams where each stream record carries multiple class labels. For example, a news article in on the disappearance of the Malaysian Airlines flight MH370 could be annotated with labels like “Asia news”, “Tragic event”, “Rescue” etc. Thus, the goal of multi-label data stream classification

is to assign a set of class labels to an instance in the stream. A detailed discussion on single label and multi-label classification is provided in Section 2.1.

In recent years, multi-label data stream classification has garnered significant interest. Two approaches for multi-label stream classification exist in literature. They are: instance-incremental methods and batch-incremental methods. The instance incremental methods [4]–[5] learn from each example as it arrives. All these methods maintain a hypothetical window of instances to train a classification model. The use of such a window also helps to detect concept drifting. The accuracy drop between consecutive windows gives an indication of concept drift. The batch-incremental methods [6]–[7], on the other hand, gather examples in batches to learn a classification model. The batch incremental methods can be further classified as chunk based and labels based approaches. In chunk based ensemble methods, an ensemble of classifiers is maintained. A classifier is learnt from each new chunk encountered in the stream. Then the oldest or most poorly performing classifier in the ensemble is replaced by the newly learned classifier. Instead of learning a new classifier per chunk, the label based batch-incremental methods maintain an ensemble of binary classifiers per class label. Whenever a new chunk arrives, new classifiers are learned. The newly learned classifier replaces the oldest or poorest classifier of only the corresponding class's ensemble. In both cases, the classifier replacement strategy keeps the model up to date in the presence of concept drift.

1.1 Related works

In this section, we briefly describe the existing multi-label data stream classification methods. We first discuss several works based on the instance incremental approach and then move on to the batch-incremental approaches. We also highlight the challenges handled by the existing methods and discuss the deficiencies of these methods.

1.1.1 Instance incremental methods

Streaming Multi-label Random Trees (SMART) [4] is an instance incremental approach that uses a random tree based ensemble technique for multi-label stream classification. This approach stems from a mainstream idea for learning label relevance and regression that builds an ensemble of multiple random trees [8]. At the beginning of the data stream, a set of binary random trees with height d is trained where d is a parameter of the model and corresponds to the tree sizes. In each tree node,

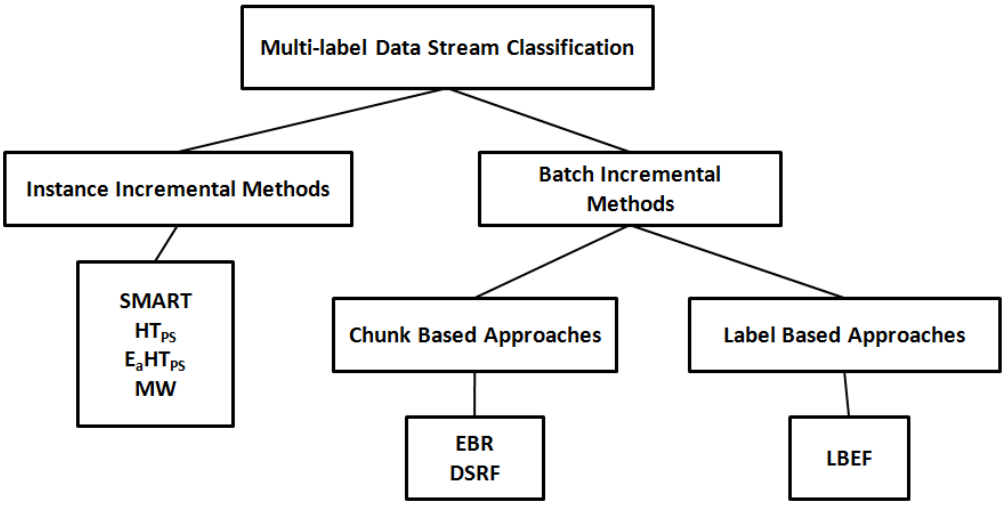


Figure 1.1: Multi-label data stream classification methods

a testing feature is randomly selected with a random splitting value within the valid range. Thus, with each tree node, the feature space is partitioned into two subspaces. All the nodes in random trees are trained from the labeled instances in the stream only. The multi-label random trees are built at the beginning of data stream and timely updated over the data stream as follows: The tree building process starts with all empty tree nodes. When training data points in the data stream arrive, each node keeps updating two types of statistical information for the multiple label data stream: (1) the probabilities of label relevances that traverse through that node, which is similar to conventional decision trees. (2) the estimated cardinality of the label sets in that node, which is similar to regression trees. SMART uses a simple "fading function" at the nodes of each tree in the ensemble to address the concept drifting problem. The fading function gradually reduces the influence of the historical data with the passage of time. The SMART framework can be explained as follows: (1) Ranking: First it learns a ranking function which estimates the probability of relevance for each label. (2) Regression: Then an additional function to estimate the label set cardinality for each instance is learned. (3) Label Set Prediction: Then for each instance, the labels are ranked according to the output of the ranking function, and the label set is predicted with the top ranked labels. For example, suppose the ranked order of three labels is $(l3 > l1 > l2)$, and the estimated label set cardinality is 2. Then the predicted label set for an instances would be $\{l1, l3\}$. This work is compared with ML-kNN [9] (an outdated multi-label k-NN based classifier) and shows only minor improvement.

Read *et al.* [10] on the other hand use Hoeffding trees [11], a special class of trees. Hoeffding

Tree is a state-of-the-art classifier for single-label data streams. A Multi-label Hoeffding Tree is an incremental decision tree classifier for multi-label data streams that it is based on the use of the Hoeffding bound as a criterion to decide whether to split the nodes. C4.5 algorithm can be adapted to perform multi-label data classification [12]. [10] uses the same strategy to develop a decision tree for multi-label data streams. It proposes a new definition of entropy to compute information gain. Entropy is a measure of the amount of uncertainty in the dataset. For each example in the dataset, it is the information needed to describe all the classes it belongs to. In case of multi-label examples, a computation of the information needed to describe all the classes that it doesn't belong to is added to the entropy definition. Moreover, pruned set (PS) classifier [13] is used as the base multi-label classifier in the leaf nodes of the tree. PS prunes away infrequently occurring label sets which eliminates much unnecessary and detrimental complexity. A post-pruning step breaks up the pruned sets into more frequently occurring subsets, and is able to reintroduce pruned instances into the data, ensuring minimal information loss.

Ensemble of multi-label Hoeffding trees with PS classifiers at the leaves of the trees, E_aHT_{PS} [14] is an extension of [10], which deploys an adaptive window based drift detection technique. This paper proposes a new experimental framework for learning and evaluating on multi-label data streams, and uses it to study the performance of various methods. Also, a method for generating synthetic multi-label concept drifting data stream is also presented in [14]. PS is employed at each leaf of a Hoeffding Tree as in [10]. Each new leaf is initialised with a majority labelset classifier. After a number of examples (1000 instances are allowed) fall into the leaf, a PS classifier is established to model the combinations observed over these examples, and then proceeds to build this model incrementally with new instances. On smaller datasets, the majority label set classifier may be in use in many leaves during much of the evaluation. The Hoeffding Tree is then wrapped in a bagged ensemble with a change detector to allow adaption to concept drift in an evolving stream.

In [10] and [14], an adaptive sized window is maintained along the data stream. Here, the window is divided into fragments. Whenever there is a drop of accuracy between subsequent fragments, concept drift is detected. Both [10] and [14] exhibit significant improvement compared to existing baseline methods. However, both the approaches are supervised, i.e., they require all the instances in the data stream to be labeled.

Multiple windows (MW) [5] adopts a multiple windows approach. To deal with concept drift and skewness in the distribution of positive and negative examples of each label, it maintains two fixed-size

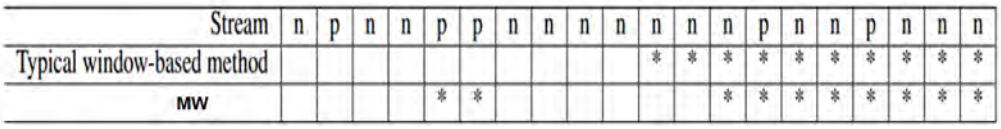


Figure 1.2: Window Construction for MW Method.

windows per label: one for positive examples and one for negative examples. The multiple windows approach employs k-nearest neighbor for classification with Binary Relevance (BR) method [15] as base multi-label classifier. The size of the positive windows is a parameter of the approach. The positive window size should be large enough to allow learning an accurate model and small enough to reduce the probability of concept drift in the window. Based on the size of the positive window, the number of examples in the negative windows is determined using a distribution ratio R ($0.3 < R < 0.5$). Figure 1.2 shows how MW constructs the positive and negative windows for a given class label. The windows are maintained as queues, i.e., the oldest instance is always deleted when the maximum window size is exceeded. An instance is deleted from buffer only after it has been deleted from all the label windows. Whenever an unseen class label appears a new positive and a new negative queue are created.

To deal with class imbalance, MW deploys an incremental thresholding technique. It helps to reduce the impact of class imbalance which is known to negatively affect the performance of the classifiers. Each k-NN classifier actually computes an estimation of the probability of the corresponding label being relevant. For each label, a different threshold is computed. For n instances, a confidence score is maintained per label. Higher the number of instances belonging to a label, higher the confidence score. For each label, the value which would more accurately approximate the observed label frequency in these n instances is selected as the threshold for that label. These threshold values are used for the next n instances and re-calculated after those instances are processed.

1.1.2 Batch incremental methods

As shown in Figure 1.1, there are two chunk based and one label based batch incremental methods of multi-label data stream classification. An universal problem all the chunk based approaches face is that the absence of a class in the current chunk leads to absence of corresponding classifier in the model learnt from that chunk. This may worsen the performance of the classification framework. The label based approach tries to address this issue.

Chunk based approaches

The first batch-incremental scheme was proposed by Qu *et al.* [6]. It assumes that stream instances arrive in chunks of size S and builds an ensemble of N classifiers for N successive chunks. To deal with concept drift, the oldest classifier is replaced by a classifier built on the latest chunk. Stacked binary relevance [16] method is used to learn from each chunk, but the method proposed in [6] could be coupled with any batch incremental multi-label learner. When learning for label l, 1) the weights of additional features, which are unrelated to l, are set to 0, as they provide no useful information for learning label l, and furthermore, they may induce noise information. Here, any certain feature selection algorithm, such as information gain, can be used to determine whether a label is related to l. 2) The weight of a related additional feature of label i is set to the training accuracy of the corresponding classifier on instances having class label l. If the classifier has poor classification performance, then there may exist a lot of noise in the classification result, which means that the additional feature introduced by label i may not be very helpful for learning l, and therefore it should be assigned with lower weight. In the testing phase, each test example is first classified by all the BRs in the ensemble. Then, the final label is determined by combining the classification results from all the binary classifiers in dynamic weighted voting ensemble approach. In static weighted voting ensemble approach, the weight of each classifier in the ensemble is derived from the most up-to-date chunk. However, different testing instances are associated with different classification difficulties. So, using a global setting of ensemble weights in the ensemble to predict a certain instance may lead to incorrect classification of the example, because the appropriate ensemble weighting for that instance might be different.

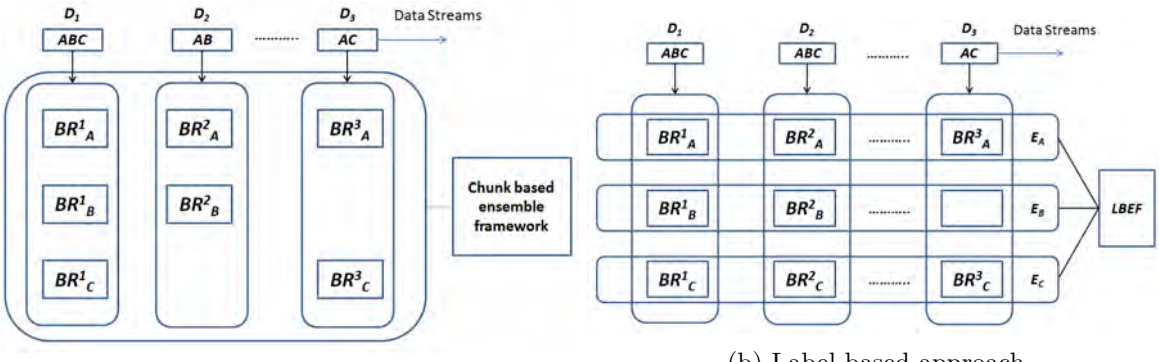
The dynamic streaming random forests (DSRF) [17] algorithm is a self-adjusting stream classification algorithm. The Random Forests algorithm [18] is an ensemble based classification technique developed by Breiman. It grows a number of binary decision trees and the classification for each new record is the plurality of the votes from the trees. It uses the Gini index [19] for selecting split attributes. Streaming Random Forests [18] is a stream-classification algorithm that builds streaming decision trees with the techniques from Breimans Random Forests. Its classification accuracies are approximately equal to those of Random Forests. The Streaming Random Forests algorithm grows binary decision trees each from a different block of data. The trees are grown using the Hoeffding bounds to decide when to stop building on each node. The algorithm selects the attribute to split us-

ing the Gini index. It takes two parameters, namely the number of trees to be built and the number of records used to grow each tree (*tree window*). The Dynamic Streaming Random Forests algorithm is a self-adjusting stream classification algorithm that is able to reflect concept changes. The algorithm, like the basic Streaming Random Forests algorithm, initially grows a defined number of trees. The difference is that the *tree window* is not constant for all trees. Instead, whenever a certain number ($tree_{min}$) of records have contributed to building a tree, the current classification error of the tree is calculated. If the error is greater than a threshold, then the algorithm stops building this tree and switches to building the next tree. This threshold ensures that none of the trees performs worse than a random tree. If error is greater than the threshold, the algorithm continues building the current tree using half the previous number of records, before it enters another test phase. Each tree has, therefore, a different *tree window* that is never greater than $2 * tree_{min}$. Once the total number of trees have been grown, the algorithm enters a test phase where the classification accuracy for the forest is calculated using previously unseen labeled data records. The classification error for each individual tree is also calculated and used to assign a weight to the tree. In addition, the algorithm derives new values of the parameters to use in the subsequent building phase.

DSRF uses an entropy-based drift-detection technique to handle concept drift and deploys a stacked binary relevance model to handle label correlations among multiple labels. The authors claim that their method is able to handle both labeled and unlabeled data. However, the work presented in [17] does not discuss how the classification and subsequent labeling process of unlabeled data is to be performed. It is applied on only on synthetic dataset with mixed performance.

Label based approaches

Label based ensemble framework ($LBEF$) [7] is a framework for classification of multi-label data streams. The chunk based ensemble frameworks are shown in Figure 3.3. In these ensembles, a set of binary classifiers built from the labeled examples are taken as base classifiers. These classifiers share the same weight. These chunk-based ensembles have some limitations for mining multi-label streams. If a class label l is not observed in a data chunk, it is not possible to build a binary classifier with respect to label l. Moreover, a problem of binary classifiers methods in data streams is that class-label imbalance may become exacerbated by large numbers of training examples. $LBEF$ (Figure 1.3b) addresses these two limitations. In $LBEF$, base classifiers are built with respect to class labels, instead of data chunks. The following example explains the label based ensemble framework:



(a) Chunk based approach

(b) Label based approach

Figure 1.3: Batch Incremental Methods.

In Figure 1.3a, data chunk D_n has three class labels $A; B; C$. Thus, three new binary classifiers can be built and added into the ensemble. For example, a new binary classifier with respect to the label A can be constructed and then added into the ensemble. When the same operation is applied to the other two class labels, we can observe that, for data chunk D_{n-k+1}, label B is missed, and thus corresponding binary classifier cannot be built. In Figure 1.3b, it can be seen that $LBEF$ learns one new binary classifier per label per chunk and then adds to the label's ensemble. As a result absence of class's instance does not lead to replacement of that class's classifier.

$LBEF$ is able to deal with the class imbalance problem [7]. $LBEF$ exhibits better performance than the baseline multi-label classification methods while reducing manual labeling cost. To reduce manual labeling cost, $LBEF$ uses vote entropy [20] to select the most informative instances from the stream to be labeled by human experts. Ensemble uses a bunch of weak classifiers to construct a strong classifier. The impact of a base classifier in the ensemble is proportional to its weight. A classifier having higher prediction accuracy will be assigned a heavier weight. The active learning is to dynamically adjust the weight vector of each base classifier based on its prediction accuracy on data streams. Obviously, stream records to which all the base classifiers in an ensemble assigns the same label does not provide useful information for identifying an accurate classification boundary. Thus, the uncertainty of an example with respect to different base classifiers can be used to adjust the weights of an classifier. This is equivalent to calculating the vote entropy. A recent work [21] also shows that the larger the vote entropy an example has, the more information it carries to adjust the weight vector of the ensemble. After adding informative examples, the decrease of expected loss of the ensemble classifier is proportional to its vote entropy. $LBEF$ handles concept drifting by adjusting

ensemble weight which is determined by coordinate wise gradient descent method [22]. As the concept evolves with time, the prediction accuracy of all base classifiers may drop significantly. $LBEF$ selects examples to capture the concept in the current chunk in the stream. Version space denotes the area containing all latent class boundaries, which can be used to measure the degree of uncertainty of an ensemble model. Both the vote entropy and the version space decrease can be considered as the reduction of uncertainty of class boundary. Thus, the ensemble classifiers can reduce uncertainty by selecting examples for which both vote entropy and the version space decrease.

$LBEF$ deals with concept drifting by adjusting ensemble weight which is determined by coordinate wise gradient descent method [22]. In weighted ensemble mechanism, long absence of a class's instance may lead to corresponding ensemble weight becoming very small. So, a class ensemble's vote will become inconsequential during the classification of an instance. As a result, an existing class's instance reappearing after a long time may be classified to an incorrect class label. On the other hand, instances on which the classifiers have the most disagreement, have larger vote entropy and hence, considered as most informative examples. But when a new class's instance x appears in the stream, all the binary classifiers will fail to classify it. As a result, according to the definition given in [17], x's vote entropy will be 0. So, x will not be chosen for manual labeling. Thus $LBEF$ fails to detect concept evolution. Moreover, $LBEF$ assumes the label set of the stream to be fixed. So, even if a novel class instance is selected for manual labeling, there is no mechanism for adding new label's ensemble in the framework.

1.1.3 Problems of the existing methods

The shortcomings of the existing multi-label data stream classification methods can be summarized as follows:

- None of the exiting multi-label classification methods address the issues of concept evolution. So, any novel class instance appearing in the stream is assigned to an already existing set of class labels which may significantly degrade classifier's performance.

- The issue of class recurrence is still unaddressed in the context of multi-label data streams. As the data stream classifiers are frequently updated with the latest data, whenever a class's instances reappear in the stream after a long interval, the classifiers are trained again with the recurring data which leads to redundant computation.

- Most of the methods except DSRF [17] and $LBEF$ [7], consider the data stream to be labeled. As

these are supervised learning methods, they have very little applicability in practical systems. Although, [17] proposes an active leaning framework for labeling unlabeled data, [6] has not discussed any such framework.

- As the amount of labeled data is scarce in real world streaming scenarios, classifiers trained with such limited amount of labeled data tend to exhibit poor performance. The problem of scarcity of labeled data in the stream is not considered in any of the existing methods.
- Apart from [14], none of the existing methods have presented an exhaustive experimental evaluation. Most of the methods have shown experimental results on 2-3 datasets.

1.2 Objective of the thesis

Motivated by the challenges of the multi-label data stream classification, we propose a layered ensemble based classification framework ($LEAD$). Our proposed framework maintains a two layered ensemble architecture, G. The two layers are called the top layer ensemble, T_{Lr} and the bottom layer ensemble, B_{Lr}. Both T_{Lr} and B_{Lr} contain label based ensembles (E_ls) [7] for different class labels (ls) in S. T_{Lr} reflects the concept present in the most recent chunk, i.e, an E_l is kept in T_{Lr} only if instances having the class label l were encountered in the most recently processed chunk. On the other hand, B_{Lr} reflects the old concept, i.e, the concept that is not present in the most recent chunk. Binary relevance classifier [15] is used as the base multi-label classifier. The major contributions of our framework are as follows:

1. To deal with the scarcity of labeled data, we have proposed a deferred classification mechanism. When classification of any unlabeled instance is failed, its classification is deferred. In the mean time, more labeled data may appear in the stream and contribute to the construction an updated classifier.
2. To address the issue of concept evolution, we have incorporated a novel class detector in the form of a fuzzy c-means clustering model. The novelty detector detects the emergence of a set of novel class labels in the multi-label data stream.
3. We have proposed a two layer (top and bottom layer) ensemble framework. The most recently encountered class's ensembles are maintained in the top layer ensemble. The bottom layer

ensemble is maintained to detect the appearance instances of recurrent class or class labels. The layered approach also helps in separating recurrent class instances from novel class instances. The layered ensemble framework is discussed in detail in Chapter 3.

4. We have conducted extensive experiments on datasets from different domains to validate the performance of our framework. We have also proposed two new experimental measures for evaluating the performance of a multi-label classification framework in the presence of concept evolution.

1.3 Outline of the thesis

The rest of the thesis is organized as follows.

In chapter 2, we present the basic components necessary to understand the idea presented in this thesis. We start with the discussion of traditional multi-label classification models. Then we briefly describe the challenges associated with data stream classification. Next, there is a discussion on how the traditional classification models can be modified to address the challenges of multi-label data stream. Then we present the role of clustering in novel concept detection in data streams and analyze an existing novel concept detection technique. Finally, we present the necessity of fuzzy clustering in the context of multi-label data and also discuss the existing fuzzy cluster validity measures.

Chapter 3 discusses the major contribution of this thesis. Our $LEAD$ framework and the phases of $LEAD$ are thoroughly discussed with their specific purposes.

In chapter 4, the experimental results are presented. The description of the datasets used in this thesis are presented at first. Then we discuss the formulation of the baseline methods to be compared with $LEAD$. Since, $LEAD$ framework is the first of its kind, we compare its performance with two baseline methods, each generated by combining a novel concept detection technique with a multi-label stream classification technique. It is followed by the discussion of the various performance metrics and their formulation. In order to establish the fact that $LEAD$ provides better result than the baseline methods we also perform statistical tests here.

Finally, in chapter 5, we conclude the thesis by highlighting some future research directions. We discuss how the proposed $LEAD$ framework can be modified and extended further to attain better performance. Also, some possible stream environments where $LEAD$ can be applied is also discussed.

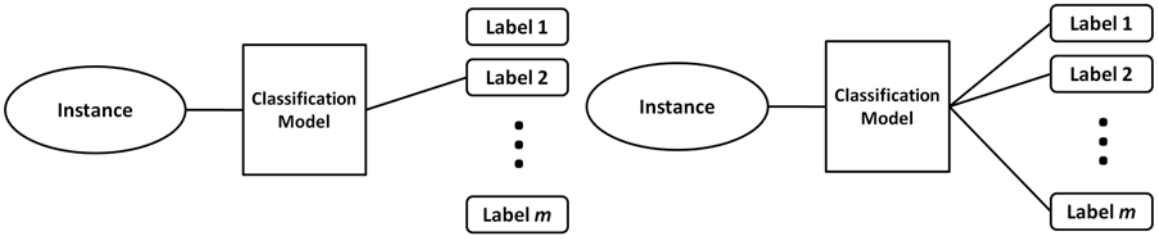
Chapter 2

Background

This chapter begins with a comparison between multi-label and multi-class learning. Then we briefly introduce the traditional multi-label classification methods, followed by an analysis of the challenges of multi-label data stream classification. The application of ensemble based techniques for performing multi-label data stream classification is then presented. Finally, we explore the fuzzy clustering techniques for novel concept detection in data streams.

2.1 Multi-class Vs Multi-label Classification

The problem of single-label classification is concerned with learning from instances, where each instance is associated with a single label l from a finite set of disjoint labels L, where $|L| > 1$. If there are more than two class labels in the label set L, then the learning problem is referred to as multi-class classification. On the other hand, the task of learning a mapping from an instance to a set of labels is referred to as a multi-label classification. In contrast to multi-class classification, alternatives in



(a) Multi-class classification (b) Multi-label classification

Figure 2.1: Multi-class Vs Multi-label classification

Table 2.1: Example of multi-class classification

Article Id	Accurate	Important	Informative	Type
1	Yes	Yes	Yes	Best
2	No	Yes	Yes	Noteworthy
3	Yes	No	No	Negligible

multi-label classification are not assumed to be mutually exclusive: multiple labels may be associated with a single example, i.e., each example can be a member of more than one class.

Figure 2.1 depicts the difference between the two classification problems. To better understand the multi-class classification problem, let us consider the example in Table 2.1. Suppose, a newspaper categorizes the news articles it publishes daily in any one of the three categories: Best, Noteworthy and Negligible. The attributes for determining the category of a news article are whether it is: accurate, important and informative. No article can be assigned to multiple categories at once. So, any classification model that will be used to classify the news articles will be considered as a multi-class classification model.

The issue of learning from multi-label data has recently attracted significant attention from many researchers, motivated by an increasing number of new applications which include semantic annotation of images and video (news clips, movies clips), functional genomics (gene and protein function), music categorization into emotions, text classification (news articles, web pages, patents, e-mails, bookmarks etc.), directed marketing and others. For example, a text document that talks about scientific contributions in medical science can belong to both science and health category, genes may have multiple functionalities (e.g. diseases) causing them to be associated with multiple classes, an image that captures a field and fall colored trees can belong to both field and fall foliage categories, a movie can simultaneously belong to action, crime, thriller, and drama categories, an email message can be tagged as both work and research project; such examples are numerous. Traditional binary and multi-class problems both can be posed as specific cases of multi-label problem. However, the generality of multi-label problems makes it more difficult than the others [9].

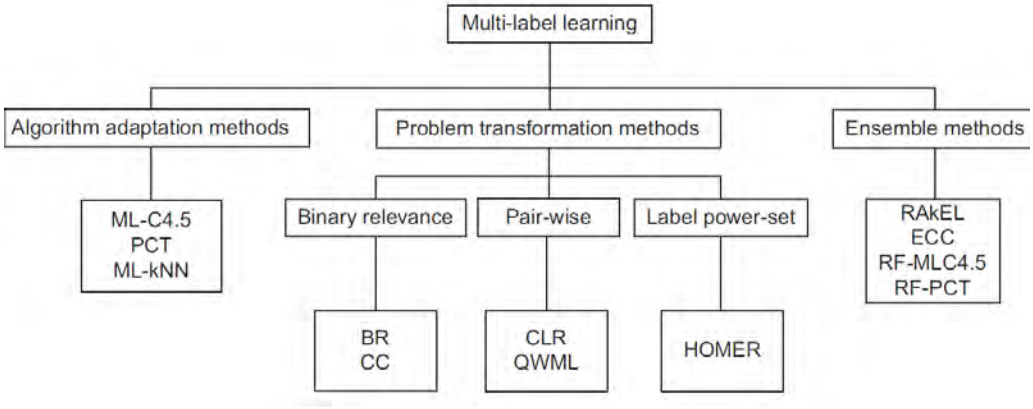


Figure 2.2: Classification of multi-Label learning methods

2.2 Methods for multi-label learning

In this section, we briefly introduce the state-of-the-art methods for multi-label learning. As shown in Figure 2.2, the existing methods in literature can be divided in three general categories: algorithm adaptation, problem transformation and ensemble methods.

2.2.1 Algorithm adaptation methods

The multi-label methods that adapt, extend and customize an existing machine learning algorithm for the task of multi-label learning are called algorithm adaptation methods. Here, we present multi-label methods proposed in the literature that are based on the following machine learning algorithms: k-Nearest neighbors, decision trees and neural networks. The extended methods are able to directly handle multi-label data.

k-Nearest neighbors

Several variants for multi-label learning (ML-kNN) of the popular k-Nearest Neighbors lazy learning algorithm have been proposed [9, 23]. The retrieval of the k-nearest neighbors is the same as in the traditional kNN algorithm. The main difference is the determination of the label set of a test example. Typically, these algorithms use prior and posterior probabilities of each label within the k-nearest neighbors. First, for each test example, its k-nearest neighbors in the training set are identified. Then, according to statistical information gained from the label sets of these neighboring examples, i.e., the number of neighboring examples belonging to each possible label, the maximum a

posteriori principle is used to determine the label set for the test example.

Decision trees

Clare et al. [12] adapted the C4.5 algorithm for multi-label data (ML-C4.5) by modifying the formula for calculating entropy. It allows multiple labels in the leaves of the tree. The modified entropy sums the entropies for each individual class label. Blockeel et al. [24] proposed the concept of predictive clustering trees (PCTs): the top-node corresponds to one cluster containing all data, which is recursively partitioned into smaller clusters while moving down the tree. PCTs are constructed using a standard top-down induction of decision trees algorithm, where the variance and the prototype function can be instantiated according to the task at hand. PCTs have been used for predicting tuples of variables, predicting time series and predicting classes organized into a hierarchy or a directed acyclic graph. However, they can also be used in the context of multi-label learning, where each label is a component of the target tuple.

Neural networks

Neural networks have also been adapted for multi-label classification [25]–[26]. BP-MLL [25] is an adaptation of the popular back-propagation algorithm for multi-label learning. The main modification to the algorithm is the introduction of a new error function that takes multiple labels into account. Another neural network based multi-label learning algorithm named Ml-RBF [26] is proposed, which is derived from the traditional radial basis function (RBF) methods. Brieflyy, the first layer of an Ml-RBF neural network is formed by conducting clustering analysis on instances of each possible class, where the centroid of each clustered groups is regarded as the prototype vector of a basis function. After that, second layer weights of the Ml-RBF neural network are learned by minimizing a sum-of-squares error function. Furthermore, Ml-RBF significantly outperforms Bp-MLL in terms of both effectiveness and efficiency.

2.2.2 Problem transformation methods

The problem transformation methods are multi-label learning methods that transform the multi-label learning problem into one or more single-label classification or regression problems. For smaller single-label problems, there exists a plethora of machine learning algorithms. Problem transforma-

tion methods can be grouped into three categories: binary relevance, label power-set and pair-wise methods.

Another problem transformation method is the label combination method, or label power-set method (LP). The basis of these methods is to combine entire label sets into atomic (single) labels to form a single-label problem (i.e., single-class classification problem). For the single-label problem, the set of possible single labels represents all distinct label subsets from the original multi-label representation. In this way, LP based methods directly take into account the label correlations. Another label power-set method is [27], which first constructs a hierarchy of the multiple labels and then constructs a classifier for the label sets in each node of the hierarchy.

Binary relevance methods

The simplest strategy for problem transformation is to use the one-against-all strategy to convert the multi-label problem into several binary classification problems. This approach is known as the binary relevance method (BR) [15]. A method closely related to the BR method is the Classifier Chain method (CC) proposed by Read et al. [28]. This method involves L binary classifiers linked along a chain. Godbole et al. [16] present algorithms which extend the binary classifiers along two dimensions: training set extension and improvement of margin. With the first approach, the training set is extended with the predictions of the binary classifiers and then a new set of binary classifiers is trained on the extended dataset. For the second extension, very similar negative training examples and the negative training examples of a complete class that are similar to the positive class are removed.

Label power-set methods

A second problem transformation method is the label combination method, or label power-set method (LP) [29]. The basis of these methods is to combine entire label sets into atomic (single) labels to form a single-label problem (i.e., single-class classification problem). For the single-label problem, the set of possible single labels represents all distinct label subsets from the original multi-label representation. In this way, LP based methods directly take into account the label correlations. However, the space of possible label subsets can be very large. To resolve this issue, Read has developed a pruned problem transformation method called Pruned Set (PS) [13], that selects only the transformed labels that occur more than a predefined number of times. Another label power-set method is HOMER [27], which first constructs a hierarchy of the multiple labels and then constructs a classifier for the label

sets in each node of the hierarchy.

Pair-wise methods

A third problem transformation approach to solving the multi-label learning problem is pair-wise or round robin classification with binary classifiers [30]. The basic idea here is to use $\frac{L(L-1)}{2}$ classifiers covering all pairs of labels. Each classifier is trained using the samples of the first label as positive examples and the samples of the second label as negative examples. To combine these classifiers, the pairwise classification method naturally adopts the majority voting algorithm. Given a test example, each classifier predicts (i.e., votes for) one of the two labels. After the evaluation of all $\frac{L(L-1)}{2}$ classifiers, the labels are ordered according to their sum of votes. A label ranking algorithm is then used to predict the relevant labels for each example. Besides majority voting in CLR, Park et al. [31] propose a more effective voting algorithm. It computes the class with the highest accumulated voting mass, while avoiding the evaluation of all possible pairwise classifiers. Mencia et al. [32] adapted the L Weighted approach to multi-label learning (QWML).

2.2.3 Ensemble methods

The ensemble methods for multi-label learning are developed on top of the common problem transformation or algorithm adaptation methods. The most well known problem transformation ensembles are the *RAKEL* system by Tsoumakas et al. [33], ensembles of pruned sets (EPS) [13] and ensembles of classifier chains (ECC) [28]. *RAKEL* constructs each base classifier by considering a small random subset of labels and learning a single-label classifier for the prediction of each element in the power-set of this subset. EPS uses pruning to reduce the computational complexity of label power-set methods, and an example duplication method to reduce the error rate as compared to label power-set and other methods. This method proved to be particularly competitive in terms of efficiency. ECC are ensemble methods that have classifier chains (CC) as base classifiers. The final prediction is obtained by summing the predictions by label and then applying threshold for selecting the relevant labels. Note that binary methods are occasionally referred to as ensemble methods because they involve multiple binary models. However, none of these models is multi-label itself and therefore we use the term ensemble strictly in the sense of an ensemble of multi-label methods. Algorithm adaptation ensemble methods are the ensembles whose base classifiers are themselves algorithm adaptation methods. An example of an algorithm adaptation ensemble method are the ensembles of predictive clustering trees

(PCTs) [34]. These ensembles use PCTs for predicting tuples of variables as base classifiers. Each base classifier makes a multi-label prediction and then these predictions are combined by using some voting scheme.

Each base classifier makes a multi-label prediction and then these predictions are combined to obtain the final classification decision. The combination of results can be done in two ways: majority voting and weighted ensemble voting. In majority voting, the class label to which an instance is classified by most of the classifiers in the ensemble is chosen as that instance's class label. In ensemble weighting methods, each classifier in the ensemble is assigned a weight. The most reliable classifier is assigned the highest weight, and the least accurate one is assigned the least weight. The final prediction Y_t (N predictors) is defined by:

$$Y_t = \frac{1}{N} \sum_{i=1}^{N} Y_{t_i} \tag{2.1}$$

where Y_{t_i} is the prediction made by the individual classifier. Another way is using relative performance(i.e. mean scaled error(MSE)) of each predictor [27], where the weight is specified by:

$$w_i = \frac{\frac{1}{MSE_i}}{\sum_{i=1}^{N} MSE_i} \tag{2.2}$$

In this weighted average, the high performance classifier will be given larger weight and vice versa. We have adopted the majority voting scheme in the $LEAD$ framework.

2.3 Data stream and its challenges

Data stream is a special form of data that has spawned several research issues in recent years. It is an infinite sequence of instances $\{x_1, x_2, \ldots, x_i, \ldots\}$, where each instance x_i is a d-dimensional feature vector. As explained in Chapter 1, each instance of a multi-label data stream can be associated with multiple class labels. In this section, we discuss the challenges that the stream classifiers have to deal with in data stream environment.

High speed nature of data streams

The inherent characteristic of data streams is its high speed. The algorithm should be able to adapt to the high speed nature of streaming information. The rate of building a classification model should be higher than the data rate. So, the amount of time available for classification is very limited compared to the traditional non-stream classification models.

Unbounded memory requirements

Classification techniques require data to be resident in memory for building the model. The huge amounts of data streams generated rapidly dictate the need for unbounded memory. This challenge has been addressed using load shedding, sampling, aggregation, and creating data

One pass learning

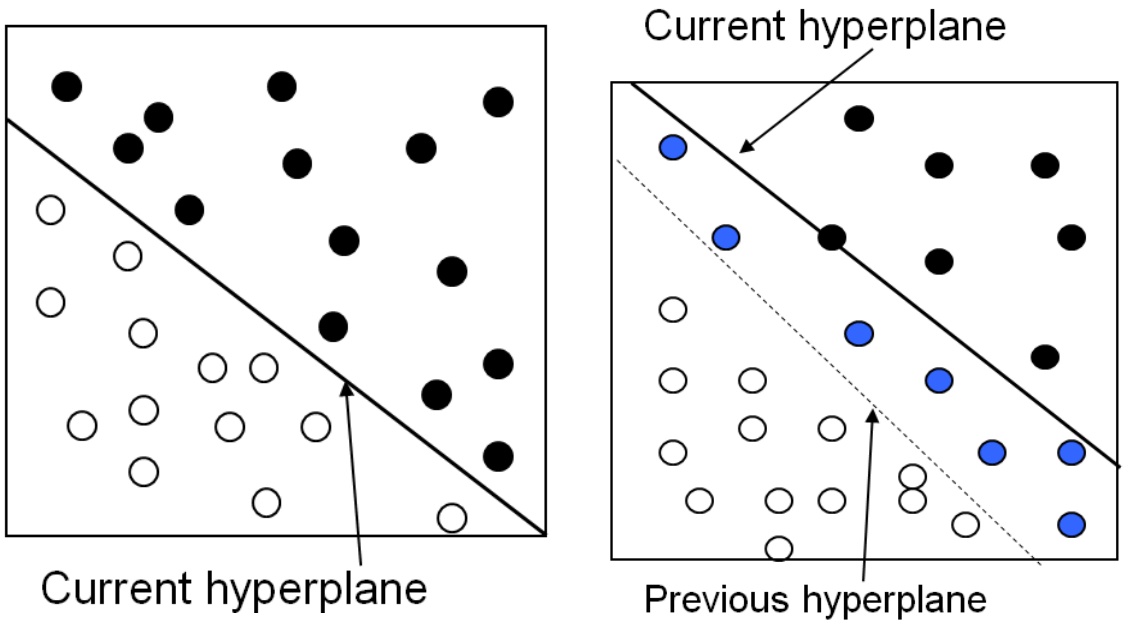
Data streams must be accessed in order and that can be read only once or a small number of times [35]. But the modern applications generate huge volume of streaming data. As discussed earlier, with the constraint of limited storage and faster processing time, data streams can be allowed to be scanned only once. So, the data stream mining models must posses the capability to learn the underlying nature of data in a single pass over the data.

Lack of labeled data

The amount of labeled data in the stream affects the quality of the learned model. Manual labeling of data is often costly and time consuming, so in a streaming environment, where data appear at a high speed, it is not always possible to manually label all the data as soon as they arrive. Thus, in practice, only a small fraction of the stream can be labeled by human experts. So, a stream classification algorithm will have very few instances to update its models in such circumstances, leading to poor classifiers.

Concept drifting

Concept drifts change the classifier results over time. This is because of the change in the underlying data patterns. This results in the model becoming stale and less relevant over time. The capture of



(a) A data chunk

(b) Presence of concept drift

Figure 2.3: Concept drift in data streams

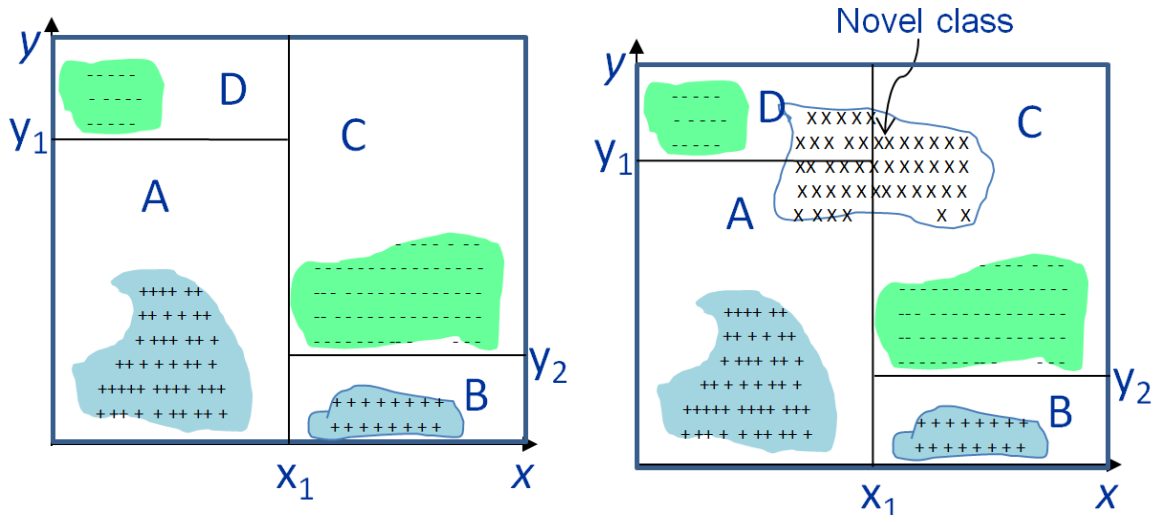
such changes would help in updating the classifier model effectively. The use of an outdated model could lead to very low classification accuracy.

Let us consider the example in Figure 2.3. As a new chunk arrives (Figure 2.3a), a new classifier is learned. The decision boundary is denoted by the straight line. The positive examples are represented by white circles while the negative examples are represented by dark circles. With the passage of time the concept of some of the examples may change. As shown in Figure 2.3b, due to concept drift some negative examples may have become positive. So, the previous decision boundary has become outdated and a new model has to be learned.

Concept evolution

Concept evolution occurs when a new class or set of classes emerge in the data stream. For example, #MH370 is currently an emerging trend in Twitter and the related posts can be associated with multiple tags like #rescue, #mystery etc.

Let us consider the example in Figure 2.3. As a new chunk arrives (Figure 2.3a), a new classifier is learned. The decision boundaries for different classes (A, B, D) are shown by the straight lines. With the passage of time some new examples belonging to a new class C may arrive that cannot be



(a) A data chunk

(b) Presence of concept drift

Figure 2.4: Concept evolution in data streams

classified by existing classifiers. So, the previous classifier has become outdated and a new classifier has to be learned.

Class recurrence

A special case of concept-evolution is that of a recurring class, which occurs when a class reappears after a long disappearance from the stream. This special case is also common because an intrusion in network traffic may reappear after a long time, or social network actors may discuss an interesting topic in Twitter at a particular time every year (e.g., Halloween).

2.4 Mining data streams

One of the goals of traditional data mining algorithms is to mine models from large databases with bounded memory. Traditional data mining algorithms require multiple scans of the training data makes them inappropriate in the streaming data environment where the examples are coming in at a higher rate than they can be repeatedly analyzed. One option for mining data streams is Incremental or online data mining methods [36]–[37]. These methods continuously revise and refine a model by incorporating new data as they arrive. However, in order to guarantee that the model trained incrementally is identical to the model trained in the batch mode, most online algorithms rely on a costly model updating procedure, which sometimes makes the learning even slower than it is in

batch mode. Recently, an efficient incremental decision tree algorithm called VFDT is introduced by Domingos et al [11]. For streams made up of discrete type of data, Hoeffding bounds guarantee that the output model of VFDT is asymptotically nearly identical to that of a batch learner.

The above mentioned incremental and on-line methods such as VFDT, all produce a single model that represents the entire data stream. It suffers in prediction accuracy in the presence of concept drifts. This is because the streaming data are not generated by a stationary stochastic process; indeed, the future examples we need to classify may have a very different distribution from the historical data. In order to make time-critical predictions, the model learned from the streaming data must be able to capture up-to-date trends and transient patterns in the stream. The idea is to revise the model by incorporating new examples and also eliminate the effects of examples representing outdated concepts. The ensemble approach offers this capability. Instead of continuously revising a single model, an ensemble of classifiers is trained. Another benefit of the ensemble approach is its efficiency and ease-of-use.

2.4.1 Ensemble based learning of data streams

Ensemble learning is a commonly used tool for building prediction models from data streams, due to its intrinsic merits of handling large volumes stream data. Different from traditional incremental and online learning approaches that merely rely on a single model [11, 38], ensemble learning employs a divide-and-conquer approach to first split the continuous data streams into small data chunks, and then build light-weight base classifiers from the small chunks. At the final stage, all base classifiers are combined together for prediction. By doing so, an ensemble model can enjoy a number of advantages, such as scaling up to large volumes of stream data, adapting quickly to new concepts, achieving lower variances than a single model, and easily to be parallelized.

Ensemble classifiers on data streams provide a generic framework for handling massive volume data streams with concept drifting. The idea of ensemble classifiers is to partition continuous data streams into small data chunks, from which a number of base classifiers are built and combined together for prediction. Two main motivations for combining classifiers are as follows:

- **Statistical (or worst case) motivation:** It is possible to avoid the worst classifier by averaging several classifiers. It was confirmed theoretically by Fumera and Roli in [39]. This simple combination was demonstrated to be efficient in many applications. There is no guarantee,

however, the combination will perform better than the best classifier.

- **Representational (or best case) motivation:** Under particular situations, fusion of multiple classifiers can improve the performance of the best individual classifier. It happens when the optimal classifier for a problem is outside of the considered "classifier space". There are many experimental evidences that it is possible if the classifiers in an ensemble make different errors. This assumption has a theoretical support in some cases when linear combination is performed.

2.5 Cluster-based novel concept detection in data streams

In this section, we first discuss a popular cluster based novel concept detection technique for the single label data streams. Then we explain how clustering can deployed in multi-label data stream scenario. Finally, we present some validity measures for fuzzy clustering.

2.5.1 Concept evolution and clustering

Detecting novel concept is a unsupervised process. As the classification model has no prior knowledge of the novel class, the best way to make sense of concept evolution is to identify hidden patterns in the data. Hence, clustering is the most suitable approach for novel concept detection. A popular clustering based novel concept detection technique is online novelty and drift detection algorithm (OLINDDA) [40].

An overview of OLINDDA is shown in Figure 2.5. The proposed approach relies on three hypersphere-based models to store knowledge about (1) the normal profile, (2) concepts that extend the normal profile and (3) novel concepts. The normal model is the only static one, remaining as a reference to the initial learning phase. It corresponds to what is usually employed by most novelty detection techniques. The extension and novelty models can be created and continuously updated. Once newly discovered concepts become part of these two models, they will also help to explain future examples. Since this structure is naturally incremental, the algorithm is able to start with a basic description and extend it, weakening the requirement of a comprehensive initial labeled dataset, that may not always be available. Furthermore, since these models are composed basically of the coordinates of centroids and respective radii, besides a few other measures and statistics, model updating is fast, which is essential when working with data streams.

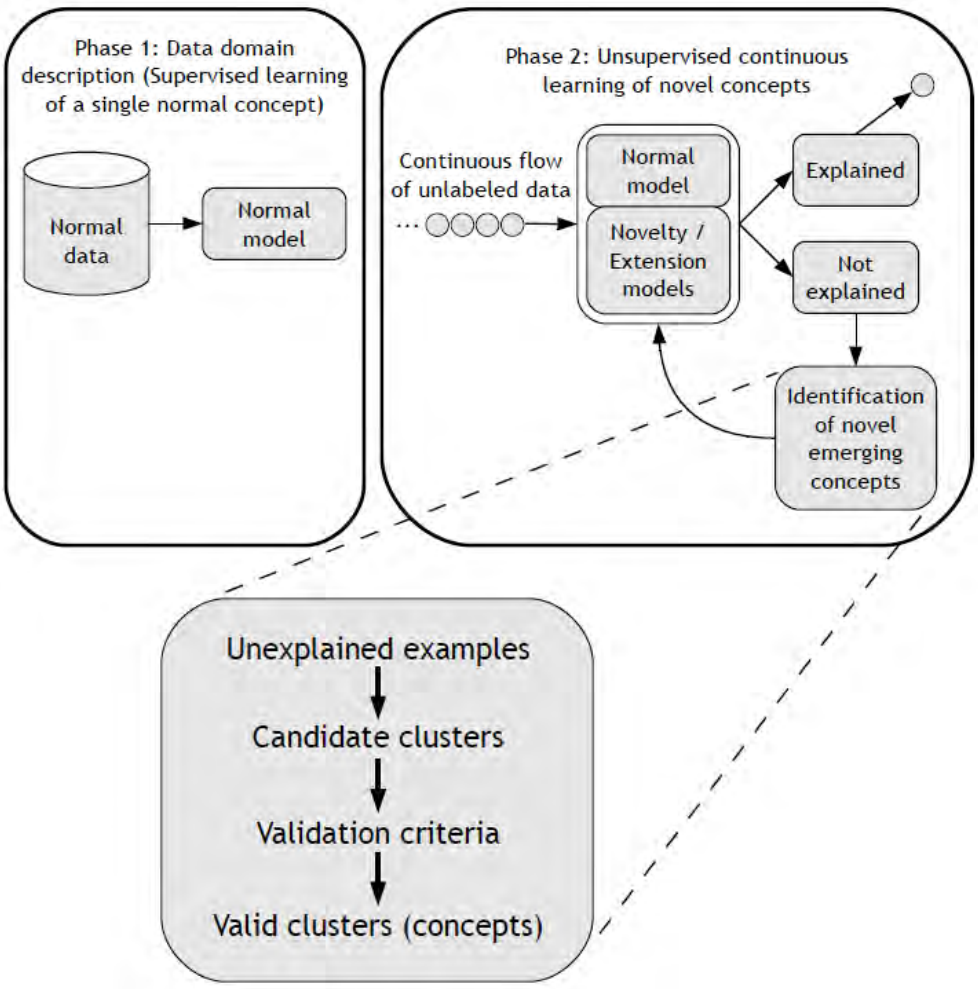


Figure 2.5: Overview of OLINDDA

Phase 1: Supervised learning of the normal concept

The proposed technique starts by modeling the normal or expected behavior in the domain under investigation, described by a set of normal examples. To build the normal model, k clusters are produced using, typically, the k-means clustering algorithm. The normal model is composed of k hyperspheres, built in feature space, obtained directly from the clusters and represented by their centers and radii. Each hypersphere center is the centroid of a cluster, and its radius is the Euclidean distance from the centroid to the farthest example of the respective cluster.

Phase 2: Unsupervised continuous learn-ing of novel concepts

For each new example, the algorithm first checks if it can be explained by the knowledge acquired until that point, represented by (up to) three models, as previously described. If the coordinates of the example lie inside a hypersphere of any of the existing models, it is considered as explained by the corresponding model. The corresponding statistics are the updated, and the example is discarded. Otherwise, the example is marked as a member of an unknown profile and moved to a short-term memory for further analysis. That memory works like a FIFO queue to avoid its uncontrolled growth. The maximum number of examples in this memory can be modified. The steps of this phase are as follows:

- **Cluster validation:** Novel concepts are initially identified as clusters of examples previously considered unknown that comply with certain restrictions. In order to discover these clusters, each time a new unknown example is found, k candidate clusters are generated from the examples currently available at the short-term memory of unknown profiles. Candidate clusters are then evaluated to determine if any presents enough evidence of the appearance of a new concept. This is not a trivial task, since it is a totally unsupervised process. On the other hand, the fact that no labels are required allows its application to a large amount of data that could not be manually classified.

- **Distinction between extension and novelty:** Once a cluster is considered valid, the algorithm analyzes its similarity to the normal concept. An extension should naturally be located in the vicinity of the region associated with the normal concept, while a concept very dissimilar to normal should be considered novelty. This notion of vicinity is defined by a hypersphere centered at the centroid of the centroids of the normal model, whose radius is the distance to the farthest centroid. If the centroid of the new cluster is located inside this hypersphere, the new concept is labeled extension. Otherwise, it is considered novelty.

- **Merging of concepts:** A new valid cluster may itself represent a new concept. However, as learning progresses, a concept may be more adequately described by a set of clusters. For this reason, OLINDDA evaluates the similarity between newly discovered concepts and existing concepts of the same model. It does that by checking if the new valid cluster intercepts any of the previous clusters. If it does, they are grouped under the same label and their statistics

are merged. If it does not, the cluster is considered a new con-cept on its own and receives a new label. A single cluster may trigger a sequence of mergers. This process tends to produce a smaller number of concepts (labels), that are usually easier to analyze, directing the algorithm toward the final goal of producing a class structure as similar to the real one as possible.

- **Dynamic adaptation of the number of clusters:** The number of clusters k is an intrinsic parameter of the k-means clustering algorithm, used (1) to create the initial normal model and (2) to periodically generate candidate clusters in the online phase. In the initial model, k is defined by a parameter, since it depends on the data distribution. For the generation of candidate clusters in the online phase, however, k is dynamically adapted to optimize the chance of discovering a valid cluster. The automatic adaptation of k takes place after each iteration in which candidate clusters have been generated. If at least one candidate cluster is considered valid, the value of k is maintained. Otherwise, the algorithm checks what prevented each cluster from being accepted: sparseness or lack of examples. Then, considering the most frequent cause of failure for all candidate clusters, it decides how to adapt k. After a few iterations, k tends to stabilize around the optimum value that generates valid clusters.

2.5.2 Fuzzy clustering and data streams

In multi-label data stream, one record may belong to multiple classes. Hence, the hard clustering methods like k-means clustering cannot be applied for novelty detection in multi-label data stream scenario. Although, some fuzzy data stream clustering methods exist in literature, novelty detection is an unexplored issue in multi-label data streams. In fuzzy clustering, every data point has a degree of associativity to clusters, rather than belonging completely to just one cluster. All the fuzzy data stream clustering algorithms in literature are based on objective functions. Most of these algorithms are based on the "Fuzzy C-means Clustering" (FCM) algorithm [41] which applies fuzzy partitioning logic. Possibilistic partitioning [42], an unconstrained fuzzy partition logic, has also been used in literature. So, the fuzzy logic based data stream clustering algorithms can be classified as either fuzzy methods or possibilistic methods.

Fuzzy C-means clustering

In fuzzy clustering, every point has a degree of belonging to clusters, as in fuzzy logic, rather than belonging completely to just one cluster. Thus, points on the edge of a cluster may be in the cluster to a lesser degree than points in the center of cluster. Fuzzy C-means Clustering a very popular fuzzy clustering algorithm. It is based on minimization of the following objective function:

$$J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^{m} ||x_i - c_j||^2 \tag{2.3}$$

Here, N is the number of candidate novel instances, u_{ij} is the degree of membership of x_i in the cluster j, x_i is the i-th of d-dimensional measured data, c_j is the d-dimension center of the cluster, and $||*||$ is any norm expressing the similarity between any measured data and the center. The algorithm takes three parameters. They are: C, m, and ϵ. C denotes the number of clusters to be created. m is the fuzziness co-efficient or fuzzifier. The fuzzifier m determines the level of cluster fuzziness. A large m results in fuzzier clusters. If $m = 1$, the cluster memberships converges to 0 or 1, which implies a crisp partitioning. In the absence of experimentation or domain knowledge, m is commonly set to 2. ϵ is the termination criterion where $0 < \epsilon < 1$.

The fuzzy C-means algorithm is very similar to the k-means algorithm. The steps are as follows:

1. Choose a number of clusters C
2. Assign randomly to each point coefficients for being in the clusters
3. Repeat until the algorithm has converged (that is, the coefficients' change between two iterations is no more than ϵ
4. Compute the centroid for each cluster, using the formula above.
5. For each point, compute its coefficients of being in the clusters, using the formula above.

The algorithm minimizes intra-cluster variance as well, but has the same problems as k-means; the minimum is a local minimum, and the results depend on the initial choice of weights.

2.5.3 Estimating fuzzy clustering validity

Clustering aims at detecting natural groups (or clusters) in multidimensional data sets. The principle is that data points within a cluster are as similar as possible whereas data points of different clusters

are as dissimilar as possible. Since clusters may have different shapes and sizes, a partition resulting from this unsupervised classification process needs to be validated. In this section, we discuss several validity measures for fuzzy clustering models.

Partition co-efficient

In fuzzy clustering, a partition coefficient F was initially designed by Bezdek [43]. The coefficient measures the amount of overlap between fuzzy clusters. Its disadvantages are the lack of direct connection to a geometric property and its monotonic decreasing tendency with the number of clusters C. F is given by-

$$F = \frac{1}{n} \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^2 \tag{2.4}$$

In this form F is inversely proportional to the overall average overlap between pairs of fuzzy subsets. A value of $F = 1$ corresponds to hard clustering.

Fuzzy within-cluster sum of square

Fuzzy within-cluster sum of square (fWCSS) [44] is applicable for correlation-based FCM algorithm. It has been designed by introducing fuzzy membership into the conventional measure within-cluster sum of square (WCSS). This new measure is

$$fWCSS(k) = \sum_{j=1}^{k} D_j \tag{2.5}$$

where D_j is given by

$$D_j = \sum_{i} u_{ij}^m d(x_i, C_j) = \sum_{i} u_{ij}^m \lfloor 1 - corr(x_i, C_j)^2 \rfloor \tag{2.6}$$

The term $\lfloor 1 - corr(x_i, C_j)^2 \rfloor$ is a Pearson correlation based distance metrics, which is the same one used in the algorithm. The $fWCSS(k)$ measures the overall compactness of fuzzy clusters, and the amount of overlap between fuzzy clusters with the inclusion of fuzzy membership. It measure uses the same distance metrics being used in the correlation-based FCM algorithm.

Compact and separate fuzzy validity criterion

The fuzzy validity criterion S [45] measures the overall average compactness and separation of a fuzzy c-partition. In this thesis, we will denote the criterion as W. W can be explicitly written as

$$W = \frac{\sum_{i=1}^{N}\sum_{j=1}^{C} u_{ij}^2 ||x_i - c_j||^2}{N \min_{i,j} ||c_i - c_j||^2} \tag{2.7}$$

A larger W indicates that all clusters are separated. On the other hand, a smaller W indicates that all the clusters in the partition are overall compact and separate to each other. W is independent of the underlying fuzzy clustering algorithm. For fuzzy c-means algorithm, W can be applied as follows:

1. Initialize $C \leftarrow 2$, $S^* \leftarrow \infty$, $C^* \leftarrow 1$, $c_{max} \leftarrow$ based on some heuristic;
2. Initialize fuzzy membership u_{ij};
3. Apply FCM to obtain c_j and u_{ij};
4. Do convergence test. If negative, go to 3;
5. Compute S;
6. If $S < S^*$, then $S \leftarrow S^*$ and $C \leftarrow C^*$;
7. If optimal candidate not found, go to 2;
8. $c = c + 1$. If $c = c_{max}$, stop;
9. Go to 2.

W is monotonically decreasing when C is large and close to N. Several heuristics exist in literature to identify the stop value c_{max} for the clustering model.

2.6 Statistical tests for comparing algorithms

In this section, we discuss the statistical tests that have been conducted in the thesis to compare the competing methods. An analysis of the significance of the tests is also presented. We have used three statistical tests. They are: Wilcoxon signed-rank test [46], Iman-Davenport test [47] and Holm's post hoc test [48].

2.6.1 Wilcoxon signed-rank test

The Wilcoxon signed ranks test is used for answering the following question: do two samples represent two different populations? It is a nonparametric procedure employed in hypothesis testing situations, involving a design with two samples. This is analogous to the paired t-test in nonparametric statistical procedures; therefore, it is a pairwise test that aims to detect significant differences between two sample means, that is, the behavior of two algorithms.

Wilcoxons test is defined as follows. Let d_i be the difference between the performance scores of the two algorithms on i-th out of n problems (if these performance scores are known to be represented in different ranges, they can be normalized to the interval [0, 1], in order to not prioritize any problem. The differences are ranked according to their absolute values; in case of ties, the practitioner can do any of the following: ignore ties, assign the highest rank, compute all the possible assignments and average the results obtained in every application of the test.

Let R^+ be the sum of ranks for the problems in which the first algorithm outperformed the second, and R the sum of ranks for the opposite. Ranks of $d_i = 0$ are split evenly among the sums; if there is an odd number of them, one is ignored:

$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \tag{2.8}$$

$$R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \tag{2.9}$$

Let T be the smaller of the sums, $T = \min(R^+, R)$. If T is less than or equal to the value of the distribution of Wilcoxon for n degrees of freedom, the null hypothesis of equality of means is rejected; this will mean that a given algorithm outperforms the other one, with the p-value associated. The table values are available at any standard books on statistical tests.

2.6.2 Iman-Davenport test

Iman-Davenport test is an extension of Friedman test [49]. The Friedman test (Friedman two-way analysis of variances by ranks) is a nonparametric analog of the parametric two-way analysis of variance. It can be used for answering the following question: in a set of k samples (where $k \geq 2$), do at least two of the samples represent populations with different median values?. It is a multiple

comparisons test that aims to detect significant differences between the behavior of two or more algorithms. The null hypothesis for Friedmans test states equality of medians between the populations. The alternative hypothesis is defined as the negation of the null hypothesis, so it is non-directional.

The first step in calculating the test statistic is to convert the original results to ranks. They are computed using the following procedure:

1. Gather observed results for each algorithm/problem pair.
2. For each problem i, rank values from 1 (best result) to k (worst result). Denote these ranks as $r_i^j (1 \leq j \leq k)$.
3. For each algorithm j, average the ranks obtained in all problems to obtain the final rank $R_j = \frac{1}{n} \sum_i r_i^j$.

Thus, it ranks the algorithms for each problem separately; the best performing algorithm should have the rank of 1, the second best rank 2, etc. Again, in case of ties, average rank is computed. Under the null hypothesis, which states that all the algorithms behave similarly (therefore their ranks R_j should be equal) the Friedman statistic F_f can be computed as:

$$F_f = \frac{12n}{k(k+1)} \left[\sum_j R_j^2 - \frac{k(k+1)^2}{4} \right] \tag{2.10}$$

which is distributed according to a χ^2 distribution with $k-1$ degrees of freedom, when n and k are big enough ($n > 10$ and $k > 5$).

Iman and Davenport proposed a derivation from the Friedman statistic given that this last metric often produces a conservative effect not desired. The proposed statistic is:

$$F_{ID} = \frac{(n-1)\chi_F^2}{n(k-1) - \chi_F^2} \tag{2.11}$$

which is distributed according to an F distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom.

2.6.3 Holm's post hoc test

The main drawback of the Friedman and ImanDavenport tests is that they only can detect significant differences over the whole multiple comparison, being unable to establish proper comparisons between

some of the algorithms considered. When the aim of the application of the multiple tests is to perform a comparison considering a control method and a set of algorithms, a family of hypotheses can be defined, all related to the control method. Then, the application of a post hoc test can lead to obtaining a p-value which determines the degree of rejection of each hypothesis.

A family of hypotheses is a set of logically interrelated hypotheses of comparisons which, in $1 * N$ comparisons, compares the $k1$ algorithms of the study (excluding the control) with the control method, whereas in $N * N$ comparisons, it considers the $\frac{k(k-1)}{2}$ possible comparisons among algorithms. Therefore, the family will be composed of $k1$ or $\frac{k(k-1)}{2}$ hypotheses, respectively, which can be ordered by its p-value, from lowest to highest. The p-value of every hypothesis in the family can be obtained through the conversion of the rankings computed by each test by using a normal approximation. The test statistic for comparing the i-th algorithm and j-th algorithm, z , depends on the main nonparametric procedure used.

$$z = (R_i - R_j)\sqrt{\frac{k(k+1)}{6n}} \tag{2.12}$$

where R_i and R_j are the average rankings by the Iman-Davenport test of the algorithms compared. The z -value in all cases is used to find the corresponding probability (p-value) from the table of normal distribution $N(0,1)$, which is then compared with an appropriate level of significance α.

The Holm's post hoc test adjusts the value of α in a step-down manner. Let $p_1, p_2, ..., p_{k-1}$ be the ordered p-values (smallest to largest), so that $p_1 \leq p_2 \leq ... \leq p_{k-1}$, and let $H_1, H_2, ..., H_{k-1}$ be the corresponding hypotheses. The Holm procedure rejects H_1 to H_{i-1} if i is the smallest integer such that $p_i > \frac{\alpha}{(k-1)}$. Holms step-down procedure starts with the most significant p-value. If p_1 is below $\frac{\alpha}{(k-1)}$, the corresponding hypothesis is rejected and p_2 is compared with $\frac{\alpha}{(k-1)}$. If the second hypothesis is rejected, the test proceeds with the third, and so on. As soon as a certain null hypothesis cannot be rejected, all the remaining hypotheses are retained as well.

Holm $APV_i : min(v, 1)$, where $v = max((1 - p_j)^(k - j) : 1 \leq j \leq i)$.

2.7 Summary

Multi-label classification itself is more difficult than single label classification. Several methods for multi-label learning exists in the literature. But the issues that arise in the multi-label data stream

environment makes classification even more challenging for the stream learning models. Although, there are online or incremental learning methods for streaming environments, the multi-model ensemble methods tend to exhibit better performance than traditional single model incremental methods. Concept evolution, a prime challenge posed by data streams, cannot be handled by the traditional stream classification models and requires unsupervised mining techniques like, the clustering of the streams. In a multi-label data stream scenario, the hard clustering methods do not apply due the multiplicity of label cardinality of the stream instances. So, fuzzy clustering techniques are applied to identify multiple novel concepts that may appear in the stream.

Chapter 3

Proposed Method

In this chapter, we first introduce the notations that will be used to describe our proposed framework. A brief overview of the proposed approach is then provided. Finally, the proposed layered ensemble framework and associated algorithms are explained in detail.

3.1 Symbols and notations

Let us consider a multi-label data stream S with $P\%$ of the instances being labeled. L denotes the set of existing class labels in S. As concept evolution is a common phenomena in the data streams L is not fixed for S because novel classes may appear in the stream with the passage of time. The average label cardinality is denoted by LC, i.e, instances in S belong to LC class labels on average. The value of LC is dependent on the data domain and assumed to be known beforehand [14, 7]. Although, SMART [4] learns the value of LC from the labeled examples observed so far by obtaining their average label cardinality and setting it as LC. For the batch incremental setting, S is divided into chunks $\{D_1, D_2, \ldots, D_j, \ldots\}$ each of size B. Here, D_j denotes the j-th chunk in the stream.

In the $LEAD$ framework, each instance x_i in a given data chunk is associated with two attributes: a set of class labels (Ls) and the time of arrival (t). For convenience, we address each data point as timestamped instance X with attributes $X.Ls$ and $X.t$. Arrival time is assigned to 0 ($X.t \leftarrow 0$) for each newly arrived X. T_d is the allowable time constraint until which classification of an instance can be deferred and T_u is the allowable time constraint until which an instance is not considered for novelty detection.

Figure 3.1 illustrates the significance of T_d and T_u with an example. Here, X_k is the latest instance

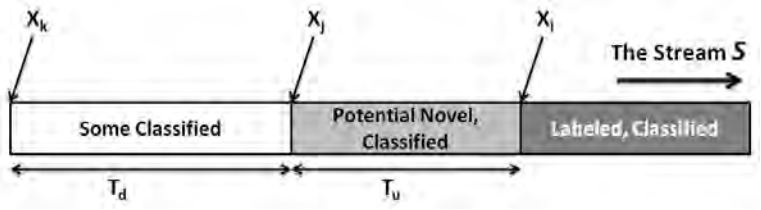


Figure 3.1: Illustration of T_d and T_u

that has arrived in S. Let, X_j be the instance that arrived T_d time units earlier, and X_i be the instance that arrived T_u time units earlier. Then, X_i and all instances that arrived before X_i (shown with dark-shaded area) are classified and labeled since all of them are at least T_u time units old. On the other hand, X_j and all instances that arrived before X_j but after X_i (the light-shaded area) are either classified or candidate for novel class detection. Since, the instances are at least T_d time units old some of them will be classified by the classifier. The rest of the instances were not classifiable by the classifier and are considered as potential novel class instances. Instances that arrived after X_j (age less than T_d) may or may not be classified (shown with the unshaded area). Since S contains both labeled and unlabeled instances, some of the instances in both light shaded ($t \leq T_u$) and unshaded ($t \leq T_d$) are labeled ($P\%$) and used to train the classifiers in the framework. The rest of the instances in those areas are unlabeled. In summary, T_u is enforced by labeling an instance X after T_u time units of its arrival, and T_d is enforced by either identifying X as potential novel instance or by classifying X within T_d time units of its arrival, for every instance X in the stream.

Setting the value of the time constraints T_d and T_u is itself an independent research problem. The value of T_d depends on both the speed of at which the stream data arrives and the speed at which the base classifiers can be constructed. If stream data arrives at a higher rate, then more labeled instances will be readily available for the classification model. As a result, informed classifiers can be learned quickly and hence, T_d can be set to a lower value. The stream data arrival rate depends on the underlying application that generates the data stream. For example, social networks like twitter, facebook, etc. generate data at a speed which is much higher than credit card transaction data of a particular bank. On the other hand, the learning rate of a classifier also contributes to the value of T_d. If a base classifier A can learn at higher rate than another base classifier B, then T_d can be set to a higher value for A than B. The faster learning rate of a classifier allows the model to defer the classification for longer period of time and still attain faster classification results. Similar to T_d, the value of T_u also depends on the aforementioned factors. Furthermore, as T_u is related to novel

class detection, its value also depends on the clustering speed of the novelty detector. If the novelty detector can perform clustering at a higher speed, then the value of T_u can be a set to higher value and still attain a fast response. So, the value of these time constraints are dictated by several factors and can be tuned based on these factors to attain a faster data stream classification framework.

LEAD maintains six buffers buf_l, buf_u, buf_d, buf_{uu}, buf_{novel} and buf_{feed} for storing instances with different status while processing S. buf_l and buf_u stores the labeled and unlabeled instances of the current data chunk, respectively. buf_d stores the instances with $X.t < T_d$ and are tired for classification until $X.t$ exceeds T_u. buf_{uu} stores the instances with $X.t < T_u$ (i.e., potential novel instance) and are tired for classification until $X.t$ exceeds T_d. buf_{novel} stores the instances with $X.t > T_u$ and are the eventual candidates for novel class detection. buf_{feed} stores the classified and novel class instances. Instances in buf_{feed} are labeled by human experts. Table 3.1 summarizes the symbols and terms used across the paper.

Table 3.1: Commonly used symbols and terms

S	Multi-label Data Stream	buf_l	Buffer of labeled instances
D_j	j-th data chunk	buf_d	Buffer of instances with deferred classification
B	Data chunk size	buf_u	Buffer of unlabeled instances
X	Time Stamped Instance	buf_{uu}	Buffer of unlabeled unclassified instances
$X.x$	Feature vector of data point x	buf_{feed}	Buffer for Manual Labeling
$X.Ls$	Label set of $X.x$	buf_{novel}	Buffer for novel class detection
$X.t$	Arrival Time of $X.x$	P	% of labeled data
L	Set of Class Labels	B_{novel}	Unlabeled buffer size
LC	Average Label Cardinality of S	T_d	Time constraint for deferred classification
l	A class label	T_u	Time constraint for novelty detection
E_l	Classifier ensemble for class label l	C	Number of Clusters for fuzzy c-means
G	Classifier ensemble	m	Fuzzyness co-efficient, $m = 2$
T_{Lr}	Top layer ensemble	ϵ	Termination criterion for fuzzy c-means, $0 < \epsilon < 1$
B_{Lr}	Bottom layer ensemble	Θ	Fuzzy Clustering Model
BR_l^i	i-th Binary Relevance classifier in E_l	N_Θ	Set of instance frequencies per cluster in Θ
N_m	Number of classifiers per label's ensemble	TBR_l	Temporary BR classifier trained class label l

3.2 Overview of the approach

Our proposed framework maintains a two layered ensemble architecture, G. The two layers are called the top layer ensemble, T_{Lr} and the bottom layer ensemble, B_{Lr}. Both T_{Lr} and B_{Lr} contain label based ensembles (E_ls) [7] for different class labels (ls) in S. T_{Lr} reflects the concept present in the most recent chunk, i.e, an E_l is kept in T_{Lr} only if instances having the class label l were encountered in the most recently processed chunk. On the other hand, B_{Lr} reflects the old concept, i.e, the concept

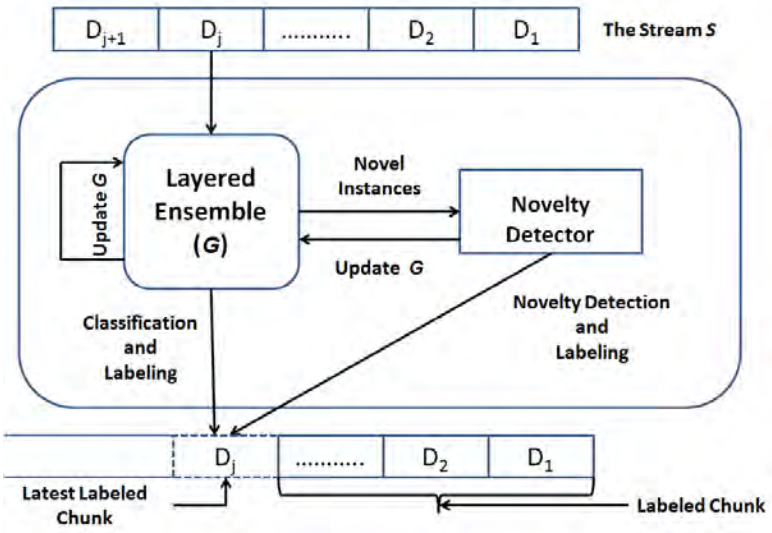


Figure 3.2: *LEAD* Framework

that is not present in the recent chunk. The motivation for maintaining the two ensemble layers is as follows:

1. Instead of classifying an instance X with all the E_ls in G, we compare it with only the E_ls in T_{Lr}. This is done because the current chunk is most likely to be similar in concept with the preceding chunk. X is only classified with the ensembles in B_{Lr} if the LC criteria is not met, i.e, $X.Ls < LC$. As a result, extraneous computation can be avoided.

2. The layered approach helps with the class recurrence scenario. B_{Lr} contains E_ls of those class labels, whose instances did not appear in the most recent chunk. If instances having any of those class labels appear in future, they can be correctly classified by B_{Lr}, a part of G.

3. The layered approach also helps to differentiate between novel class and recurrent class instances. Novel class instances can't be classified by either of top or bottom layer ensembles. But failure to differentiate between novel class and recurrent class instances will increase the false alarm rate. The layered ensemble approach pre-emptively limits the false alarm rate. Moreover, failure to detect novel class results in poor classification performance which is also addressed in this framework.

In each E_l, a set of N_m binary relevance (BR) classifiers [15] is maintained as base multi-label classifiers. The BR approach transforms a multi-label problem into multiple binary problems and

tackles each problem independently. The advantage of this approach is that it can be combined with any binary classification algorithm. Furthermore, it can easily handle the appearance of instances with a new label by training a new binary classifier. Other methods like CC [28], RAKEL [33], HOMER [27], PCT [50], ML-kNN [9] etc. could have also been used in our $LEAD$ framework.

The $LEAD$ framework is depicted in Figure 3.2. As a new chunk D_j arrives, one BR model BR_l^j is first trained for instances of each l in D_j. Then G tries to classify the instances in D_j by using T_{Lr} and B_{Lr}. If T_{Lr} can classify an instance it is stored in buf_{feed} for manual labeling. Otherwise, the instance is forwarded to B_{Lr} for classification. If B_{Lr} fails to classify the instance, it is stored in buf_d for deferred classification. Any instance stored in buf_d that cannot be classified within the time interval T_d is considered as a prospective novel instance and stored into buf_{novel}. When the buf_{novel} has enough ($|buf_{novel}| \geq B_{novel}$) instances, a "Novelty Detector" tries to identify a set of novel classes from the prospective novel instances. The novel class instances are then labeled and forwarded to the output of the framework. The "Novelty Detector" also propagates the new class label information to G for subsequent refinement. Finally, G is updated to best reflect the current concept of stream S. In each E_l, the most poorly performing classifier (BR_l^j) is always replaced by the newly trained classifier. The replacement strategy is introduced to keep the model up to date in order to deal with concept drift.

3.3 Algorithmic framework

The $LEAD$ framework can be divided in the following phases: (i) Initial layered ensemble construction, (ii) Classification, (iii) Novel class detection, and (iv) Ensemble refinement and update. While Algorithm 3.1 outlines the entire $LEAD$ framework, the other algorithms (Algorithms 3.2–3.4) correspond to different phases of $LEAD$. To illustrate the construction and update of G (phase i and iv), we consider the following scenario: Let, the first four chunks of a data stream S are $D_1 = \{A, C, F\}$, $D_2 = \{A, C, H\}$, $D_3 = \{A, F, Q\}$, and $D_4 = \{A, C, H\}$ where $L = \{A, C, F, H, Q\}$. For simplicity the chunks are represented with class labels of the labeled instances. We assume that $N_m = 2$, i.e. the initial G is constructed from two chunks and each E_l in G will maintain two BR_l^is. As we analyze the different phases of $LEAD$, we inspect how they function for the given example. The different states of G for the aforementioned scenario are shown in Figure 3.3.

Algorithm 3.1 LEAD(S)

Input: Data Stream S

Set buf_l, buf_d, buf_u, buf_{uu}, and buf_{novel} as empty
$G \leftarrow$ **Build-initial-Ensemble**()
while S is not empty **do**
    Load New Chunk D_j
    $buf_l \leftarrow$ labeled instances of D_j
    $buf_u \leftarrow$ unlabeled instances of D_j
    $TBR \leftarrow$ **Train**(buf_l) ▷ $TBR \leftarrow$ Set of TBR_l
    for all $TBR_l \in TBR$ **do**
        Add TBR_l to ensemble E_l of G
    end for
    for each instance x in buf_u **do**
        $X \leftarrow$ empty
        $X.x \leftarrow x$
        $X.t \leftarrow 0$
        Classify(G, X) ▷ Algorithm 3.2
    end for
    $buf_{temp} \leftarrow$ empty
    for each instance X in buf_d **do**
        if $X.t \leq T_d$ and $X.t \neq 0$ **then**
            Classify(G, X)
        else if $X.t > T_d$ **then**
            Add(buf_{uu},$X.x$)
        else
            Add(buf_{temp}, X) ▷ Classification deferred.
        end if
    end for
    $buf_d \leftarrow buf_{temp}$
    $buf_{temp} \leftarrow$ empty
    for each instance X in buf_{uu} **do**
        if $X.t \leq T_u$ **then**
            Classify(G, X)
        else if $X.t > T_u$ **then**
            Add(buf_{novel},$X.x$)
        else
            Add(buf_{temp}, X) ▷ Novelty Detection deferred.
        end if
    end for
    $buf_{uu} \leftarrow buf_{temp}$
    if $|buf_{novel}| \geq B_{novel}$ **then**
        $L' \leftarrow$**DetectNovelLabelSet**(buf_{novel}) ▷ with C, ϵ, m. Algorithm 3.3
    end if
    Manual labeling of buf_{novel} and buf_{feed}
    $buf_l \leftarrow buf_l \cup \{buf_{novel} \cup buf_{feed}\}$
    $G \leftarrow$**Update-Model**(L,L') ▷ Algorithm 3.4
    Empty all buffers
end while

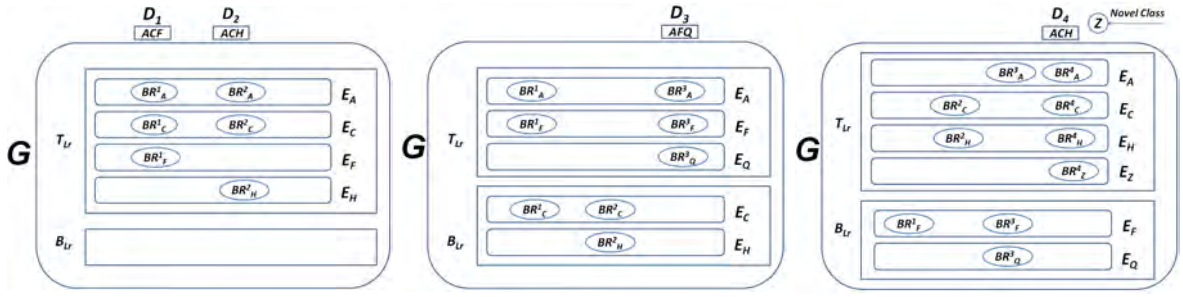
3.3.1 Initial ensemble construction

The initial ensemble G is constructed by **Build-initial-Ensemble**() procedure (Algorithm 3.1, Line 2) with the first N_m data chunks. Whenever a new chunk D_j arrives, its labeled and unlabeled instances are stored in buf_l and buf_u, respectively. For each class label l in buf_l, $LEAD$ first trains a new BR_l^j using instances having class label l. It then adds BR_l^j to the corresponding label based ensemble E_l. The process is continued until N_m chunks have arrived.

To visualize this process, let us consider Figure 3.3a. As the data chunk D_1 arrives, $LEAD$ first trains classifiers BR_A^1, BR_C^1 and BR_F^1 and then adds to E_A, E_C and E_F, respectively. These three ensembles are the members of the top layer ensemble T_{Lr}. $LEAD$ repeats the same process for the chunk D_2. It means one more classifier is added to both E_A and E_C because D_2 contains instances with the class labels A and C. Moreover, a new label based ensemble E_G is added to T_{Lr} due the presence of instances with the class label G. Initially, all the E_ls are added to T_{Lr} and B_{Lr} remains empty.

3.3.2 Stream data classification

This phase involves Algorithms 3.1 and 3.2. Starting from the $(N_m + 1)$-th data chunk, labeled and unlabeled instances of a newly arrived chunk D_j are first stored in buf_l and buf_u, respectively. Then a temporary binary classifier TBR_l is trained with instances having class label l. TBR_l does not replace any existing classifier in E_l, rather acts as the representative of the concept of the current chunk D_j. Each instances x of buf_u (Algorithm 1, Lines 16-18) is then fetched and assigned time 0 ($X.t \leftarrow 0$). As discussed before, such an instances is called the timestamped instance (X). Each E_l in T_{Lr} and if required the B_{Lr}, attempt to classify X (Algorithm 3.2). Classification is done based



(a) Initial Ensemble Construction (b) Ensemble update (c) Ensemble update

Figure 3.3: Different states of the layered ensemble architecture.

on the majority voting of the classifiers in E_l. The vote of TBR_l is considered during classification decision. During classification the following cases may arise:

1. X can be classified by the T_{Lr}.
2. X can be classified by B_{Lr}, but not T_{Lr}.
3. X can be classified by neither T_{Lr} nor by B_{Lr}, i.e. $|X.Ls| = 0$

Case 1: X is classifiable by T_{Lr} and is labeled accordingly (Algorithm 3.2, Lines 1–18). If $|X.Ls| < LC$, then X is further classified by B_{Lr}. The advantage with this approach is that if X's label set satisfies the cardinality condition we need not check the bottom layer. As a result, excess computation can be avoided across the entire stream S.

Case 2: X is classifiable by B_{Lr} but not by T_{Lr}, i.e, $X.x$ belongs to classes whose instances haven't appeared in recent chunks. Hence, the corresponding E_ls were demoted to the bottom layer ensemble. So, Case 2 handles to the class recurrence scenario.

Case 3: This case arises when $X.Ls$ is empty (Algorithm 3.2, Lines 19-29). Which means the current architecture can't classify X. There can be two sub-cases: a) X belongs to a novel class; b) some classifier BR_l^j can classify X, yet X is voted unclassifiable by majority of the classifiers of E_l. Time stamped instances with Case 3a, are stored in buf_{uu} for further consideration. Case 3b may occur when X's class label is l, but all the E_ls in G are underdeveloped and can't classify X. So, X's classification is deferred and stored in buf_d for further classification attempts.

The classified instances are stored in buf_{feed}. The **Classify**(G, X) procedure is followed up by deferred classification (Algorithm 1, Lines 21-31). Each X in buf_d is fetched again for classification. Then $X.t$ is updated. If $X.t$ is less than T_d and X's classification has failed, it is again stored in buf_d. On the other hand, if $X.t$ exceeds T_d, it is stored in buf_{uu}. To avoid classification attempt on X twice in the same iteration, the $X.t = 0$ condition is checked (line 23). Finally, before an X in buf_{uu} is eligible for novelty detection (when $X.t < T_u$), it is again tried for classification. If classification fails, $X.t$ is updated. If X is eligible for novelty detection it is stored in buf_{novel} for further analysis in novel class detection phase.

3.3.3 Novel class detection

This phase involves Algorithm 3.3 and deals with instances in buf_{novel}. In the multi-label data stream, one record may belong to multiple classes. Hence, the hard clustering methods like k-means clustering

Algorithm 3.2 Classify(G, X)

Input: Layered ensemble G, Time stamped instance X
Output: Classification Decision.

for all $l \in L$ and E_l in top layer **do**
  $Dsn \leftarrow$ empty set of decisions
  for all classifier $i \in E_l$ **do**
    $Dsn \leftarrow Dsn \cup BR_l^i(X.x)$
  end for
  $decision_l \leftarrow$ **majority-voting**(Dsn)
end for
$X.Ls \leftarrow \overline{\textbf{decision}}$
if $|X.Ls| < LC$ **then**
  for all $l \in L$ and E_l in bottom layer **do**
    $Dsn \leftarrow$ empty set of decisions
    for all classifier $i \in E_l$ **do**
      $Dsn \leftarrow Dsn \cup BR_l^i(X.x)$
    end for
    $decision_l \leftarrow$ **majority-voting**(Dsn)
  end for
end if
$X.Ls \leftarrow \overline{\textbf{decision}}$
if X.Ls is empty **then**
  if some classifier N_m of any E_l can classify $X.x$ **then**
    $X.t \leftarrow X.t + 1$
    Add(buf_d, X)
  else ▷ no classification decision can be made
    $X.t \leftarrow X.t + 1$
    Add(buf_{uu}, X)
  end if
else ▷ X can be classified
  Add(buf_{feed}, X)
end if

cannot be applied for novelty detection in multi-label data streams. In the *LEAD* framework, we deploy fuzzy clustering technique for the novel class detection. In fuzzy clustering, every data point has a degree of associativity to clusters, rather than belonging completely to just one cluster. Here, we have used the Fuzzy C-means (FCM) clustering algorithm [41] for constructing our novelty detector.

Algorithm 3.3 DetectNovelLabelSet(buf_{novel})

Input: Buffer of Novel Instances buf_{novel}
Output: Novel Label Set L'

1: $c \leftarrow 2$
2: **for** $c < |buf_{novel}|$ **do**
3: $\quad \Theta_c \leftarrow$ **Fuzzy-C-Means**(buf_{novel}, c, m, ϵ)
4: $\quad \Omega \leftarrow \Omega \cup \Theta_c$
5: **end for**
6: $c_{max} \leftarrow c$ value from where monotonically decreasing tendency of W starts
7: $\Theta_{best} \leftarrow \min_{2 \leq c \leq c_{max}} \{\min_{\Theta_c \in \Omega} W\}$
8: $L' \leftarrow$ **Cluster-to-Class**(Θ_{best})

In our approach, a fuzzy clustering model Θ is generated using **Fuzzy-C-Means**() procedure. Apart from buf_{novel}, the procedure takes three more parameters. They are: C, m, and ϵ. The main assumption behind novel class detection is that an instance should be closer to the instances of its own class (compactness) and farther apart from the instances of other classes (separation). The validity of the fuzzy clusters obtained are measured based on a compact and separate fuzzy validity criterion W as discussed in Section 2.5.3. A lower value of W for a cluster combination indicates better compactness within a cluster and more separation among all the clusters.

A problem of implementing W is that it has a tendency to eventually decrease when c becomes large. So, the value of W is meaningless when c gets close to $|buf_{novel}|$. To address this issue, [45] deploys a heuristic to identify the maximum acceptable value, c_{max}. According to the heuristic, first the W value is obtained for $c = 2$ to $|buf_{novel}|-1$. Then the c value from which W starts monotonically decreasing is considered to be c_{max}. With $c = 2$ to $|buf_{novel}| - 1$, the cluster combination for which the minimum W value can be found is selected as the best fuzzy clustering (Θ_{best}). The clusters in Θ_{best} are selected as novel clusters by the novelty detector. The novel clusters are then labeled and corresponding class labels are added to L' (Algorithm 3.3, Line 8), a set of novel class labels. L' is used by ensemble refinement and update phase for subsequent update of G.

Algorithm 3.4 Update-Model(L,L')

Input: Existing Label set L, Novel Label Set L'
Output: Update G

1: **for all** $l \in L$ **do**
2: Remove TBR_l from E_l
3: **end for**
4: $BR^j \leftarrow$ **Train**(buf_l) ▷ $BR \leftarrow$ Set of BR_l^j
5: $L_{curr} \leftarrow$ empty
6: **for all** $BR_l^j \in BR$ **do**
7: $L_{curr} \leftarrow L_{curr} \cup l$
8: Remove the worst performing Binary classifier in E_l
9: Add BR_l^j to ensemble E_l of G
10: **end for**
11: **for all** $l' \in L'$ **do**
12: $E_{l'} \leftarrow$ Ensemble for Label l'
13: Add $E_{l'}$ to T_{Lr}
14: **end for**
15: **for all** $l \in L_{curr}$ and $E_l \in B_{Lr}$ **do**
16: Remove E_l from B_{Lr}
17: Add E_l to T_{Lr}
18: **end for**
19: $L_{old} \leftarrow L - L_{curr}$
20: **for all** $l \in L_{old}$ and $E_l \in T_{Lr}$ **do**
21: Remove E_l from T_{Lr}
22: Add E_l to B_{Lr}
23: **end for**

3.3.4 Ensemble refinement and update

This phase corresponds to Algorithm 3.4. All the instances of buf_{novel} and buf_{feed} are first added to buf_l (Algorithm 3.2, Line 47). All the TBR_ls are removed from G next. Then new classifiers are learned with instances having class label l. The poorest classifier BR_l^j in E_l is replaced with the new classifier. Moreover, the novelty detector may detect novels classes and new ensemble $E_{l'}$ is constructed for each of the new class labels l'. $E_{l'}$s are then added to T_{Lr}. Finally, top layer E_ls whose class instances were not encountered in the current chunk are transferred to B_{Lr}. On the other hand, bottom layer E_ls whose class instances were encountered in the current chunk are transferred to T_{Lr} (Algorithm 3.4, Lines 15-23).

To visualize the ensemble update process, let us consider the scenarios in Figure 3.3b and 3.3c corresponding to the example discussed at the start of this section. After data chunk D_3 is processed by the previous two phases, a new ensemble E_Q is generated due to the presence of labeled instances

of class Q (Figure 3.3b). E_Q is then added to T_{Lr}. As instances of class C and G were not present in D_3, E_C and E_G are demoted to B_{Lr}. The other E_ls kept in T_{Lr}. From Figure 3.3c, it can be seen that a novel class Z has been identified by the novelty detector. So, E_Z is constructed with a classifier BR_Z^4 and added to T_{Lr}. As instances of class C and G were present in D_4, E_C and E_G are promoted to T_{Lr}. But E_F and E_Q are demoted to B_{Lr} due to the absence of instances of class F and G in D_4.

3.4 Summary

$LEAD$ is a multi-phase classification framework that can detect and identify concept evolution in multi-label data streams. The primary component of $LEAD$ is a layered ensemble architecture that can classify the existing class instances and isolate the novel class instances from existing class instances. The novelty detector component of the framework can identify emergence novel concepts in the stream. The ensemble architecture can also detect recurrent class instances and classify them accordingly. The ensemble architecture of $LEAD$ is updated and refined regularly to keep the model up to date even in the presence of concept drift and concept evolution.

Chapter 4

Experimental Result

In this chapter, we evaluate the effectiveness of our proposed *LEAD* framework and compare its performance with two baseline methods. We first describe the datasets used for experimental evaluation. The evaluation measures for analyzing the predictive performance of the competing methods are then discussed. Two new evaluation measures for novel class detection have been proposed in this thesis. The classification and novel class detection performance of the competing methods are then analyzed. We also establish the statistical significance of *LEAD* by using statistical tests.

4.1 Datasets

We have used 11 different multi-label datasets. Of them, 10 are multi-label classification benchmark data sets from different application domains and 1 is a synthetic datasets. Table 4.1 presents the basic statistics of the datasets. We can note that the datasets vary in size: from 1702 up to 10,00,000 instances, from 103 up to 47236 features (numeric or nominal), from 6 to 374 labels, and from 1.252 to 4.376 label cardinality. First, we discuss the benchmark datasets followed by a discussion on the synthetic datasets.

4.1.1 Benchmarks datasets

The benchmark datasets are from three domains: biology, multimedia and text categorization. From the biological domain, we have the yeast dataset [51]. It is a widely used dataset, where genes are instances in the dataset and each gene can be associated with 14 biological functions (labels). The datasets that belong to the multimedia domain are: scene, corel5k and mediamill. Scene [52] is a

Table 4.1: 11 datasets from different application domains. B = nominal attributes, N = numeric attributes

Name	Domain	Instances	Labels	Attributes	Cardinality
20NG	Text	28596	22	1001B	2.2
corel5k	Multimedia	5000	374	499B	3.522
enron	Text	1702	53	1001B	3.378
mediamill	Multimedia	43907	101	120N	4.376
rcv1v2(subset1)	Text	6000	101	47236N	2.88
scene	Multimedia	2407	6	294N	1.074
tmc2007	Text	28596	22	500B	2.158
yeast	Biology	2417	14	103N	4.237
slashdot	Text	3782	22	1079B	1.2
imdb	Text	120919	28	1001B	2.0
SynT-D	Synthetic	1000000	8	30B	3.0

widely used scene classification dataset. Each scene can be annotated in the following six contexts: beach, sunset, field, fall-foliage, mountain, and urban. The Corel5k [53] dataset contains Corel images that are segmented using normalized cuts. The segmented regions are then clustered into 499 bins, which are further used to describe the images. Each image can be then assigned several of the 374 possible labels. Mediamill [54] originates from the 2005 NIST TRECVID challenge dataset, which contains data about annotated videos. The label space is represented by 101 "annotation concepts", such as explosion, aircraft, face, truck, urban, etc.

The domain of text categorization is represented with six datasets: 20NG, enron, tmc2007, slashdot, imdb, and finally, rcv1v2: subset1. 20NG is the classic 20 newsgroups collection [55] with around 20,000 articles sourced from 20 newsgroups. Enron [56] is a dataset that contains the e-mails from 150 senior Enron officials. The e-mails were categorized into several categories developed by the UC Berkeley Enron Email Analysis Project. The labels can be further grouped into four categories: coarse genre, included/forwarded information, primary topics, and messages with emotional tone. tmc2007 [57] contains instances of aviation safety reports that document problems that occurred during certain flights. The labels represent the problems being described by these reports. We use a reduced version of this dataset with the top 500 attributes selected, same as Tsoumakas et al. [58]. Slashdot [59] consists of article blurbs with subject categories, mined from *http://slashdot.org/*. IMDB data set [28] contains 120,919 movie plot text summaries gathered from the Internet Movie Database. This data set is labeled as one or more classes out of 28 labels. rcv1v2 text data sets [60] are popularly

used as benchmark in text classification. The data set contains Reuter newswire stories and we use one of the subsets of the dataset each containing 6000 instances with 47236 numeric features. rcv1v2 text data were collected in time sequence, and thus has concept drifting problem.

To simulate in our proposed framework, these data sets are processed as follows:

1. First, a set of class labels are selected at random as novel class labels. The number of novel class labels range from 2 to 10 depending on the label set cardinality ($|L|$) of a dataset. To simulate concept evolution, instances of these class labels are not included in the data stream until 50% instances of the entire dataset has already been sent to the stream.

2. Then a set of class labels are selected at random as recurrent class labels. The number of recurrent class labels range from 2 to 4 depending on the label set cardinality ($|L|$) of a dataset. Instances of these class labels are initially included in the data stream. But to simulate class recurrence, after a few chunks are processed by our framework, the selected recurrent class instances are excluded from inserting into the stream. These class instances are again pushed into the stream after 60% of the remaining instances in the dataset have been included in the stream.

3. While inserting an instance in the data stream, a random value v $(0 < v < 1)$ is picked. The instance is then pushed in the stream as labeled ($|X.Ls| > 0$) or unlabeled ($|X.Ls| = 0$) with probability P.

4.1.2 Synthetic datasets

The benchmark methods apart from rcv1v2(subset1) don't present any concept drift. The SynT-Drift dataset can be found in [14] that simulates concept drift. We denote the data set as SynT-D. Three concept drifts of varying type, magnitude, and extent occur in SynT-drift. For N generated examples, the drifts are centred over examples $1/N$,$2/N$,and $3/N$, extending over $N/1000$, $N/100$, and $N/10$ examples, respectively. In the first drift, only 10% of label dependencies change. In the second drift, the underlying concept changes and more labels are associated on average with each example (a higher label cardinality). In the third drift, 20% of label dependencies change. Such types and magnitudes of drift can be found in real world data. To simulate concept evolution and class recurrence in SynT-D, we have also deployed the processing steps as discussed in Section 4.1.1. We call this variation

SynT-DN. Here, two classes were chosen at random for simulating concept evolution and two classes were chosen at random for simulating class recurrence.

4.2 Baseline methods

To the best of our knowledge, there is no approach that can classify multi-label data streams and detect novel class simultaneously. So, we like to compare our approach with a combination of two baseline techniques: $OLINDDA$ [40] combined with either Ensemble of Hoeffding Trees with Pruned Sets classifier (E_aHT_{PS}) [14] or Chunk Based Ensemble Framework ($CBEF$) with binary relevance classifier. $OLINDDA$ works as novel class detector and both E_aHT_{PS} and $CBEF$ perform multi-label classification. As discussed in Section 1.1, E_aHT_{PS} is a bagged adaptive window based instance incremental approach for classifying concept drifting multi-label data streams. On the other hand, $CBEF$ is a batch incremental multi-label classification process where classifiers are learned per chunk and an ensemble of these classifiers are maintained for classification of data stream instances. Whenever a new chunk arrives, a new classifier is learned. The new classifier then replaces the one of the existing classifiers. The classifier that most poorly classifies the current chunk is chosen for replacement. $OLINDDA$ has been discussed in detail in Section 2.5.1. We denote the baseline methods as OE ($OLINDDA$ combined with E_aHT_{PS}) and OC ($OLINDDA$ combined with $CBEF$).

4.3 Experimental Setup

The experiments on the baseline methods are conducted as follows: for each instance, $OLINDDA$ determines whether the instance is novel. The classification of an instance is delayed by T_d time interval. That is, OLINDDA is given T_d time intervals to determine whether an instance is novel. If the stream instance is identified as a novel class instance, then it is considered novel and not classified using the classifiers. Otherwise, the instance is assumed to be an existing class instance, and its class is predicted using the classifier corresponding to the baseline method. We use $OLINDDA$ as the novelty detector since it is a novel concept detection technique that has shown to have outperformed other novelty detection techniques in data streams [40]. However, $OLINDDA$ assumes that there is only one "normal" class, and all other classes are "novel". So, it is not directly applicable to the multi-label novelty detection problem, where any combination of classes can be considered as the "existing" classes. Therefore, we propose an alternative solution. We build parallel $OLINDDA$ models, one for

each class, which evolve simultaneously. Whenever the instances of a novel class appear, we create a new $OLINDDA$ model for that class. A test instance is declared as novel, if all the existing class models identify this instance as novel. In all experiments, the ensemble size and chunk size are kept the same for all the baseline techniques. Besides, the same base learner is used for E_aHT_{PS}, $CBEF$ and $LEAD$.

All the methods (E_aHT_{PS}, $CBEF$ and $LEAD$) were developed in JAVA. E_aHT_{PS} was developed using the MEKA (http://sourceforge.net/projects/meka/) library and $CBEF$ was developed using the WEKA library [61]. $LEAD$ was developed using both MULAN (http://mulan.sourceforge.net) and WEKA library.

For parameter settings we have classified the datasets into three categories based on number of instances. They are: Small (corel5k, enron, scene, yeast, slashdot, rcv1v2(subset1)), Medium (20NG, mediamill, tmc2007) and Large (imdb, SynT-D, SynT-DN).

Table 4.2: Parameters used in experimentation

Category	B	B_{novel}	N_m	T_d	T_u	C	m	ϵ
Small	300	450	4	3	2	10	2	0.3
Medium	1000	2500	5	3	3	20		
Large	2500	3000	5	3	3	20		

The parameter used in $LEAD$ for different categories of datasets are shown in Table 4.2. The parameters B and B_{novel} are given in terms of number of instances. N_m denotes the number of data chunks considered for initial G construction. T_d and T_u denote the maximum time interval for deferred classification and novel class detection of an instance, respectively. C is the number of clusters initially created. For all the categories, m and ϵ are set to 2 and 0.3, respectively. These parameters are tuned to achieve overall satisfactory performance.

For $CBEF$, the two parameters B and N_m are required. For each dataset, these parameter were set to their optimal values to obtain the best performance for $CBEF$. For E_aHT_{PS}, the adaptive window size is kept equal to chunk size B. The initial classifier is constructed using the first B examples. Among the class labels encountered in the first B examples, the 30 most frequent labels are considered to construct the Pruned Set (PS) classifier. For $OLINDDA$ the parameter settings are B, B_{novel}, N_m, C and T_d. Here, C is interpreted as the number of clusters built in the initial model. The other parameters bear the same meaning as $LEAD$. All the parameter values are set as

per Table 4.2. The experimental results presented in this paper were averaged over 30 runs. During each of the 30 runs, results were averaged over all evaluation windows for OE . On other hand, results were averaged over all the data chunks for OC and $LEAD$ during each of the 30 runs.

4.4 Evaluation measures

Now, we present the measures that are used to evaluate the predictive performance of the competing methods in our experiments. In the definitions below, y_i denotes the set of true labels of example x_i and $h(x_i)$ denotes the set of predicted labels for the same examples. All definitions refer to the multi-label setting. Furthermore, we propose two new evaluation metric for novel class detection.

4.4.1 Evaluation measures for classification

In this thesis, three measures are used for evaluation classification performance. They are: *Accuracy*, F_1-*score* and $Macro - F_1$. The classification measures can be further divided into example based measures and labels based measures.

Example based measures for classification

The example based evaluation measures are based on the average differences of the actual and the predicted sets of labels over all examples of the evaluation dataset. *Accuracy* and F_1-*score* are both example based measures of evaluation.

Accuracy for a single example x_i is defined by the Jaccard similarity coefficients between the label sets $h(x_i)$ and y_i. *Accuracy* is micro-averaged across all examples:

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N}\frac{|h(x_i) \cap y_i|}{|h(x_i) \cup y_i|} \tag{4.1}$$

F_1-*score* is the harmonic mean between precision and recall and is defined as:

$$F_1 - score = \frac{1}{N}\sum_{i=1}^{N}\frac{2 \times |h(x_i) \cap y_i|}{|h(x_i) + y_i|} \tag{4.2}$$

F_1 is an example based metric and its value is an average over all examples in the dataset. F_1 reaches its best value at 1 and worst value at 0.

Label based measures for classification

The label-based evaluation measures assess the predictive performance for each label separately and then average the performance over all labels. $Macro - F_1$ is a label based evaluation metric.

$Macro - F_1$ is the harmonic mean between precision and recall, where the average is calculated per label and then averaged across all labels. If p_j and r_j are the precision and recall then $Macro - F_1$ is defined as:

$$Macro - F_1 = \frac{1}{Q} \sum_{j=1}^{Q} \frac{2 \times p_j \times r_j}{p_j + r_j} \tag{4.3}$$

where p_j and r_j are defined as follows:

$$p_j = \frac{1}{Q} \sum_{j=1}^{Q} \frac{tp_j}{tp_j + fp_j} \tag{4.4}$$

$$r_j = \frac{1}{Q} \sum_{j=1}^{Q} \frac{tp_j}{tp_j + fn_j} \tag{4.5}$$

Here, tp_j and fp_j are the number of true positives and false positives for the j-th label and fn_j is the number of false negatives for the j-th label considered as a binary class.

4.4.2 Evaluation measures for novel class detection

In this section, we propose two new evaluation measures for novel class detection. They are: PF_{novel} and AT_{novel}. PF_{novel} is an example based measure whereas AT_{novel} is a label based measure.

$\mathbf{PF_{novel}}$

PF_{novel} is the fraction of existing class instances identified as novel class instances and is defined as follows:

$$PF_{novel} = \frac{F_{novel}}{N_{existing}} \tag{4.6}$$

where F_{novel} is the number of existing class instances identified as novel class and $N_{existing}$ is the total number of existing class instances. The closer the value of PF_{novel} to 0 the better the performance of the novel class detector. The higher value of PF_{novel} occurs when the novelty detector misidentifies

many existing class instances as novel class instances. This phenomenon usually takes place when recurrent class instances appear in the stream and the novelty detector fails to identify them as existing class instance.

$\mathbf{AT_{novel}}$

AT_{novel} is a measure of the prediction accuracy of the novel class detector. AT_{novel} is defined as the average number of novel class instances identified correctly (true novel) across all novel classes in the data stream and expressed as follows:

$$AT_{novel} = \frac{1}{Q_{novel}} \sum_{i=1}^{Q_{novel}} \frac{(P_{novel})_i}{(T_{novel})_i} \tag{4.7}$$

where Q_{novel} is the number of novel classes identified, $(P_{novel})_i$ is the number of instances predicted as class i instances and $(T_{novel})_i$ is the actual number of instances of novel class i.

The value $AT_{novel} << 1$ occurs for two reasons. They are: (1) if many of the novel class instances have been identified as existing class instance and (2) if very few existing class instances have been identified as novel class instances. So, PF_{novel} will also be very small when $AT_{novel} << 1$. The value $AT_{novel} > 1$ also occurs for two reasons. They are: (1) if the number of true novel class instances identified is either large or small but, a large number of existing class instances are identified as novel class instances or (2) if the number of true novel class instances identified is large but, a small number of existing class instances identified as novel class instances. So, the value of AT_{novel} alone cannot be a measure of the performance of the novelty detector. If PF_{novel} is very close to 1, then AT_{novel} might attain a value close to or greater than 1 which denotes poor performance. On the other hand, given PF_{novel} is close to 0, it can be safely said that the closer the value of AT_{novel} to 1 the better the performance of the novel class detector.

4.5 Results

In this section, we present the classification and novelty detection performance of $LEAD$ and compare them with the baseline methods. Then the performances of the competing approaches are analyzed using several statistical tests.

Table 4.3: Comparison of classification performance for $P = 40$.

Dataset	Approach	*Accuracy*	Macro-F_1	$F1$-score
20 Ng	LD	**0.5643**	**0.5986**	**0.6172**
	OE	0.4671	0.4991	0.5178
	OC	0.3874	0.4057	0.4213
corel5k	LD	**0.4503**	**0.4693**	**0.4824**
	OE	0.2214	0.2471	0.2765
	OC	0.1986	0.2109	0.2347
enron	LD	**0.4029**	**0.4057**	**0.4419**
	OE	0.2718	0.3043	0.3190
	OC	0.2306	0.2500	0.2883
mediamill	LD	**0.3291**	**0.3733**	**0.4202**
	OE	0.2115	0.2361	0.2734
	OC	0.1803	0.2017	0.2394
rcv1v2(s1)	LD	**0.7236**	0.7395	**0.7820**
	OE	0.7080	**0.7461**	0.7619
	OC	0.6416	0.6604	0.6986
scene	LD	**0.7044**	**0.6820**	**0.7090**
	OE	0.5890	0.6021	0.6487
	OC	0.5264	0.6019	0.6514
tmc2007	LD	**0.5163**	0.5273	**0.6207**
	OE	0.4901	**0.5381**	0.5688
	OC	0.4349	0.4528	0.4862
yeast	LD	**0.5209**	**0.5564**	**0.6321**
	OE	0.3243	0.3432	0.3756
	OC	0.3581	0.3726	0.3901
slashdot	LD	**0.5529**	**0.5623**	**0.5786**
	OE	0.4183	0.4375	0.4736
	OC	0.4402	0.4604	0.4847
imdb	LD	**0.3209**	**0.3648**	**0.4162**
	OE	0.2316	0.2743	0.2906
	OC	0.1958	0.2174	0.2538
SynT-DN	LD	**0.1957**	**0.1899**	**0.2201**
	OE	0.1539	0.1710	0.1964
	OC	0.0986	0.1046	0.1207

4.5.1 Classification performance

Table 4.3–4.6 compare the classification performance of $LEAD$ (LD), $OLINDDA - E_aHT_{PS}$ (OE) and $OLINDDA - CBEF$ (OC). The classification results were compared with four different percentage of labeled data in the stream ($P = 40, 60, 80, 90$). For all the measures, higher value indicates better performance.

Table 4.3 shows the classification performance of $LEAD$, OE and OC when 40% of the instances in S are labeled. In all dataset categories $LEAD$ outperforms both OE and OC except for two datasets in case of the Macro-F_1 measure. OE is better than $LEAD$: for rcv1v2(subset1) and tmc2007 dataset. But the difference between those values are quite insignificant (1%). $LEAD$ exhibits very good performance for scene and rcv1v2(subset1) dataset. The overall performance of $LEAD$ for the small datasets are quite noteworthy. As the datasets size increases, there has been a drop in accuracy

Table 4.4: Comparison of classification performance for $P = 60$.

Dataset	Approach	*Accuracy*	Macro-F_1	$F1$-score
20 Ng	LD	**0.6140**	**0.6497**	**0.6615**
	OE	0.4909	0.5246	0.5563
	OC	0.4382	0.4598	0.4779
corel5k	LD	**0.4981**	**0.5016**	**0.5140**
	OE	0.2485	0.2549	0.2801
	OC	0.2046	0.2179	0.2483
enron	LD	**0.4710**	**0.4800**	**0.5080**
	OE	0.3230	0.3361	0.3604
	OC	0.2712	0.2890	0.3172
mediamill	LD	**0.3615**	**0.4096**	**0.4493**
	OE	0.2482	0.2637	0.3159
	OC	0.2031	0.2378	0.2675
rcv1v2(s1)	LD	**0.7912**	**0.8010**	**0.8369**
	OE	0.7293	0.7546	0.7852
	OC	0.6739	0.7005	0.7327
scene	LD	**0.7095**	**0.7011**	**0.7150**
	OE	0.5909	0.6198	0.6505
	OC	0.5798	0.6213	0.6568
tmc2007	LD	**0.5370**	**0.5839**	0.5946
	OE	0.4986	0.5294	**0.6089**
	OC	0.4426	0.4608	0.4972
yeast	LD	**0.5539**	**0.5887**	**0.6589**
	OE	0.3444	0.3761	0.3916
	OC	0.3760	0.3902	0.4253
slashdot	LD	**0.5679**	**0.5781**	**0.5898**
	OE	0.4261	0.4407	0.4823
	OC	0.4498	0.4712	0.5048
imdb	LD	**0.3786**	**0.4056**	**0.4608**
	OE	0.2479	0.2796	0.312
	OC	0.2294	0.2439	0.2883
SynT-DN	LD	**0.2041**	**0.2181**	**0.2366**
	OE	0.1624	0.1829	0.2017
	OC	0.1108	0.1187	0.1485

but not too alarming. For large datasets the accuracy of $LEAD$ is lower compared to the other data categories.

Table 4.4 shows the classification performance of $LEAD$, OE and OC when 60% of the instances in S are labeled. $LEAD$ exhibits very good performance for scene and rcv1v2(subset1) dataset. The overall performance of $LEAD$ for the small datasets are quite noteworthy. As the datasets size increases, again there has been a drop in accuracy but not too alarming. For large datasets the accuracy of $LEAD$ is lower compared to the other data categories. But in all dataset categories $LEAD$ outperforms both OE and OC.

Table 4.5 shows the classification performance of $LEAD$, OE and OC when 80% of the instances in S are labeled. $LEAD$ exhibits very good performance for scene and rcv1v2(subset1) dataset. The overall performance of $LEAD$ for the small datasets are quite noteworthy. As the datasets size

Table 4.5: Comparison of classification performance for $P = 80$.

Dataset	Approach	*Accuracy*	Macro-F_1	$F1$-score
20 Ng	LD	**0.6572**	**0.6905**	**0.7086**
	OE	0.5195	0.5436	0.5742
	OC	0.4726	0.4938	0.5139
corel5k	LD	**0.5273**	**0.5418**	**0.5596**
	OE	0.2653	0.2813	0.308
	OC	0.2194	0.2328	0.2738
enron	LD	**0.5132**	**0.5249**	**0.5470**
	OE	0.3433	0.3614	0.3877
	OC	0.3095	0.3308	0.3692
mediamill	LD	**0.3968**	**0.4283**	**0.4701**
	OE	0.3093	0.3142	0.3419
	OC	0.2558	0.2697	0.3146
rcv1v2(s1)	LD	**0.8289**	**0.8421**	**0.8613**
	OE	0.7583	0.7762	0.8196
	OC	0.7024	0.7321	0.7692
scene	LD	**0.7570**	**0.7685**	**0.7896**
	OE	0.6263	0.6514	0.6761
	OC	0.6093	0.6152	0.6286
tmc2007	LD	**0.5784**	**0.6051**	**0.6813**
	OE	0.5186	0.5568	0.6112
	OC	0.4619	0.4893	0.5385
yeast	LD	**0.5934**	**0.6367**	**0.6567**
	OE	0.3597	0.3918	0.4043
	OC	0.3929	0.4247	0.4302
slashdot	LD	**0.6450**	**0.6483**	**0.6626**
	OE	0.4529	0.4627	0.5089
	OC	0.4746	0.4904	0.5381
imdb	LD	**0.4228**	**0.4410**	**0.5011**
	OE	0.3411	0.3532	0.3927
	OC	0.2503	0.2724	0.3385
SynT-DN	LD	**0.2235**	**0.2333**	**0.2683**
	OE	0.1983	0.2075	0.2436
	OC	0.1402	0.1604	0.1847

increases, there has been a drop in accuracy but not too alarming. For large datasets the accuracy of $LEAD$ is lower compared to the other data categories. But in all dataset categories $LEAD$ outperforms both OE and OC. .

Table 4.6 shows the classification performance of $LEAD$, OE and OC when 90% of the instances in S are labeled. But in all dataset categories $LEAD$ outperforms both OE and OC. There is one case with the F_1-score measure where OE is better than $LEAD$: for SynT-DN dataset. But the difference between those values are quite insignificant (1%). $LEAD$ exhibits very good performance for scene and rcv1v2(subset1) dataset. The overall performance of $LEAD$ for the small datasets are quite noteworthy. As the datasets size increases, there has been a drop in accuracy but not too alarming. For large datasets the accuracy of $LEAD$ is lower compared to the other data categories.

As can be seen from the results, $LEAD$ performs overall best under all of the measures of predictive

Table 4.6: Comparison of classification performance for $P = 90$.

Dataset	Approach	*Accuracy*	Macro-F_1	$F1$-score
20 Ng	LD	**0.6982**	**0.7259**	**0.7439**
	OE	0.5420	0.5683	0.5908
	OC	0.5076	0.5294	0.5531
corel5k	LD	**0.5619**	**0.5672**	**0.5928**
	OE	0.2916	0.2943	0.3277
	OC	0.2410	0.2659	0.2983
enron	LD	**0.5496**	**0.5716**	**0.5901**
	OE	0.3876	0.4219	0.4498
	OC	0.3521	0.3967	0.4254
mediamill	LD	**0.4273**	**0.4627**	**0.4952**
	OE	0.3511	0.3487	0.3976
	OC	0.2810	0.3102	0.3299
rcv1v2(s1)	LD	**0.8532**	**0.8694**	**0.8891**
	OE	0.7931	0.8126	0.8401
	OC	0.7257	0.7611	0.7988
scene	LD	**0.7738**	**0.7791**	**0.7903**
	OE	0.6442	0.6619	0.6999
	OC	0.6318	0.6459	0.6784
tmc2007	LD	**0.6047**	**0.6347**	**0.7029**
	OE	0.5211	0.5371	0.5923
	OC	0.4851	0.5003	0.5497
yeast	LD	**0.6136**	**0.6497**	**0.6597**
	OE	0.3608	0.4010	0.4112
	OC	0.3846	0.4171	0.4398
slashdot	LD	**0.6697**	**0.6569**	**0.6821**
	OE	0.4986	0.5197	0.5477
	OC	0.5155	0.5316	0.5872
imdb	LD	**0.4637**	**0.4966**	**0.5273**
	OE	0.4172	0.4210	0.4788
	OC	0.2578	0.2996	0.3629
SynT-DN	LD	**0.2471**	**0.2589**	0.2863
	OE	0.2285	0.2306	**0.2982**
	OC	0.1698	0.1727	0.1893

performance, starting from very limited amount of labeled data to almost completely labeled data. Among the baseline methods, OE has a better performance than OC for medium and large datasets. It is because E_aHT_{PS} uses multi-label Hoeffding Trees. But Hoeffding trees are rather conservative learners and need a considerable number of examples to identify the best split point for a node. As predictive performance is averaged over evaluation windows, initial low performance can have an overall negative impact on figures. OC outperforms OE in scene and slashdot datasets and shows comparable performance in enron and yeast dataset. $LEAD$ outperforms both in all the datasets. However, as datasets get larger OE's performance improves and for the largest dataset SynT-DN, OE and $LEAD$'s predictive performance are somewhat closer.

To corroborate the aforementioned observations, we selected two popular statistical methods to show the statistical differences among the obtained results. For a pair-wise comparison between $LEAD$

Table 4.7: Wilcoxon signed rank test summary between $LEAD$ and baseline methods for standard data-sets

		OE				OC			
P	**Algorithm**	R^+	R^-	**Hypothesis**($\alpha = 0.05$)	**p-value**	R^+	R^-	**Hypothesis**($\alpha = 0.05$)	**p-value**
40	*Accuracy*	66	0	Rejected for $LEAD$	0.003346	66	0	Rejected for $LEAD$	0.003346
	Macro-F_1	63	3	Rejected for $LEAD$	0.003346	66	0	Rejected for $LEAD$	0.003346
	F_1-Score	66	0	Rejected for $LEAD$	0.003346	66	0	Rejected for $LEAD$	0.003346
60	*Accuracy*	66	0	Rejected for $LEAD$	0.003346	66	0	Rejected for $LEAD$	0.003346
	Macro-F_1	66	0	Rejected for $LEAD$	0.003346	66	0	Rejected for $LEAD$	0.003346
	F_1-Score	65	1	Rejected for $LEAD$	0.004439	66	0	Rejected for $LEAD$	0.003346
80	*Accuracy*	66	0	Rejected for $LEAD$	0.003346	66	0	Rejected for $LEAD$	0.003346
	Macro-F_1	66	0	Rejected for $LEAD$	0.003346	66	0	Rejected for $LEAD$	0.003346
	F_1-Score	66	0	Rejected for $LEAD$	0.003346	66	0	Rejected for $LEAD$	0.003346
90	*Accuracy*	66	0	Rejected for $LEAD$	0.003346	66	0	Rejected for $LEAD$	0.003346
	Macro-F_1	66	0	Rejected for $LEAD$	0.003346	66	0	Rejected for $LEAD$	0.003346
	F_1-Score	65	1	Rejected for $LEAD$	0.004439	66	0	Rejected for $LEAD$	0.003346

and baseline methods, Wilcoxon signed rank test [46] was used. To detect significant statistical difference among multiple algorithms, we used the Iman-Davenport test [47], which is actually an extension of the Friedman test [49]. If the Iman-Davenport test indicated a significant difference, we applied Holm's post hoc analysis [48] to find out which algorithm(s) is(are) actually distinctive. The test results are summarised in Table 4.7–4.10. The tests were conducted on all the classification performance measures and across four different percentage of labeled data ($P = 40, 60, 80, 90$).

Table 4.7 shows the summary of Wilcoxon signed rank test on the *Accuracy*, Macro-F_1 and F_1-score results obtained from datasets given in Table 4.1. Here, R^+ corresponds to the sum of ranks for $LEAD$ and R^- for the compared algorithm. This same notation is used throughout this thesis. These results show that null hypothesis has been rejected in favor of $LEAD$ against all the compared baseline methods with a significance level 0.05. In fact the associated p-values indicate that even a significance level 0.004 would have resulted in the rejection of null hypothesis in favor of $LEAD$. Also we can see from the results, our approach performs very well even with lack of labeled data. Even if the amount of labeled data increases in the stream, $LEAD$ succeeds to maintain better performance than the baseline methods.

The results of the Iman-Davenport test and Holm's post hoc analysis for *Accuracy* are presented in Table 4.8 where these tests were carried out within a particular percentage of labeled data (P) considering $LEAD$ present each time. Here, Iman-Davenport test has rejected the null hypothesis for all the baseline methods. That is why we continued for Holm's post hoc analysis for all of them. The Holm's post hoc analysis also rejected the null hypothesis in favor of $LEAD$ against all the algorithms

Table 4.8: Iman Davenport test and Holm's post hoc test summary for *Accuracy* between $LEAD$ and baseline methods for standard data-sets with $\alpha = 0.05$

Accuracy								
		Holm's post hoc test						
P	**Iman-Davenport test** ($\alpha = 0.05$)	**Algorithm**	**k**	**i**	**z**	**p-value**	$(\frac{\alpha}{k-i})$	**Hypothesis**
40	$\chi^2 = 18.73$, $F_f = 57.22$, $p = 5.31e-9$, Rejected	OE	3	2	-2.77	5.58e-3	0.050	Rejected
		OC	3	1	-4.26	2.01e-5	0.025	Rejected
60	$\chi^2 = 18.73$, $F_f = 57.22$, $p = 5.31e-9$, Rejected	OE	3	2	-2.77	5.58e-3	0.050	Rejected
		OC	3	1	-4.26	2.01e-5	0.025	Rejected
80	$\chi^2 = 18.73$, $F_f = 57.22$, $p = 5.31e-9$, Rejected	OE	3	2	-2.77	5.58e-3	0.050	Rejected
		OC	3	1	-4.26	2.01e-5	0.025	Rejected
90	$\chi^2 = 18.73$, $F_f = 57.22$, $p = 5.31e-9$, Rejected	OE	3	2	-2.77	5.58e-3	0.050	Rejected
		OC	3	1	-4.26	2.01e-5	0.025	Rejected

Table 4.9: Iman Davenport test and Holm's post hoc test summary for Macro-F_1 between $LEAD$ and baseline methods for standard data-sets with $\alpha = 0.05$

Macro-F_1								
		Holm's post hoc test						
P	**Iman-Davenport test** ($\alpha = 0.05$)	**Algorithm**	**k**	**i**	**z**	**p-value**	$(\frac{\alpha}{k-i})$	**Hypo**
40	$\chi^2 = 14.73$, $F_f = 20.25$, $p = 1.56e-5$, Rejected	OE	3	2	-1.92	5.50e-2	0.050	Rejected
		OC	3	1	-3.84	1.24e-4	0.025	Rejected
60	$\chi^2 = 17.63$, $F_f = 40.42$, $p = 9.42e-8$, Rejected	OE	3	2	-2.98	2.8e-3	0.050	Rejected
		OC	3	1	-4.05	5.10e-5	0.025	Rejected
80	$\chi^2 = 18.73$, $F_f = 57.22$, $p = 5.31e-9$, Rejected	OE	3	2	-2.77	5.58e-3	0.050	Rejected
		OC	3	1	-4.26	2.01e-5	0.025	Rejected
90	$\chi^2 = 18.73$, $F_f = 57.22$, $p = 5.31e-9$, Rejected	OE	3	2	-2.77	5.58e-3	0.050	Rejected
		OC	3	1	-4.26	2.01e-5	0.025	Rejected

for different percentage of labeled data.

The results of the Iman-Davenport test and Holm's post hoc analysis for Macro-F_1 are presented in Table 4.9 where these tests were carried out within a particular percentage of labeled data (P) considering $LEAD$ present each time. Here, Iman-Davenport test has rejected the null hypothesis for all the baseline methods. That is why we continued for Holm's post hoc analysis for all of them. The Holm's post hoc analysis also rejected the null hypothesis in favor of $LEAD$ against all the algorithms for different percentage of labeled data.

The results of the Iman-Davenport test and Holm's post hoc analysis for F_1-score are presented in Table 4.10 where these tests were carried out within a particular percentage of labeled data (P) considering $LEAD$ present each time. Here, Iman-Davenport test has rejected the null hypothesis for all the baseline methods. That is why we continued for Holm's post hoc analysis for all of them. The Holm's post hoc analysis also rejected the null hypothesis in favor of $LEAD$ against all the algorithms for different percentage of labeled data.

Table 4.10: Iman Davenport test and Holm's post hoc test summary for F_1-score between $LEAD$ and baseline methods for standard data-sets with $\alpha = 0.05$

F_1-Score								
		Holm's post hoc test						
P	**Iman-Davenport test** ($\alpha = 0.05$)	**Algorithm**	**k**	**i**	**z**	**p-value**	$(\frac{\alpha}{k-i})$	**Hypo**
40	$\chi^2 = 17.63$, $F_f = 40.42$, $p = 9.42e-8$, Rejected	OE	3	2	-2.98	2.8e-3	0.050	Rejected
		OC	3	1	-4.05	5.10e-5	0.025	Rejected
60	$\chi^2 = 15.27$, $F_f = 22.70$, $p = 7.1e-6$, Rejected	OE	3	2	-2.56	1.05e-2	0.050	Rejected
		OC	3	1	-3.84	1.24e-4	0.025	Rejected
80	$\chi^2 = 18.73$, $F_f = 57.22$, $p = 5.31e-9$, Rejected	OE	3	2	-2.77	5.58e-3	0.050	Rejected
		OC	3	1	-4.26	2.01e-5	0.025	Rejected
90	$\chi^2 = 16.54$, $F_f = 30.33$, $p = 8.78e-7$, Rejected	OE	3	2	-2.35	1.90e-2	0.050	Rejected
		OC	3	1	-4.05	5.10e-5	0.025	Rejected

Table 4.11: Performance improvement of $LEAD$ over OE and OC. Here, $LEAD = L$, $OE = E$ and $OC = C$

Dataset	$\mathbf{PI^{40}_{(L,E)}}$	$\mathbf{PI^{40}_{(L,C)}}$	$\mathbf{PI^{60}_{(L,E)}}$	$\mathbf{PI^{60}_{(L,C)}}$	$\mathbf{PI^{80}_{(L,E)}}$	$\mathbf{PI^{80}_{(L,C)}}$	$\mathbf{PI^{90}_{(L,E)}}$	$\mathbf{PI^{90}_{(L,C)}}$
20 Ng	17.22	31.35	20.05	28.63	20.95	28.09	22.37	27.30
corel5k	50.83	55.90	50.11	58.92	49.69	58.39	48.10	57.11
enron	32.54	42.76	31.42	42.42	33.11	39.69	29.48	35.94
mediamill	35.73	45.21	31.34	43.82	22.05	35.53	17.83	34.24
rcv1v2(s1)	2.16	11.33	7.82	14.83	8.52	15.26	7.04	14.94
scene	16.38	25.27	16.72	18.28	17.27	19.51	16.75	18.35
tmc2007	5.07	15.77	7.15	17.58	10.34	20.14	17.13	19.78
yeast	37.74	31.25	37.82	32.12	39.38	33.79	41.20	37.32
slashdot	24.34	20.38	24.97	20.80	29.78	26.42	25.55	23.03
imdb	27.83	38.98	34.52	39.41	19.32	40.80	10.03	44.40
SynT-DN	21.36	49.62	20.43	45.71	11.28	37.27	7.53	31.28

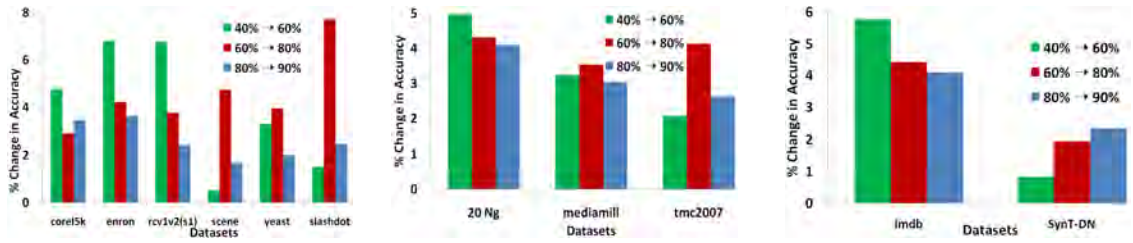
The lower significance level of the statistical tests can be further substantiated by depicting percentage of improvement of accuracy of $LEAD$ over the baseline methods. We denote the improvement as $PI^P_{A,B}$, i.e., the percentage increase/decrease in the accuracy from method A to B for P% labeled data. The results have been shown in Table 4.11. A positive value indicates increase in accuracy whereas a negative value indicates decrease in accuracy. From the results it can be seen that, for all the datasets, $LEAD$ exhibits superior performance in terms of accuracy for all the different percentage of labeled data.

4.5.2 Effect of labeled data

Now, we analyze the performance of $LEAD$, OE and OC with the change of P, i.e. percentage of labeled data in the stream. Three cases of change in P is considered. They are: (1) when amount of labeled data is changed from 40% to 60%, (2) when amount of labeled data is changed from 60% to 80% and (3) when amount of labeled data is changed from 40% to 60%. As the amount of labeled data increases in the stream, the performance of the classifiers should also increase. As more labeled data are available, the classifiers will be trained more comprehensively. As a result, the classifiers will reflect the concept of the data much better. This general observation is supported in Figure 4.1, 4.2 and 4.3. In general, for all the methods, *Accuracy* increases most for the last two cases of change in P .

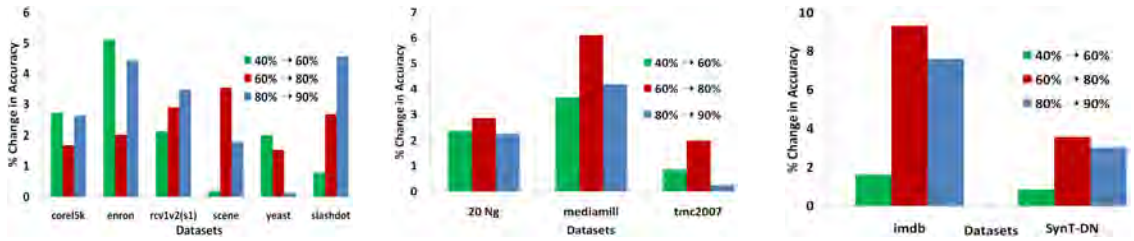
Figure 4.1 shows the change (increase/decrease) in *Accuracy* of $LEAD$ with the increase of labeled data in the stream. For smaller datasets like corel5k, enron and rcv1v2(subset1), there is a higher increase in accuracy as the amount of labeled data is changed from 40% to 60%. For the other three smaller datasets, accuracy increases the most when amount of labeled data is changed from 60% to 80%. The increase in accuracy is steady for all the small datasets when amount of labeled data is changed from 80% to 90%. For medium datasets there is a significant increase in accuracy for the first two cases of change in P (40-60 and 60-80). For one of the large datasets (SynT-DN), change in accuracy is not quite high for any of the three cases. On the other hand, for imdb datasets there is a considerable rise in accuracy for all three cases. As can be seen, the increase of *Accuracy* of $LEAD$ does not depend on the size of the dataset. So, there is no specific trend in the *Accuracy* of $LEAD$.

Figure 4.2 shows the change (increase/decrease) in *Accuracy* of OE with the increase of labeled data in the stream. For smaller datasets like corel5k, enron and yeast, there is a higher increase in accuracy as the amount of labeled data is changed from 40% to 60%. For the two of the smaller



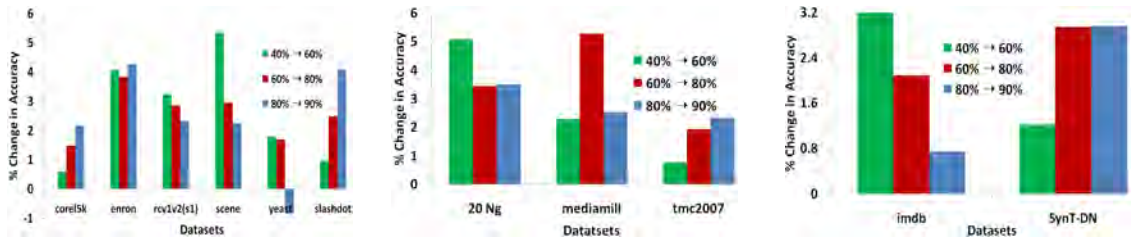
(a) Small Datasets (b) Medium Datasets (c) Large Datasets

Figure 4.1: Change in *Accuracy* with P for $LEAD$.



(a) Small Datasets (b) Medium Datasets (c) Large Datasets

Figure 4.2: Change in *Accuracy* with P for *OE*.



(a) Small Datasets (b) Medium Datasets (c) Large Datasets

Figure 4.3: Change in *Accuracy* with P for *OC*.

datasets, slashdot and rcv1v2(subset1) accuracy increases the most when amount of labeled data is changed from 80% to 90%. The increase in accuracy is steady for all the small datasets when amount of labeled data is changed from 60% to 80%. But for scene dataset, the increase in accuracy for this case exceeds the other two cases. For medium datasets there is a significant increase in accuracy for the first two cases of change in P (40-60 and 60-80). But the increase in accuracy is highest when the amount of labeled data is changed from 60% to 80%. For one of the large datasets (SynT-DN), change in accuracy is not quite high for any of the three cases. On the other hand, for imdb datasets there is a considerable rise in accuracy for the last two cases. As discussed in earlier, *OE* exhibits better performance for larger dataset. Figure 4.2 supports this observation. For four of the medium and large datasets (except tmc2007), there is a significant increase in *Accuracy* of *OE* with increasing P. But *LEAD* as a consistent increase in *Accuracy* for such datasets.

Figure 4.3 shows the change (increase/decrease) in *Accuracy* of *OC* with the increase of labeled data in the stream. For smaller datasets like enron, rcv1v2(subset1), scene and yeast, there is a higher increase in accuracy as the amount of labeled data is changed from 40% to 60%. For the two of the smaller datasets, slashdot and corel5k accuracy increases the most when amount of labeled data is changed from 80% to 90%. The increase in accuracy is steady for all the small datasets when amount of labeled data is changed from 60% to 80%. An interesting case arises for yeast dataset. As the

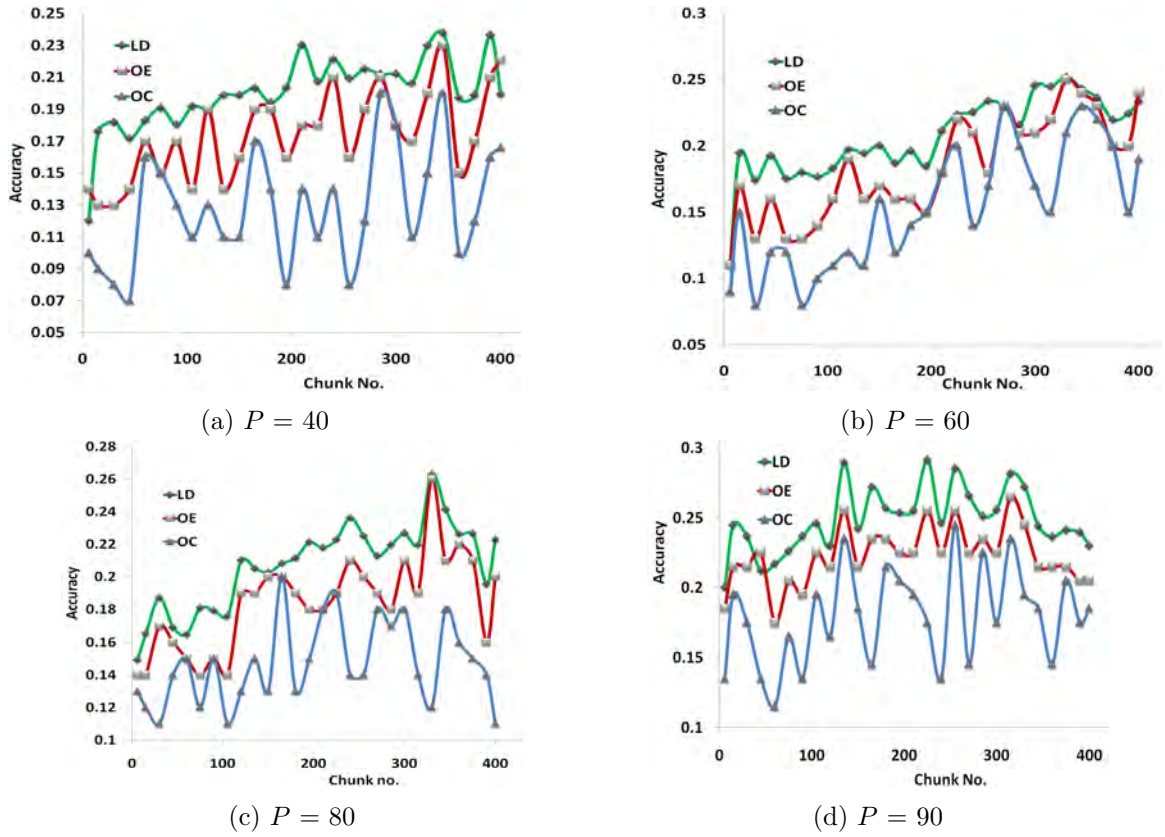
amount of labeled data is changed from 80% to 90% accuracy actually drops in case of yeast dataset (decrease by 0.83%). For medium datasets there is a significant increase in accuracy for the last two cases of change in P (60-80 and 80-90). But the increase in accuracy is highest when the amount of labeled data is changed from 60% to 80%. For one of the large datasets (SynT-DN), change in accuracy is not quite high for any of the three cases. On the other hand, for imdb datasets there is a considerable rise in accuracy for all the cases.

4.5.3 Response to concept Drift

As discussed in Section 4.4.1, the synthetic dataset SynT-D has been designed to simulate concept drift where as Synt-DN simulated both concept drift and concept evolution. Here, we analyze the performance of the competing methods only in the presence of concept drift. The data chunk size B for $LEAD$ and OC are kept same (2500 instances) as the window size of OE. So, *Accuracy* is measured at the same time interval for all the methods. Figure 4.4 shows the *Accuracy* of the methods for different P values over 400 chunks/windows.

From Figure 4.4 it can be seen that, $LEAD$ exhibits better response to concept drift than both OE and OC. Even if the percentage of labled data is changed, $LEAD$ maintains its supremacy in the presence of concept drift. As the value of P is increased, $LEAD$ continues to show better performance than the previous P value. The same case can be observed for both OC and OE also. But none of the methods can supersede $LEAD$.

OC uses the $CBEF$ model for classification. $CBEF$ always replaces one of the existing classifiers with the newly trained classifier. There are two strategies: (1) replace the oldest classifier or (2) replace the classifier that shows the worst classficatiob performance for the current chunk of the data stream. But due to this replacement strategy, one or more class's classifier may be missed in the current model. For example, let instances having some class label l had appeared in D_i but not in D_j. Unfortunately, classifier trained from D_i shows very poor performance for D_j. This means that a concept drift has occurred. Hence, the classifier will be replaced. As a result, $CBEF$ will have no knowledge on class label l's instances. So, in the presence of concept drift, OC has the poorest performance of all the methods. On the other hand, OE deploys an adaptive window (ADWIN) based ensemble of Hoeffding trees. Occasionally, the ADWIN models have a tendency to detect change when there is none; or when it is only temporary. Each time drift is detected, new model is learned that may prematurely replaces relevant classifiers from E_aHT_{PS}. The primary advantage of $LEAD$ is



(a) $P = 40$ (b) $P = 60$

(c) $P = 80$ (d) $P = 90$

Figure 4.4: *Accuracy* of the approaches over chunks/windows for SynT-D.

that, it maintains a label based ensemble of binary classifiers for each of the classes encountered in the stream. A number of binary classifiers (N_m) are maintained in each E_l. Each time a new model is learned instead of replacing the entire E_l, only one of the classifiers in the ensemble is replaced. So, an already leaned concept is never omitted and makes $LEAD$ robust in the presence of drift.

Since, $LEAD$ succeeds to avoid both forced and premature reconstruction of classifiers, its performance is better than the baseline methods. So, it can be safely concluded that, in the presence of concept evolution and class recurrence along with concept drift, $LEAD$ will exhibit better performance than both OE and OC. The performance of $LEAD$ compared to both OE and OC for SynT-DN dataset supports this claim (Table 4.3, 4.4, 4.5 and 4.6).

4.5.4 Novel class detection performance

Table 4.12–4.15 compares the novel class detection performance of $LEAD$, OE and OC. The results were compared with four different percentage of labeled data in the stream ($P = 40, 60, 80, 90$).

PF_{novel} indicates the false alarm rate which is a measure of a classifier's ability to separate novel class instances from existing class instances. As mentioned in Section 4.4.2, there is a correlation between PF_{novel} and AT_{novel}. So, the AT_{novel} cannot alone provide any significant insight on the novelty detection performance of the methods. First, we will analyze the false alarm detection performance of the competing methods. Then we will observe the correlation between PF_{novel} and AT_{novel} for all the methods and analyze the behavior of $LEAD$ for the observed values.

Table 4.12 shows the novel class detection performance of $LEAD$, OE and OC when 40% of the instances in S are labeled. $LEAD$ exhibits very good performance for scene and yeast dataset. The overall performance of $LEAD$ for the datasets slashdot and rcv1v2(subset1) is quite noteworthy. But for corel5k and enron $LEAD$ has a higher false alarm rate than the other smaller datasets. Even OE and OC shows better performance for these two datasets than $LEAD$. For two of the medium datasets 20Ng and mediamill, $LEAD$ as a higher false alarm rate than the other dataset tmc2007. In case of large dataset, the performance of $LEAD$ is not consistent. For imdb dataset, $LEAD$'s false alarm rate is high. On the other hand, for SynT-DN dataset $LEAD$ has a very low false alarm rate.

Table 4.13 shows the novel class detection performance of $LEAD$, OE and OC when 60% of the instances in S are labeled. $LEAD$ exhibits very good performance for scene and yeast dataset. The overall performance of $LEAD$ for the datasets slashdot, enron and rcv1v2(subset1) is quite noteworthy. But for corel5k $LEAD$ has a higher false alarm rate than the other smaller datasets. Even OE and OC shows better performance for this dataset than $LEAD$. For one of the medium datasets mediamill, $LEAD$ as a higher false alarm rate than the other two datasets. For tmc2007, $LEAD$ exhibits very good performance. In case of large datasets, the performance of $LEAD$ is not consistent. For imdb dataset, $LEAD$'s false alarm rate is high. On the other hand, for SynT-DN dataset $LEAD$ has a very low false alarm rate.

Table 4.14 shows the novel class detection performance of $LEAD$, OE and OC when 80% of the instances in S are labeled. $LEAD$ exhibits very good performance for scene, slashdot, rcv1v2(subset1) and yeast dataset. The overall performance of $LEAD$ for the datasets enron and corel5k is quite noteworthy. For the medium datasets, performance of $LEAD$ is mixed. For one of the medium datasets mediamill, $LEAD$ as a higher false alarm rate than the other two datasets. For tmc2007, $LEAD$ exhibits very good performance. On other hand, for 20Ng dataset, $LEAD$'s performance is satisfactory. In case of large datasets, the performance of $LEAD$ is not consistent. For imdb dataset, $LEAD$'s false alarm rate is high. On the other hand, for SynT-DN dataset $LEAD$ has a very low

Table 4.12: Comparison of novelty detection performance for $P = 40$.

Dataset	Approach	PF_{novel}	AT_{novel}
20 Ng	LD	0.3223	0.8782
	OE	0.5719	1.0826
	OC	0.6196	1.1780
corel5k	LD	0.3712	0.9128
	OE	0.0727	0.1492
	OC	0.0493	0.1609
enron	LD	0.5247	1.0209
	OE	0.1443	0.1869
	OC	0.0914	0.1635
mediamill	LD	0.4149	0.8694
	OE	0.5926	1.0020
	OC	0.6286	1.0807
rcv1v2(s1)	LD	0.1671	0.9517
	OE	0.1967	0.8932
	OC	0.2243	0.9120
scene	LD	0.0110	0.9048
	OE	0.2981	0.8945
	OC	0.3386	0.8525
tmc2007	LD	0.0617	0.9586
	OE	0.3109	0.9123
	OC	0.3326	0.8726
yeast	LD	0.0241	0.9672
	OE	0.4262	0.6132
	OC	0.4598	0.6962
slashdot	LD	0.1518	0.8439
	OE	0.4189	0.7524
	OC	0.4372	0.7193
imdb	LD	0.4351	0.9256
	OE	0.6872	0.5798
	OC	0.7129	0.5635
SynT-DN	LD	0.0309	0.9794
	OE	0.2892	0.6152
	OC	0.2739	0.5931

false alarm rate.

Table 4.15 shows the novel class detection performance of $LEAD$, OE and OC when 90% of the instances in S are labeled. $LEAD$ exhibits very good performance for scene, slashdot, rcv1v2(subset1), enron and yeast dataset. The overall performance of $LEAD$ for the datasets corel5k is quite noteworthy. For the medium datasets, performance of $LEAD$ is mixed. For one of the medium datasets mediamill, $LEAD$ as a higher false alarm rate than the other two datasets. For tmc2007, $LEAD$ exhibits very good performance. On other hand, for 20Ng dataset, $LEAD$'s performance is satisfactory. In case of large datasets, the performance of $LEAD$ is not consistent. For imdb dataset, $LEAD$'s false alarm rate is high. On the other hand, for SynT-DN dataset $LEAD$ has a very low false alarm rate.

From the values of the false alarm rate observed for different datasets across different percentage

Table 4.13: Comparison of novelty detection performance for $P = 60$.

Dataset	Approach	PF_{novel}	AT_{novel}
20 Ng	LD	0.2574	0.9199
	OE	0.3582	0.7431
	OC	0.3826	0.7298
corel5k	LD	0.2819	1.0000
	OE	0.1896	0.2278
	OC	0.1711	0.2537
enron	LD	0.1760	0.8724
	OE	0.2541	0.8829
	OC	0.2909	0.8516
mediamill	LD	0.3754	0.9279
	OE	0.4872	0.7653
	OC	0.4528	0.7192
rcv1v2(s1)	LD	0.1198	0.9907
	OE	0.1676	0.8978
	OC	0.1927	0.9322
scene	LD	0.0140	0.9217
	OE	0.3134	0.9143
	OC	0.3078	0.8621
tmc2007	LD	0.0253	0.9715
	OE	0.2876	0.9348
	OC	0.3289	0.9034
yeast	LD	0.0269	0.9485
	OE	0.3921	0.8420
	OC	0.4002	0.8091
slashdot	LD	0.1211	0.9247
	OE	0.4341	1.0680
	OC	0.4422	1.1051
imdb	LD	0.3647	0.9178
	OE	0.4471	0.6904
	OC	0.4728	0.6297
SynT-DN	LD	0.0295	0.9920
	OE	0.3472	0.7908
	OC	0.3651	0.7528

of labeled data, it can be summarized that, as the percentage of labeled data is increased the false alarm rate of $LEAD$ decreases. For smaller datasets, the performance continues to improve as $LEAD$ outperforms more and more smaller datasets. Also, for medium datasets there is a marked improvement in $LEAD$'s performance as P increases. On the other hand, large datasets the although there is an improvement in the performance of $LEAD$. But the scale of improvement is not quite high to take note of.

4.5.5 Analysis of false alarm rates

Figure 4.5–4.8 shows the false alarm rate of the three methods for different values of P across different datasets. In majority of datasets with different P values, $LEAD$ shows lower frequency of false alarms than OE and OC. For corel5k dataset with $P = 40$ and $P = 60$, $LEAD$ exhibits higher number

Table 4.14: Comparison of novelty detection performance for $P = 80$.

Dataset	Approach	PF_{novel}	AT_{novel}
20 Ng	LD	0.1718	1.0000
	OE	0.2956	0.9483
	OC	0.4329	0.7773
corel5k	LD	0.2135	1.0000
	OE	0.3284	0.6675
	OC	0.3381	0.6724
enron	LD	0.1253	1.0000
	OE	0.2240	0.9117
	OC	0.2389	0.8887
mediamill	LD	0.3518	1.0000
	OE	0.3679	0.7927
	OC	0.5826	0.7283
rcv1v2(s1)	LD	0.0942	0.9974
	OE	0.1507	0.9627
	OC	0.2025	0.8967
scene	LD	0.0090	0.9890
	OE	0.2376	0.9367
	OC	0.2578	0.8956
tmc2007	LD	0.0186	0.9883
	OE	0.2238	0.9278
	OC	0.2554	0.9146
yeast	LD	0.0214	0.9978
	OE	0.3763	0.8618
	OC	0.3939	0.8337
slashdot	LD	0.0982	1.0000
	OE	0.2745	0.9381
	OC	0.2983	0.9193
imdb	LD	0.3288	0.9759
	OE	0.3662	0.8129
	OC	0.3970	0.7926
SynT-DN	LD	0.0182	1.0001
	OE	0.1391	0.8752
	OC	0.1446	0.8706

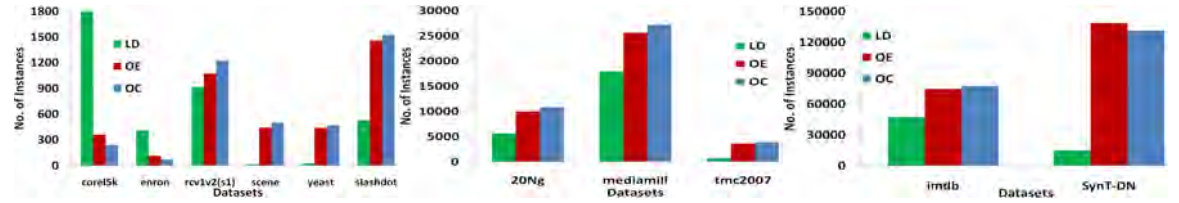
of false alarms than both OE and OC. Also, for enron dataset with $P = 40$, $LEAD$ shows inferior results. In both cases, as the value of P increases, $LEAD$ attains lower false alarm rate than the other two methods. The overall superior performance of $LEAD$ is due to the ability to differentiate recurrent classes from novel classes. Both OE and OC fail to identify class recurrence and hence, treat recurrent class instances as novel class instance. As a result, the false alarm rate increases. But PF_{novel} doest not indicate the novel class detection success.

In this paper, we have coupled PF_{novel} with AT_{novel} to analyze novel class detection performance. Let us consider the case $AT_{novel} << 1$ first. For corel5k dataset ($P = 40$, $N_{existing} = 4951$), both PF_{novel} and AT_{novel} are small for OE and OC. The value of F_{novel} for $LEAD$, OE and OC are 1838, 350 and 244 respectively. The similar trend continues for $P = 60$. The same case arises in enron dataset for OE and OC with $P = 40$. But AT_{novel} is quite small for both OE and OC which

Table 4.15: Comparison of novelty detection performance for $P = 90$.

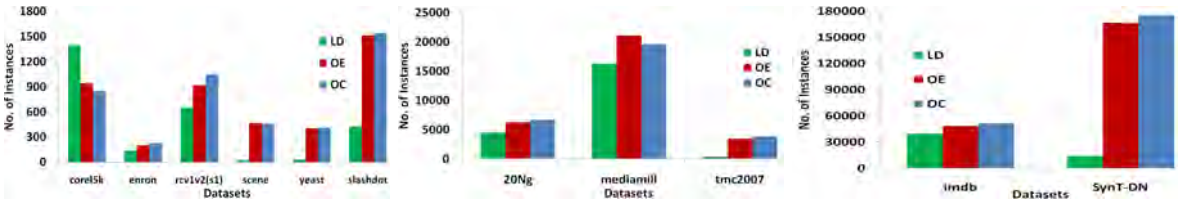
Dataset	Approach	PF_{novel}	AT_{novel}
20 Ng	LD	0.1362	1.0000
	OE	0.2804	0.9573
	OC	0.2982	0.9175
corel5k	LD	0.1719	1.0000
	OE	0.2874	0.8259
	OC	0.3048	0.8169
enron	LD	0.0820	1.0000
	OE	0.1584	0.9378
	OC	0.1712	0.9193
mediamill	LD	0.3162	1.0000
	OE	0.3038	0.8195
	OC	0.3529	0.7712
rcv1v2(s1)	LD	0.0701	0.9863
	OE	0.1286	0.9694
	OC	0.1463	0.9214
scene	LD	0.0096	0.9900
	OE	0.1923	0.9444
	OC	0.2013	0.9246
tmc2007	LD	0.0132	1.0090
	OE	0.1625	0.9378
	OC	0.1318	0.9056
yeast	LD	0.0200	1.0000
	OE	0.3113	0.8858
	OC	0.3423	0.8579
slashdot	LD	0.0734	0.9980
	OE	0.1172	0.9563
	OC	0.1428	0.9265
imdb	LD	0.2719	0.9892
	OE	0.3187	0.8365
	OC	0.3285	0.8108
SynT-DN	LD	0.0176	1.0000
	OE	0.0980	0.9426
	OC	0.1235	0.9292

indicates failure to detect large amount of novel class instance. Interestingly, the value of AT_{novel} for $LEAD$ is quite close to 1 which indicates good novel class detectability. For enron, although AT_{novel} is higher for $LEAD$, large value of PF_{novel} indicates higher false alarm rate, which might be the primary contributor to higher value of AT_{novel} than better novel class detectability. Similar scenario ($AT_{novel} > 1$ and large PF_{novel}) can be seen in case of mediamill ($P = 40$) and slashdot ($P = 60$) datasets for OE and OC. Another case is high value of PF_{novel} and considerably lower value of AT_{novel}. For example, yeast ($P = 40$), slashdot ($P = 40$) and imdb ($P = 40$) for OE and OC. This indicates high false alarm rate with low novel class detectability. Low PF_{novel} with high AT_{novel} indicates lower false alarm rate with better novel class detectability. This scenario can be seen in case of, $LEAD$ for medium and large datasets with ($P = 60, 80$).



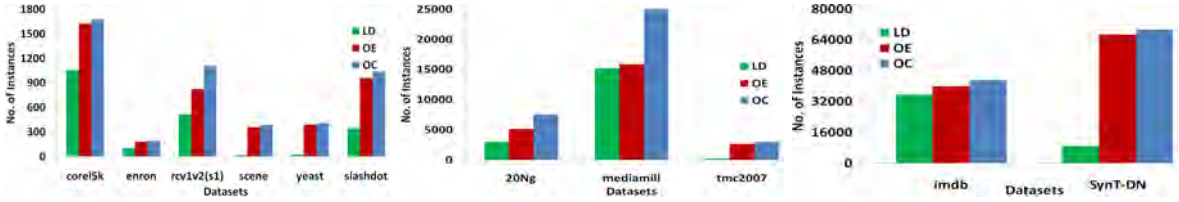
(a) Small (b) Medium (c) Large

Figure 4.5: False alarm frequency of the competing methods for $P = 40$



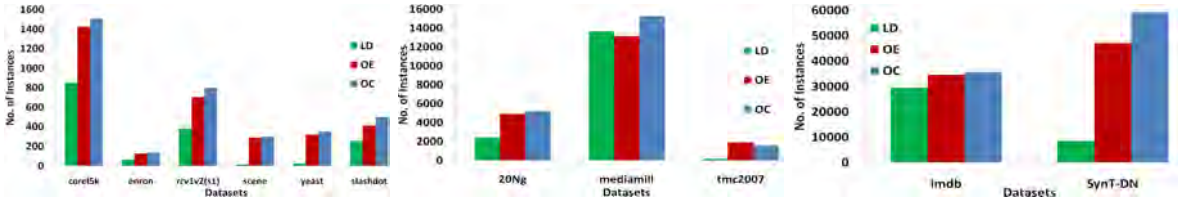
(a) Small (b) Medium (c) Large

Figure 4.6: False alarm frequency of the competing methods for $P = 60$



(a) Small (b) Medium (c) Large

Figure 4.7: False alarm frequency of the competing methods for $P = 80$



(a) Small (b) Medium (c) Large

Figure 4.8: False alarm frequency of the competing methods for $P = 90$

4.5.6 Novelty Detection: LEAD vs OLINDDA

We observe in the evaluation that $LEAD$ outperforms both OE and OC in detecting novel classes for most of the cases. The main reason behind the poorer performance of OE and OC in detecting novel classes is the way OLINDDA detects novel class. OLINDDA makes two strong assumptions about a novel class and normal classes. First, it assumes a spherical boundary (or, convex shape) of the normal model. It updates the radius and centroid of the sphere periodically and declares anything outside the sphere as a novel class if there is evidence of sufficient cohesion among the instances outside the boundary. The assumption that a data class must have a convex/spherical shape is too strict to be maintained for a real-world problem. Second, it assumes that the data density of a novel class must be at least that of the normal class. If a novel class is sparser than the normal class, the instances of that class would never be recognized as a novel class. But in a real-world problem, two different classes may have different data densities. OLINDDA would fail in those cases where any of the assumptions are violated. On the other hand, $LEAD$ does not consider the shape of the class for novelty detection. Rather, it uses a soft clustering model to identify potential novel clusters and then uses cluster validity measure to identify the best clusters from among them. Therefore, $LEAD$ can detect novel classes much more efficiently. Besides, OLINDDA is less sensitive to concept-drift, which results in falsely declaring novel classes when drift occurs in the existing class data. On the other hand, $LEAD$ correctly distinguishes between concept-drift and concept-evolution, avoiding false detection of novel classes in the event of concept-drift. $LEAD$ uses classifier replacement strategy to handle concept drift while using the layered ensemble architecture to identify concept evolution. The performance of OE and OC are somewhat similar for novel class detection as they both use $OLINDDA$.

In general, existing novel class detection techniques have limited applicability since those are similar to one-class classifiers. That is, they assume that there is only one "normal" class, and all other classes are novel. However, our technique is applicable to the more realistic scenario where there are more than one existing classes in the stream. Besides, our novel class detection technique requires neither any specific data distribution nor the classes to have convex shape.

Table 4.16: Response of $LEAD$ for $T_u = 0$

P	Dataset	PF_{novel}	AT_{novel}	P	Dataset	PF_{novel}	AT_{novel}
40	scene	0.4285	0.6992	60	scene	0.3948	0.7361
	20NG	0.5912	0.6089		20NG	0.5572	0.6215
	imdb	0.1497	0.8259		imdb	0.1376	0.8531
P	Dataset	PF_{novel}	AT_{novel}	P	Dataset	PF_{novel}	AT_{novel}
60	scene	0.2710	0.8152	90	scene	0.2109	0.8934
	20NG	0.5136	0.6480		20NG	0.4527	0.6732
	imdb	0.1139	0.9206		imdb	0.1078	0.9154

4.5.7 Parameter Sensitivity (T_u)

According to Table 4.2, for all three types of datasets, the value of T_u is set to some non-zero value. In this section, we will observe the novel class detection performance of $LEAD$ when T_u is set to 0. We have conducted the experiment on three datasets, one from each of the dataset categories. The chosen datasets are: scene, 20NG and imdb. The results are shown in Table 4.16.

When T_u is set to 0, all three datasets exhibit the similar pattern in their novel class detection performance. For all three datasets, as the percentage of labeled data is increased, the value of PF_{novel} continues to decrease. The higher the value of PF_{novel}, the poorer the false alarm rate, i.e., a higher percentage of existing class instances have been identified as novel class instances. On the other hand, the value of AT_{novel} continues to increase. Compared to Table 4.12-4.15, we observe that Table 4.16 exhibits inferior novel class detection performance. So, why such behavior is being exhibited by the $LEAD$ framework? The answer lies in the two parameters T_u and P.

As T_u is set to 0, potential novel instances are not allowed to accumulate in buf_{novel}. Rather, the instances are sent to novelty detector as soon as T_d has elapsed. So, instances arriving in the same chunk are chosen for novelty detection at the same time. On the other hand, setting T_u to a non-zero value allows several chunks' potential novel instances to accumulate in buf_{novel}. As a result, the number of potential novel instances might be lower when $T_u = 0$. In addition to that, when the value of P is lower, most of the instances in the stream are unlabaled. So, the classification of a large number of instances are deferred. As a result, even if $T_u = 0$, a large number of instances are considered for novelty detection. Hence, a lot of existing class instances are identified as novel class instances. So, PF_{novel} will be higher than usual. The existing class instances will also be the primary contributors in the value of AT_{novel}. When the value of P is higher, most of the instances

in the stream are labeled. So, the classification of a small number of instances are deferred. Hence, a small number of instances are considered for novelty detection. As a result, very few existing class instances are identified as novel class instances. So, PF_{novel} will be low when P is large. The value of AT_{novel} will depend on the number of actual novel class instances identified.

The analysis made in the aforementioned paragraph has been corroborated in Table 4.16. For all the datasets, PF_{novel} decreases as P increases. For lower P values, AT_novel is lower. So, $LEAD$ has lower novel class detectability and the a large number of existing class instances contribute to the value of AT_{novel} as they are identified as novel class instance. On the other hand, for higher P values, lower false alarm rate with high AT_{novel} indicates better novel class detectability. So, setting the value of $T_u = 0$ affects the performance of $LEAD$ at a larger scale, when the percentage of labeled data in the stream is lower.

4.6 Summary

This chapter presents the evaluation of $LEAD$'s performance on several multi-label classification problems, including 10 real-world problems and one synthetic problems. We use the synthetic problem to analyze how $LEAD$ responds in the presence of concept drift in the data stream. We modify all the datasets to introduce concept evolution and class recurrence in the stream and analyze both classification and novel class detection performance of $LEAD$ and two other baseline methods. With the evaluation measures used in this thesis, we observe that $LEAD$ has a superior classification performance than the baseline methods. We also propose two new evaluation measures for novel class detection. The interpretation of the values for different datasets indicate that $LEAD$ also outperforms the baseline methods in case of novel class detection. Further analysis on the synthetic dataset reveals $LEAD$'s robustness in the presence of concept drift.

Chapter 5

Conclusion

In this thesis, we have addressed several challenging problems related to multi-label data steam classification. Classifying multi-label data itself is a more challenging task than classifying single label data. When this data assumes the stream behavior, the classification models are faced with several other challenges like concept drifting, concept evolution and class recurrence. Furthermore, real world streaming applications suffer from availability of limited amount of labeled data. Apart from the issue of concept drifting, the existing methods in literature have ignored the rest of the issues related to multi-label data stream classification. To deal with these challenges, we have proposed a batch incremental label based data stream classification framework, *LEAD*. Existing methods assume that the number of class labels in the stream is fixed and hence, the novel class instances are misclassified by the existing techniques. We show how to detect novel class instances automatically even when the classification model is not trained with the novel class instances. Novel class detection becomes more challenging in the presence of concept-drift. Failure to detect class recurrence causes false identification of novel classes.

A new layered ensemble architecture is introduced in *LEAD* to deal with these challenges. The top layer of the ensemble architecture reflects the most recent concept of the data stream whereas the bottom layer represents the older concepts of the stream. As a result, even if instances of older class labels that did not appear in the stream for a long time suddenly appears in the stream the bottom layer will be able to classify those instances. So, *LEAD* addresses the class recurrence issue. Moreover, if an instance cannot be classified by any of the layers in the ensemble architecture, it is considered as a potential novel class instance. Such instances are candidates for novel class detection. So, the layered

approach also helps to differentiate between recurrent and novel class instances which significantly reduces false identification of recurrent class instances as novel class instances. The problem of limited amount of labeled data in the stream is handled by a deferred classification mechanism. We impose two time constraints T_d and T_u on the instances of the data stream. If an instance cannot be classified in the first try, then T_d is the allowable time constraint until which classification of that instance can be deferred. After an instances has been identified as a potential novel instance, attempts are still made to classify the instance until its time stamp exceeds T_u. In the mean time, more labeled data will appear in the stream that may help the newly trained classifiers to classify the previously unclassified instances. Failure to classify an instance within T_u time units leads to considering the instance as a novel instance. Apart from the usual classification performance measures for multi-label data stream classification, we propose two new measures to evaluate the novel class detection performance of a framework. Empirical results on both synthetic and real-world data sets have demonstrated the effectiveness of the *LEAD* framework.

5.1 Future work

In this section, we discuss the research areas that can be ventured in future. The data stream models offer some promising research avenues where our proposed *LEAD* framework can be applied. As mentioned in Chapter 1, there are a lot of real world applications that generate data streams. We can apply *LEAD* for meaningful analysis of those applications. Besides, we also hope to improve the performance of our framework by introducing new methodologies.

5.1.1 Social network analysis

Social network analysis (SNA) is the analysis of social networks, e.g., facebook, twitter etc. It is the mapping and measuring of relationships and flows between people, groups, organizations, computers, URLs, and other connected knowledge entities in the social networks. SNA provides both a visual and a mathematical analysis of human relationships. Social networks are one of the primary sources of data streams. In future, we can apply our *LEAD* framework to analyze various aspects of the social networks, e.g. detecting emerging trends, identifying malicious contents and users etc.

Trend detection

Trend detection is defined as the process of spotting the underlying emerging patterns in a given data. Social networks are one of the primary conduits of information, opinions, and behaviors. They carry news about products, jobs, and various social programs; influence decisions to become made, to contribute socially; and drive political opinions and attitudes toward other groups. In view of this, it is important to understand how beliefs and behaviors of individuals in a social network are influenced by their virtual social setting. The identification of emergent and important topics discussed in social networks as well as other traditional online news sources is crucial for a better understanding of societal concerns. It also helps users to keep abreast of trends without having to surf through vast amounts of shared information. If we consider posts in social networks as instances of data stream, our *LEAD* framework can be deployed to identify the emergence of new trends (concept) in the social network. For example, during the Shahbag movement in Bangladesh, social networks like twitter were flooded with posts containing #shahbag hashtag. Our *LEAD* framework can identify appearance of such trends using the layered ensemble architecture and novelty detector. Moreover, some trends appear periodically, which can be considered as recurrent trends and can easily be identified by *LEAD*.

Topic models

In machine learning and natural language processing, topic model is a type of statistical model for discovering the abstract "topics" that occur in a collection of documents. These models help us develop new ways to search, browse and summarize large archives of texts. In recent years, topic models have been applied in social networks to identify and extract specific topics, e.g., discovering spoilers, identifying malicious contents (e.g., spam, threats etc.) in posts and so forth. We can incorporate *LEAD* in such topic models to address such issues in social networks.

A spoiler is an element of a disseminated summary or description of any piece of fiction that reveals any plot elements which threaten to give away important details concerning the turn of events of a dramatic episode. Typically, the details of the conclusion of the plot, including the climax and ending, are especially regarded as spoiler material. It can also be used to refer to any piece of information regarding any part of a given media that a potential consumer would not want to know beforehand. Because enjoyment of fiction depends a great deal upon the suspense of revealing plot details through standard narrative progression, the prior revelation of how things will turn out

can "spoil" the enjoyment that some consumers of the narrative would otherwise have experienced. Currently, social network posts containing spoilers about media are one of the major concerns for the users. For example, after the airing of each episode of the popular television series "Game of Thrones", some of the users post information regarding the events of that episode which spoils the enjoyment of other users. We can apply *LEAD* as a part of a filter to isolate such posts. Another research problem in social networks analysis is identifying posts containing violent threats, spam information etc. Similar to spoiler detection models, our *LEAD* framework can be incorporated in spam detection models to identify the emergence of such malicious posts in social network.

5.1.2 Performance improvement

In future, we hope to further improve the performance of *LEAD*. One approach can be to parallelize the components of *LEAD* to develop a faster classification framework. Moreover, we can refine the architecture of *LEAD* to make it a more robust framework in the face of the challenges posed by the data stream environment.

Distributed classification framework

A future research avenue can be a distributed classification framework where classification and novel class detection mechanisms can be paralleled to attain better performance. Currently, classification and novel class detection are sequential processes in *LEAD*. In a distributed *LEAD* framework, we can perform these tasks in separate computing resources. A central computing resource (master) where the ensemble architecture may reside will receive the data stream and the classifiers will be trained. Potential novel instances will be then transferred to a secondary computing resource (slave) that will perform novel class detection. The updated results of novelty detection can again be sent to the master. This work share will ensure faster performance of individual resources.

Layered label based ensembles

To further improve the performance of *LEAD*, we may maintain two layers of binary classifiers in each label based ensemble, E_l. Instead of deleting the poorly performing classifier, it can be demoted to the bottom layer of E_l. We can also incorporate a fading function of time to assign lesser weight to much older concepts. Introducing layered labeled based ensembles in the layered ensemble architecture may ensure more robustness in the presence of concept drift.

Bibliography

[1] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

[2] Mohammad M Masud, Qing Chen, Latifur Khan, Charu Aggarwal, Jing Gao, Jiawei Han, and Bhavani Thuraisingham. Addressing concept-evolution in concept-drifting data streams. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 929–934. IEEE, 2010.

[3] Mohammad M Masud, Clay Woolam, Jing Gao, Latifur Khan, Jiawei Han, Kevin W Hamlen, and Nikunj C Oza. Facing the reality of data stream classification: coping with scarcity of labeled data. *Knowledge and information systems*, 33(1):213–244, 2012.

[4] Xiangnan Kong and PS Yu. An ensemble-based approach to fast classification of multi-label data streams. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, pages 95–104. IEEE, 2011.

[5] Eleftherios Spyromitros Xioufis, Myra Spiliopoulou, Grigorios Tsoumakas, and Ioannis Vlahavas. Dealing with concept drift and class imbalance in multi-label stream classification. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*, pages 1583–1588. AAAI Press, 2011.

[6] Wei Qu, Yang Zhang, Junping Zhu, and Qiang Qiu. Mining multi-label concept-drifting data streams using dynamic classifier ensemble. In *Advances in Machine Learning*, pages 308–321. Springer, 2009.

[7] Peng Wang, Peng Zhang, and Li Guo. Mining multi-label data streams using ensemble-based active learning. In *SDM*, pages 1131–1140. SIAM, 2012.

[8] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

[9] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, 40(7):2038–2048, 2007.

[10] Jesse Read, Albert Bifet, Geoffrey Holmes, and Bernhard Pfahringer. Efficient multi-label classification for evolving data streams. 2010.

[11] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM, 2000.

[12] Amanda Clare and Ross D King. Knowledge discovery in multi-label phenotype data. In *Principles of data mining and knowledge discovery*, pages 42–53. Springer, 2001.

[13] Jesse Read, Bernhard Pfahringer, and Geoffrey Holmes. Multi-label classification using ensembles of pruned sets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 995–1000. IEEE, 2008.

[14] Jesse Read, Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Scalable and efficient multi-label classification for evolving data streams. *Machine Learning*, 88(1-2):243–272, 2012.

[15] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

[16] Shantanu Godbole and Sunita Sarawagi. Discriminative methods for multi-labeled classification. In *Advances in Knowledge Discovery and Data Mining*, pages 22–30. Springer, 2004.

[17] Hanady Abdulsalam, David B Skillicorn, and Patrick Martin. Classifying evolving data streams using dynamic streaming random forests. In *Database and Expert Systems Applications*, pages 643–651. Springer, 2008.

[18] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[19] Lidia Ceriani and Paolo Verme. The origins of the gini index: extracts from variabilità e mutabilità (1912) by corrado gini. *The Journal of Economic Inequality*, 10(3):421–443, 2012.

[20] Ido Dagan and Sean P Engelson. Committee-based sampling for training probabilistic classifiers. In *ICML*, volume 95, pages 150–157, 1995.

[21] Kyupil Yeon, Moon Sup Song, Yongdai Kim, Hosik Choi, and Cheolwoo Park. Model averaging via penalized regression for tracking concept drift. *Journal of Computational and Graphical Statistics*, 19(2), 2010.

[22] Jinseog Kim, Yuwon Kim, and Yongdai Kim. A gradient-based optimization algorithm for lasso. *Journal of Computational and Graphical Statistics*, 17(4), 2008.

[23] Eleftherios Spyromitros, Grigorios Tsoumakas, and Ioannis Vlahavas. An empirical study of lazy multilabel classification algorithms. In *Artificial Intelligence: Theories, Models and Applications*, pages 401–406. Springer, 2008.

[24] Hendrik Blockeel, Luc De Raedt, and Jan Ramon. Top-down induction of clustering trees. *arXiv preprint cs/0011032*, 2000.

[25] Min-Ling Zhang and Zhi-Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *Knowledge and Data Engineering, IEEE Transactions on*, 18(10):1338–1351, 2006.

[26] Min-Ling Zhang. Ml-rbf: Rbf neural networks for multi-label learning. *Neural Processing Letters*, 29(2):61–74, 2009.

[27] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD08)*, pages 30–44, 2008.

[28] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333–359, 2011.

[29] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. In *Data mining and knowledge discovery handbook*, pages 667–685. Springer, 2010.

[30] Johannes Fürnkranz. Round robin classification. *The Journal of Machine Learning Research*, 2:721–747, 2002.

[31] Sang-Hyeun Park and Johannes Fürnkranz. Efficient pairwise classification. In *Machine Learning: ECML 2007*, pages 658–665. Springer, 2007.

[32] Eneldo Loza Mencía, Sang-Hyeun Park, and Johannes Fürnkranz. Efficient voting prediction for pairwise multilabel classification. *Neurocomputing*, 73(7):1164–1176, 2010.

[33] Grigorios Tsoumakas and Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *Machine Learning: ECML 2007*, pages 406–417. Springer, 2007.

[34] Dragi Kocev. Ensembles for predicting structured outputs. *Informatica (03505596)*, 36(1), 2012.

[35] S Guha, A Meyerson, N Mishra, R Motwani, and L O'Callaghan. Clustering data streams: Theory and practice. *IEEE TKDE*, 15(3):515–528, 2003.

[36] Paul E Utgoff. Incremental induction of decision trees. *Machine learning*, 4(2):161–186, 1989.

[37] Johannes Gehrke, Venkatesh Ganti, Raghu Ramakrishnan, and Wei-Yin Loh. Boat-optimistic decision tree construction. In *SIGMOD Conference*, volume 12, 1999.

[38] Carlotta Domeniconi and Dimitrios Gunopulos. Incremental support vector machine construction. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 589–592. IEEE, 2001.

[39] Giorgio Fumera, Fabio Roli, and Alessandra Serrau. A theoretical analysis of bagging as a linear combination of classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(7):1293–1299, 2008.

[40] Eduardo J Spinosa, André Ponce de Leon F de Carvalho, and João Gama. Cluster-based novel concept detection in data streams applied to intrusion detection in computer networks. In *Proceedings of the 2008 ACM Symposium on Applied computing*, pages 976–980. ACM, 2008.

[41] James C Bezdek, Robert Ehrlich, and William Full. Fcm: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2):191–203, 1984.

[42] Raghuram Krishnapuram and James M Keller. A possibilistic approach to clustering. *Fuzzy Systems, IEEE Transactions on*, 1(2):98–110, 1993.

[43] James C Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Kluwer Academic Publishers, 1981.

[44] Mingrui Zhang, Wei Zhang, Hugues Sicotte, and Ping Yang. A new validity measure for a correlation-based fuzzy c-means clustering algorithm. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, pages 3865–3868. IEEE, 2009.

[45] Xuanli Lisa Xie and Gerardo Beni. A validity measure for fuzzy clustering. *IEEE Transactions on pattern analysis and machine intelligence*, 13(8):841–847, 1991.

[46] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1(6):80–83, 1945.

[47] David J Sheskin. *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.

[48] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.

[49] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[50] Hendrik Blockeel, Luc De Raedt, and Jan Ramon. Top-down induction of clustering trees. *arXiv preprint cs/0011032*, 2000.

[51] Andr Elisseeff and Jason Weston. A kernel method for multi-labelled classification. In *In Advances in Neural Information Processing Systems 14*, pages 681–687. MIT Press, 2001.

[52] Matthew R Boutell, Jiebo Luo, Xipeng Shen, and Christopher M Brown. Learning multi-label scene classification. *Pattern recognition*, 37(9):1757–1771, 2004.

[53] Pinar Duygulu, Kobus Barnard, Joao FG de Freitas, and David A Forsyth. Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary. In *Computer VisionECCV 2002*, pages 97–112. Springer, 2002.

[54] Cees GM Snoek, Marcel Worring, Jan C Van Gemert, Jan-Mark Geusebroek, and Arnold WM Smeulders. The challenge problem for automated detection of 101 semantic concepts in multi-

media. In *Proceedings of the 14th annual ACM international conference on Multimedia*, pages 421–430. ACM, 2006.

[55] Yan-Shi Dong and Ke-Song Han. Boosting svm classifiers by ensemble. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1072–1073. ACM, 2005.

[56] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *Machine learning: ECML 2004*, pages 217–226. Springer, 2004.

[57] Ashok N Srivastava and Brett Zane-Ulman. Discovering recurring anomalies in text reports regarding complex space systems. In *Aerospace Conference, 2005 IEEE*, pages 3853–3862. IEEE, 2005.

[58] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333–359, 2011.

[59] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[60] Bishan Yang, Jian-Tao Sun, Tengjiao Wang, and Zheng Chen. Effective multi-label active learning for text classification. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 917–926. ACM, 2009.

[61] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.