Modeling Recurring Concepts in Single-label and Multi-label Streams

A thesis submitted for the degree of

Doktor der Naturwissenschaften

at the Department of Physics, Mathematics and Computer Science
at the *Johannes Gutenberg-Universität*
in Mainz

Zahra Ahmadi
born in Tehran,IRAN

Mainz, 10.10.2019

Examination date: 11.12.2019 D77

"In the past, censorship worked by blocking the flow of information. In the twenty-first century, censorship works by flooding people with irrelevant information. [...] *In ancient times having power meant having access to data. Today having power means knowing what to ignore."*

Yuval Noah Harari, Homo Deus: A History of Tomorrow

Abstract

Today, we have access to a vast amount of data in the forms of images, speech signals, structured and unstructured texts, and sensor-based signals. Our digital universe is growing quickly. Statistics indicate that 500 million tweets are posted every day. 65 billion messages are transferred on *WhatsApp* per day. 294 billion emails are sent daily via different platforms. Each self-driving car creates 4 terabytes of data per day. According to a study by *Digital Universe*, the amount of data produced by humans and machines will exceed 44 billion terabytes by 2020. This means that there will be 5,200 gigabyte of data for every person on earth. It is estimated that by 2025, the created data will increase to 463 million terabytes per day. Processing and leveraging knowledge from these sources of data requires proper infrastructure and efficient methods to analyze them in real-time. *Data stream mining* is the field of propounding such scalable and efficient methods, which can process data incrementally.

Incremental induction from a limited set of observations of an unknown distribution has been the topic of many studies for a long time. Depending on the application, the target class can be only one or many labels among which some unknown dependencies exist. Although this problem is challenging enough, in many of the stream mining applications, the statistical properties of the input and target variable(s) may change over time in unforeseen ways. This phenomena is called *concept drift*. If not considered and captured properly, the trained online models quickly become obsolete over time. However, these drifts are not well-defined and could contain any change in the statistical properties of data, adding more difficulty to the prediction problem.

In this thesis, our overall focus is to model one type of drifts which is called *recurrent concepts*. Recurrent concepts are important to be captured independently, as most of stream mining methods employ a forgetting mechanism in the learning process and forget their outdated extracted knowledge. To this end, we propose the *GraphPool* and *multi-label GraphPool* frameworks for both single-label and multi-label data streams. These frameworks keep a pool of *concepts* and their transitions in a first-order Markov chain to quickly recover from drifts in the streams with periodic behavior. In the course of designing such a framework for multi-label streams, we develop an efficient algorithm for classifying stationary multi-label streams. To show the effectiveness of our methods, we conduct an extensive set of experiments with both synthetic and real-world data.

Zusammenfassung

Heutzutage haben wir Zugang zu einer riesigen Menge an Daten in Form von Bildern, Sprachsignalen, strukturierten und unstrukturierten Texten, sowie sensorbasierten Signalen. Unser digitales Universum wächst rasant. Aus Statistiken geht hervor, dass 500 Millionen Tweets täglich hochgeladen werden. 65 Milliarden Nachrichten werden jeden Tag über Whatsapp versandt. 294 Milliarden Emails werden täglich über verschiedene Plattformen verschickt. Jedes selbstfahrende Auto erzeugt täglich 4 Terabyte an Daten. Laut einer Studie von *Digital Universe* wird die Menge an von Menschen und Maschinen produzierten Daten 44 Milliarden Terabytes bis 2020 übersteigen. Das bedeutet, zu jedem Menschen auf der Erde wird es 5,200 Gigabyte an Daten geben. Es gibt Schätzungen, dass sich die täglich generierte Datenmenge bis 2025 auf 463 Millionen Terabyte erhöht. Um diese Datenmengen zu verarbeiten und aus den Datenquellen Wissen herzuleiten, werden geeignete Infrastrukturen und effiziente Methoden benötigt, die in der Lage sind, die Daten zur Laufzeit zu analysieren. *Data-Stream-Mining* ist das Gebiet, das solche skalierbaren und effizienten Methoden bereitstellt, welche Daten inkrementell verarbeiten können.

Die inkrementelle Induktion einer beschränkten Menge an Beobachtungen einer unbekannten Verteilung ist seit langer Zeit Gegenstand vieler Studien. Abhängig von der Anwendung kann die Zielklasse nur ein oder viele Label haben, zwischen denen unbekannte Abhängigkeiten existieren können. Obwohl dieses Problem bereits eine Herausforderung darstellt, können sich zusätzlich in vielen Stream-Mining-Anwendungen die statistischen Eigenschaften des Inputs und der Zielvariable(n) unvorhergesehen über die Zeit ändern. Dieses Phänomen wird als *Concept Drift* bezeichnet. Wenn dies nicht berücksichtigt und ordentlich gehandhabt wird, werden trainierte Online-Modelle schnell obsolet. Außerdem sind diese Drifts nicht wohldefiniert und können jegliche Änderungen in den statistischen Eigenschaften der Daten enthalten, was das Prognoseproblem noch schwieriger gestaltet.

Der Hauptfokus dieser Arbeit ist die Modellierung eines Typs von Drifts, welcher als *Recurrent Concepts* bezeichnet wird. Es ist wichtig, dass Recurrent Concepts unabhängig voneinander abgefangen werden, da die meisten Stream-Mining-Methoden einen Vergessen-Mechanismus in den Lernprozess einbauen und ihr veraltetes extrahiertes Wissen vergessen. Wir schlagen die *GraphPool* und *Multi-Label GraphPool*-Umgebungen für Single-Label und Multi-Label-Datenströme vor. Diese Umgebungen behalten einen Pool von *Konzepten* und ihrer Überführung in eine Markovkette erster Ordnung, um sich schnell an periodische Drifts in den Strömen anzupassen. Im Zusammenhang mit der Konzipierung einer solchen Umgebung für Multi-Label-Ströme, entwickeln wir einen effizienten Algorithmus für die Klassifizierung von stationären Multi-Label-Strömen.

Um die Effektivität unserer Methoden zu demonstrieren, führen wir eine umfangreiche Menge von Experimenten anhand von synthetischen und realen Daten durch.

Acknowledgements

Acknowledgements have been removed from the online version.

Publications

Parts of this thesis have been published on international conferences and in journals:

1. Zahra Ahmadi and Stefan Kramer. "Modeling multi-label recurrence in data streams". *To be appeared in:* Proceedings of International Conference on Big Knowledge (2019).

2. Zahra Ahmadi and Stefan Kramer. "A label compression method for online multi-label classification". In: Pattern Recognition Letters 111 (2018), pp. 64 – 71.

3. Zahra Ahmadi and Stefan Kramer. "Modeling recurring concepts in data streams: A graph-based framework". In: Knowledge and Information Systems 55.1 (2018), pp. 15 – 44.

4. Zahra Ahmadi, Marcin Skowron, Aleksandrs Stier, Stefan Kramer. "An in-depth experimental comparison of RNTNs and CNNs for sentence modeling". In: Proceedings of International Conference on Discovery Science (2017), pp. 144 – 152.

5. Mohammad Javad Hosseini, Zahra Ahmadi, Hamid Beigy. "Using a classifier pool in accuracy based tracking of recurring concepts in data stream classification". In: Evolving Systems 4.1 (2013), pp. 43 – 60.

Other publications during my PhD, which are not included in the thesis:

1. Zahra Ahmadi, Peter Martens, Christopher Koch, Thomas Gottron, Stefan Kramer. "Towards bankruptcy prediction: deep sentiment mining to detect financial distress from business management reports". In: Proceedings of the IEEE International Conference on Data Science and Advanced Analytics (2018), pp. 293 – 302.

2. Patrick Rehn, Zahra Ahmadi, Stefan Kramer. "Forest of normalized trees: fast and accurate density estimation of streaming data". In: Proceedings of the IEEE International Conference on Data Science and Advanced Analytics (2018), pp. 199 – 208.

3. Derian Boer, Zahra Ahmadi, Stefan Kramer. "Privacy preserving client/vertical-servers classification". In: Lecture Notes in Artificial Intelligence for ECML PKDD MIDAS/PAP Workshops (2018), pp.125 – 140.

4. Rui Li, Zahra Ahmadi, Stefan Kramer. "Constrained latent dirichlet allocation for subgroup discovery with topic rules". In: Proceedings of European Conference on Artificial Intelligence (2014), pp. 519 – 524.

5. Junming Shao, Zahra Ahmadi, Stefan Kramer. "Prototype-based learning on concept-drifting data streams". In: Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2014), pp. 412 – 421.

List of Notations

Notation	Meaning
x, y, l, m, n	scalar values of a feature, class, label, number of features, and number of instances
$\boldsymbol{x}, \boldsymbol{l}$	vector of features and labels
X, Y, L	depending on the context, random variables related to the feature, class, and label or batch of instances, class values, and label sets
$\mathcal{X}, \mathcal{Y}, \mathcal{L}$	space of features, class, and labels
$\boldsymbol{\mu}, \boldsymbol{\Sigma}$	mean vector and covariance matrix
$\mathcal{C}$	a learner (mapping function)
$\mathcal{D}$	distance function
$\mathbb{R}$	set of real-valued numbers
$[\![\cdot]\!]$	an indicator function
$\vert\ .\ \vert$	size of a set
$\vert\vert.\vert\vert$	norm of a vector or matrix
Δ	symmetric distance of two sets
TP, TN, FP, FN	true positive, true negative, false positive, and false negative values
$< w_1 w_2 \dots w_n >$	a sequence of words
$\mathbf{W}, \mathbf{V}$	a matrix of weights and tensor
$.^T$	transpose of a vector or matrix

Contents

Chapter 1

Introduction

We are now in the era of accessing tremendous amounts of data. Millions of text data, including news articles, companies' reports, scientific publications, tweets, and blogs, are published every day. In the biomedical field alone, more than one million papers pour into the PubMed database each year – about two papers per minute [105]. Every second, on average, around $6,000$ tweets are tweeted on Twitter, which means more than 500 million tweets per day[1]. Over 100 million photos and videos are uploaded every day on *Instagram*[2]. Billions of different types of sensors frequently report their measurements. Processing such data and extracting knowledge from these streams require efficient methods that can act in real time. The field of *stream mining* has emerged to focus on developing such methods for different types of data.

A stream of data is a timely ordered sequence of instances that in many cases can be read only once or be available only for a limited period. Thus, the proposed knowledge extraction methods dealing with streaming data should possess some unique features. From their nature, stream mining methods should be incrementally updateable and anytime. *Anytime* refers to the fact that the learner is capable of returning imperfect results at any point in time, which allows it to remain functional even if a perfect solution could not be found within the necessary time frame. In terms of complexity, the methods should be simple and ideally sublinear in time and space with respect to the data size because of our limited computational resources. Such properties allow the learner to be employed in real-time decisions. In this thesis, we focus on the *supervised* approaches, where an expert provides the correct labels of each instance and the goal of the learner is to correctly predict the class of new instances based on the extracted knowledge from previous data. An instance can belong to only one true class or many of them. For example, in one application, one may be interested in classifying

[1]`https://www.internetlivestats.com/twitter-statistics/` (access date: 1.3.2019)
[2]`https://www.omnicoreagency.com/instagram-statistics/` (access date: 1.3.2019)

news texts into either fake or real news. This is a *single-label* problem with two classes. However, the goal of another application could be to find different possible categories that the news texts belong to, e.g., international, social, politics, election, sports, to name a few. In this case, one instance can belong to more than one possible class, and there may be some unknown correlations among different classes. This problem is called *multi-label learning*.

It has been now over two decades that researchers have developed various supervised methods based on possibly different assumptions for a variety of real-world streaming problems. Although there have been extensive studies on data with a single target class, not many studies focused on multi-label stream mining, perhaps due to the only recent emergence.

Another critical challenge of processing streaming data is the possibility of change in the underlying distribution of data. In a classic learning problem, our goal is to find the best approximation for the function that separates classes well, and for that, we assume that the training instances are consistent with that target, or in other words, training data is stationary. However, when the data is received incrementally, this assumption is most likely not valid. Remember our news categorization problem: Depending on the period of time, news in each category may vary. For example, in politics, the texts may vary from the Iran nuclear deal to tapping German Chancellery to parliament election. Such *drift* in the learning function poses more challenge to the problem of stream mining, especially for the multi-label setting. We can imagine that many of this news will show up again some possibly long time later, e.g., the elections will frequently repeat every few years and each party will represent similar or almost the same discussions and plans. While considering to handle different types of drifts, our main focus in this thesis is to learn these *recurring concepts*.

1.1 Thesis contributions and structure

The main contribution of this thesis lies in modeling recurrent concepts for streaming data, no matter whether the data has a single target class or multiple labels. We achieve our goal by developing a pool based method that is actively updated and manages its size by frequently checking the similarity of existing concepts and merging them when necessary. Concepts are identified as similar by applying a similarity measure, as in *PASC*, or more generally by applying a statistical likelihood test, as in *GraphPool*. To extend the recurring framework to the multi-label setting, we also developed a novel efficient scalable multi-label streaming method based on a random projection of the label space and later integrated that model into the pool-based framework. Table 1.1 presents an overview of the methods proposed in each chapter of this thesis. The individual contributions are given at the

Table 1.1: Overview of methods introduced in this thesis. Chapter 3 studies deep neural networks as an offline learner for single- and multi-label text data. Chapter 4 proposes a new multi-label stream mining method. Chapters 5 and 6 propose two novel recurrent concept methods for single-label streams. Chapter 7 combines the proposed methods from Chapter 4 and 6 to achieve the first multi-label recurrent model.

		Learner		
		Offline	Online	
			Non-recurrent	Recurrent
Data	Single-label	Chapter 3	–	*PASC* (Chapter 5) *GraphPool* (Chapter 6)
	Multi-label		*RACE* (Chapter 4)	*multi-label GraphPool* (Chapter 7)

beginning of each chapter. Here, we review them as they appear in the thesis:

- A great portion of our massive sources of data is text data. Hence, understanding complex documents is an essential task. Recently, deep neural networks have shown promising results in image and speech processing. Their application to many different natural language processing tasks, such as sentiment analysis, has shown a considerable performance improvement. *Recursive Neural Tensor Networks* (*RNTNs*) are a well-known successful approach that integrates semantic content of a sentence (i.e., its parse tree information) with a recursive neural network architecture. Another successful deep network architecture, especially on image data, is *Convolutional Neural Networks* (*CNNs*). Chapter 3 investigates an in-depth study of these two popular deep neural network architectures for several text categorization data. It also proposes two different ways of automatic labeling to reduce the need for intensive manual labeling in RNTNs.

  Published as: Zahra Ahmadi, Marcin Skowron, Aleksandrs Stier, Stefan Kramer. "An In-Depth Experimental Comparison of RNTNs and CNNs for Sentence Modeling". In: Proceedings of International Conference on Discovery Science (2017), pp. 144 – 152.

- Chapter 4 focuses on a stationary stream of multi-label data. We consider the sparsity of label sets in most of the multi-label datasets beside their unknown dependencies among labels. As a result, we propose an efficient multi-label learning method that reduces the space of labels by applying random projections, *Random Compression* (*RACE*). Different from deep neural network architectures, it finds the mapping function analytically based on the least squares solution.

  Published as: Zahra Ahmadi and Stefan Kramer. "A label compression method for online multi-label classification". In: Pattern Recognition Letters 111 (2018), pp. 64 – 71.

- Chapter 5 focuses on the problem of recurring concepts for single-label streams. It builds a new framework, *Pool and Accuracy based Stream Classification* (*PASC*), by extending one of the successful existing frameworks for handling recurrent concepts, *Conceptual Clustering and Prediction* (*CCP*). In this chapter, we try to address the limits of the CCP framework and improve the similarity measure in such a way that it is less domain-specific. We also study the effect of different prediction strategies.

  Published as: Mohammad Javad Hosseini, Zahra Ahmadi, Hamid Beigy. "Using a classifier pool in accuracy based tracking of recurring concepts in data stream classification". In: Evolving Systems 4.1 (2013), pp. 43 – 60.

- Chapter 6 continues the problem of recurrent concepts for single-label streams and proposes a new first-order Markov chain framework for connecting concepts in the pool. Using this directed graph-based modeling of the concepts, we are able to capture the periodic behavior of the data. To remove the domain dependency of the similarity measure, we apply a likelihood statistical test. This approach is called *GraphPool*, and it also benefits from an effective merging strategy to manage the pool size.

  Published as: Zahra Ahmadi and Stefan Kramer. "Modeling recurring concepts in data streams: A graph-based framework". In: Knowledge and Information Systems 55.1 (2018), pp. 15 – 44.

- Finally, chapter 7 addresses the problem of recurrent concepts in multi-label streams. For that, we benefit from our efficient multi-label stream classifier (*RACE*) and integrate it into the successful *GraphPool* framework. However, this combination is not trivial. We needed to come up with new ways of concept representations and concept comparisons for multi-label data, which all are discussed in this chapter.

  To be appeared in: Zahra Ahmadi and Stefan Kramer. "Modeling Multi-Label Recurrence in Data Streams". International Conference on Big Knowledge (2019).

Before investigating on our contributions, we review the literature and problem definitions in Chapter 2.

Chapter 2

Background

The background chapter will introduce concepts and algorithms from machine learning and data mining that are related to the proposed methods in this thesis. We focus on the classification problem, where the learner is provided a set of training instances and supposed to predict unseen test instances. Section 2.1 explains the problem of multi-label classification, its challenges, its most common evaluation measures, and a review of literature on the topic. In Section 2.2, we introduce the problem of stream classification and concept drifts, and review the literature. Finally, in Section 2.3, we provide an overview of deep learning methods and their application to multi-label problems.

2.1 Multi-label classification

Standard classification is the task of assigning the correct class to previously unknown test instances based on training instances. Training data are most commonly described by features and an associated target class or class label. Many modern data mining applications, however, need to deal with more than one label per instance. This problem is called *multi-label learning*. As a formal definition, we define *multi-label data* and the concept of a *multi-label classifier* as follows:

Definition 1. Let $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^m$ and $\boldsymbol{l} = (l_1, l_2, \ldots, l_l) \in \{0,1\}^l = \mathcal{L}$ denote an instance of an m-dimensional feature space and the relevant binary label set of size l, respectively. We assume that instances are generated independently from a joint distribution $P(X, L)$ over $\mathcal{X} \times \mathcal{L}$.

The goal of a multi-label classifier $\mathcal{C} : \mathcal{X} \to \mathcal{L}$ is to learn a prediction function $\mathcal{C}$ from a given set of training instances that minimizes the risk, ξ, with respect to a label-wise decomposable loss function $(f(.))$. This risk function is defined as the expected loss over the joint distribution $P(X, L)$:

$$\xi = \mathbb{E}_{X,L} f(L, \mathcal{C}(X)). \tag{2.1}$$

Multi-label classification can be viewed as a generalization of multi-class classification where labels do not exclude each other and may have unknown dependencies among each other as well as with the features.

2.1.1 Properties of multi-label datasets

Not all multi-label datasets are equal even with the same number of labels and instances. The number of labels for each instance can vary among different datasets, and this will influence the performance of a multi-label classifier. Hence, it is important to notice the properties of a multi-label dataset while comparing different multi-label learning algorithms.

Label cardinality (LC) indicates the average number of labels per instance and, if normalized by the number of labels, it is called *label density* (LD):

$$LC = \frac{1}{n}\sum_{i=1}^{n} \mid \boldsymbol{l}_i \mid, \quad LD = \frac{LC}{l}. \tag{2.2}$$

Unique label set (UL) indicates the total number of distinct label combinations observed in the dataset and, if normalized by the number of instances, it is called *proportion of unique label set* (PUL):

$$UL = \mid \{\boldsymbol{l} \mid \exists \boldsymbol{x} : (\boldsymbol{x}, \boldsymbol{l}) \in S\} \mid, \quad PUL = \frac{UL}{n}. \tag{2.3}$$

2.1.2 Evaluation measures

Let $T = (\boldsymbol{x}_i, \boldsymbol{l}_i) \mid_{i=1}^{n}$ be a multi-label test set with n instances and l labels, and let $\boldsymbol{l}_i$ and $\mathcal{C}(\boldsymbol{x}_i)$ be the sets of true and predicted labels for instance i. To assess the performance of multi-label models, we should consider all labels and the output of a model for each label. Therefore, we need specific measures to evaluate multi-label models. Gibaja and Ventura [61] summarize well-known performance measures into two categories: *measures to evaluate bipartitions*, and *measures to evaluate rankings*. In the following, we will explain both categories briefly. TP, FP, TN, and FN are the number of true positives, false positives, true negatives, and false negatives, respectively.

Bipartition-based measures: These measures can be either measured *label-based* or *example-based*. Label-based measures are calculated for each label and then averaged across all labels, while example-based measures evaluate the performance of the multi-label model on each test instance separately and return the mean value across the test set. Here, we introduce the most common example-based and label-based measures:

- *0/1 subset accuracy* or *exact match ratio* computes the percentage of test instances whose predicted label set is exactly the same as their corresponding ground-truth:

$$0/1\ subset\ accuracy = \frac{1}{n}\sum_{i=1}^{n} [\![\boldsymbol{l}_i = \mathcal{C}(\boldsymbol{x}_i)]\!]. \tag{2.4}$$

  Subset accuracy can intuitively be regarded as a multi-label counterpart of accuracy in single-label problems. However, it is rather a very strict measure, especially when the label space is large and even more challenging in the data stream setting.

- *Hamming loss* evaluates the fraction of misclassified instance-label pairs. That means it takes into account misclassification of both relevant labels and irrelevant predictions:

$$Hamming\ Loss = \frac{1}{nl}\sum_{i=1}^{n} \mid \boldsymbol{l}_i \,\Delta\, \mathcal{C}(\boldsymbol{x}_i) \mid = \frac{1}{nl}\sum_{i=1}^{n}(FP_i + FN_i), \tag{2.5}$$

  where Δ stands for the symmetric distance between two sets. Again, Hamming loss can be interpreted as a generalization of the misclassification rate in single-label problems. Studies show that classifiers aiming at optimizing subset accuracy will perform rather poorly when evaluated in terms of Hamming loss, and vice versa [42].

- Information retrieval based measures: We can use a multi-label version of *precision*, *recall*, *F-measure*, and *accuracy* measures to evaluate a multi-label model. *Example-based accuracy* is the proportion of correctly classified labels out of the total number of labels, whereas *example-based precision* and *example-based recall* are the fraction of correctly classified labels out of the predicted positive labels and the actual labels, respectively. *F1-measure* is the harmonic mean of precision and recall. The following equations indicate their definitions:

$$\begin{aligned}
Example-based\ precision &= \frac{1}{n}\sum_{i=1}^{n}\frac{\mid \boldsymbol{l}_i \cap \mathcal{C}(\boldsymbol{x}_i)\mid}{\mid \mathcal{C}(\boldsymbol{x}_i)\mid} = \frac{1}{n}\sum_{i=1}^{n}\frac{TP_i}{TP_i+FP_i}, \qquad (2.6)\\
Example-based\ recall &= \frac{1}{n}\sum_{i=1}^{n}\frac{\mid \boldsymbol{l}_i \cap \mathcal{C}(\boldsymbol{x}_i)\mid}{\mid \boldsymbol{l}_i\mid} = \frac{1}{n}\sum_{i=1}^{n}\frac{TP_i}{TP_i+FN_i},\\
Example-based\ F1 &= \frac{1}{n}\sum_{i=1}^{n}\frac{2\mid \boldsymbol{l}_i \cap \mathcal{C}(\boldsymbol{x}_i)\mid}{\mid \boldsymbol{l}_i\mid + \mid \mathcal{C}(\boldsymbol{x}_i)\mid} = \frac{1}{n}\sum_{i=1}^{n}\frac{2TP_i}{2TP_i+FP_i+FN_i},\\
Example-based\ accuracy &= \frac{1}{n}\sum_{i=1}^{n}\frac{\mid \boldsymbol{l}_i \cap \mathcal{C}(\boldsymbol{x}_i)\mid}{\mid \boldsymbol{l}_i \cup \mathcal{C}(\boldsymbol{x}_i)\mid} = \frac{1}{n}\sum_{i=1}^{n}\frac{TP_i}{TP_i+FP_i+FN_i}.
\end{aligned}$$

Label-based measures can be any binary evaluation measure (e.g., accuracy, precision, recall, and F-measure) applied to the prediction results of each label and obtaining an average value across all labels. Averaging can be done in two ways: *macro* and *micro*. In micro averaging, we first aggregate the values of all the contingency tables for all instances together and then calculate the measure across all labels:

$$M_{micro} = M\big(\sum_{i=1}^{l} TP_i, \sum_{i=1}^{l} FP_i, \sum_{i=1}^{l} TN_i, \sum_{i=1}^{l} FN_i\big), \tag{2.7}$$

where M is a binary evaluation measure. By contrast, in macro averaging, we compute one measure for each label and then average over all the labels:

$$M_{macro} = \frac{1}{l} \sum_{i=1}^{l} M(TP_i, FP_i, TN_i, FN_i). \tag{2.8}$$

For example, *micro-averaged precision* and *macro-averaged precision* are calculated as follows:

$$\begin{aligned} Micro-averaged\,precision &= \frac{\sum_{i=1}^{l} TP_i}{\sum_{i=1}^{l} TP_i + \sum_{i=1}^{l} FP_i}, \\ Macro-averaged\,precision &= \frac{1}{l} \sum_{i=1}^{l} \frac{TP_i}{TP_i + FP_i}, \end{aligned} \tag{2.9}$$

Conceptually speaking, macro-averaged measures give an equal weight to each label, regardless of its frequency, while micro-averaged measures give an equal weight to each instance and tend to be dominated by the performance in the most common labels. Therefore, macro-averaged measures are better to be used in the problems with skewed training data across different categories, whereas the micro-averaged measures are better for the applications where the density of the class is important.

Ranking-based measures: If a multi-label learner is able to provide a ranking of the predicted labels, the following measures can also evaluate the performance of the learner:

- *One-error* measures the probability of not getting even one of true labels by counting how many times the top ranked predicted label is not in the set of true labels:

  $$One-error = \frac{1}{n} \sum_{i=1}^{n} [\![\arg\min_{l \in \mathcal{L}} \boldsymbol{\tau}(\boldsymbol{x}_i, l) \notin \boldsymbol{l}_i]\!], \tag{2.10}$$

  where $\boldsymbol{\tau}$ is a rank function. Obviously, this measure is not a good measure in multi-label problems, as it only considers the top-ranked label.

- *Coverage* measures the average needed depth in the ranking to cover all the labels associated with an instance:

$$Coverage = \frac{1}{n}\sum_{i=1}^{n} \max_{l \in \mathcal{L}} \boldsymbol{\tau}(\boldsymbol{x}_i, l) - 1. \tag{2.11}$$

  This measure is especially used in applications where predicting all true labels are important even with a few extra false positives. It is possible to have a good coverage while having high one-error.

- *Macro-averaged AUC* and *Micro-averaged AUC* indicate the averaged rankings of all instances per label over labels and the averaged rankings of correct and incorrect labels over all instances and labels, respectively:

$$Macro\ AUC = \frac{1}{l}\sum_{i=1}^{l} \frac{\mid \{(\boldsymbol{x'}, \boldsymbol{x''}) \in \mathcal{X}_j \times \bar{\mathcal{X}}_j | \boldsymbol{\tau}(\boldsymbol{x'}, l_i) \geq \boldsymbol{\tau}(\boldsymbol{x''}, l_i)\} \mid}{\mid \mathcal{X}_j \mid\mid \bar{\mathcal{X}}_j \mid}, \tag{2.12}$$

$$Micro\ AUC = \frac{\mid \{(\boldsymbol{x'}, \boldsymbol{x''}, l', l'') | \boldsymbol{\tau}(\boldsymbol{x'}, l') \geq \boldsymbol{\tau}(\boldsymbol{x''}, l''), (\boldsymbol{x'}, l') \in \mathcal{S}^+, (\boldsymbol{x''}, l'') \in \mathcal{S}^-\} \mid}{\mid \mathcal{S}^+ \mid\mid \mathcal{S}^- \mid},$$

  where $\mathcal{X}_j = \{\boldsymbol{x}_i | l_j \in \boldsymbol{l}_i\}$ and $\bar{\mathcal{X}}_j = \{\boldsymbol{x}_i | l_j \notin \boldsymbol{l}_i\}$ correspond to the set of test instances with and without label l_j, and $\mathcal{S}^+ = \{(\boldsymbol{x}_i, l) | l \in \boldsymbol{l}_i\}$ and $\mathcal{S}^- = \{(\boldsymbol{x}_i, l) | l \notin \boldsymbol{l}_i\}$ represent the set or relevant and irrelevant instance-label pairs.

2.1.3 Multi-label learning methods

Very early multi-label studies focused on multi-label text categorization [120], and gradually the topic attracted attention from different communities, such as bioinformatics [36] and image labeling [24]. Along these years, several single-label classification algorithms have been adapted to the multi-label scenario (e.g., ML-kNN [206]). In contrast, some other methods are algorithm-independent and convert the original multi-label problem into one or several single-label problems, and apply one of the existing single-label learning methods to the transformed data. Thus, Tsoumakas *et al.* [178] categorize the multi-label classification approaches into two general groups: *algorithm adaptation* and *problem transformation* methods. In the following, we briefly review the literature of both approaches:

Agorithm adaptation methods

Many of the classical single-label classification methods have been adapted to the multi-label schema. ML-C4.5 [36] is a multi-label adaptation of the popular C4.5 decision tree learner, where leaves can contain multiple labels and the entropy definition is adapted to consider how much information is needed to describe to what labels a certain pattern belongs:

$$entropy(S) = \sum_{i=1}^{l} \Big(P(l_i) \log P(l_i) + (1 - P(l_i)) \log(1 - P(l_i)) \Big). \tag{2.13}$$

A multi-label adaptation of the famous SVM was proposed in *Rank-SVM* [46]. The approach aims to optimize l linear classifier to minimize the empirical ranking loss with quadratic programming in its dual form and the kernel trick to manage nonlinearity. The multi-label margin is defined as:

$$\min_{(\boldsymbol{x}_i, \boldsymbol{l}_i) \in S} \min_{(l_j, l_k) \in \boldsymbol{l}_i \times \bar{\boldsymbol{l}}_i} \frac{< w_j - w_k, \boldsymbol{x}_i > + b_j - b_k}{||w_j - w_k||}, \tag{2.14}$$

where S is the training set, and w and b are weight vector and bias of linear classifiers. The boundary for each pair of relevant-irrelevant labels corresponds to the hyperplane $< w_j - w_k, \boldsymbol{x}_i > + b_j - b_k$.

Multi-Label k-Nearest Neighbor (ML-kNN) [206] was the first multi-label version from lazy learning family of algorithms. ML-kNN finds the k nearest instances to the test instance, and stores for each label the number of instances, which belong to that label $(c_i, 1 \leq i \leq l)$. Then it uses the *Maximum A Posteriori* (MAP) principle to identify the labels:

$$l_i = \begin{cases} 1 & \text{if } P(c_i|l_i = 1)P(l_i = 1) \geq P(c_i|l_i = 0)P(l_i = 0) \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

Later, Cheng and Hüllermeier combined instance-based learning and logistic regression to capture interdependencies between labels [34]. The key idea is to consider labels of neighboring instances as features of unseen examples.

AdaBoost.MH is a multi-label adaptation of AdaBoost proposed for text categorization [156]. It minimizes Hamming loss and maintains a set of weights not only over the training set but also over labels. *AdaBoost.MR* [156] is another multi-label variation of the AdaBoost algorithm, which aims to minimize the ranking loss. For that, it has to take into account all labels misorderings. Therefore, it maintains the set of weights for each instance and a pair of labels.

There are many other single-label approaches, from associative classification [143], generative models [184], neural networks (e.g., Backpropagation for Multi-Label Learning (BP-MLL) [207] and Multi-Label Radial Basis Function (ML-RBF) [205]), to evolutionary algorithms (e.g., Multi-Label Ant-Miner (MuLAM) [30]) that have been adapted to the multi-label scenario.

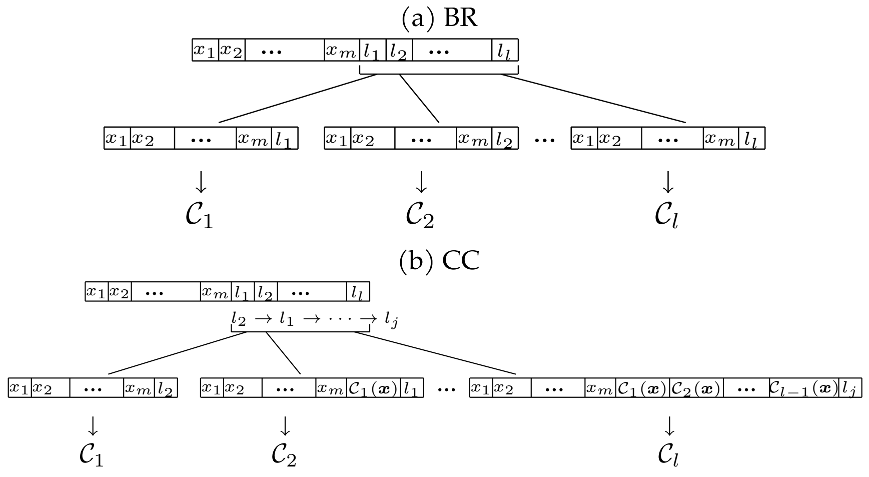

(a) BR
x_1 x_2 ··· x_m l_1 l_2 ··· l_l
x_1 x_2 ··· x_m l_1
x_1 x_2 ··· x_m l_2
x_1 x_2 ··· x_m l_l
$\mathcal{C}_1$
$\mathcal{C}_2$
$\mathcal{C}_l$
(b) CC
x_1 x_2 ··· x_m l_1 l_2 ··· l_l
$l_2 \to l_1 \to \cdots \to l_j$
x_1 x_2 ··· x_m l_2
x_1 x_2 ··· x_m $\mathcal{C}_1(\boldsymbol{x})$ l_1
x_1 x_2 ··· x_m $\mathcal{C}_1(\boldsymbol{x})$ $\mathcal{C}_2(\boldsymbol{x})$ ··· $\mathcal{C}_{l-1}(\boldsymbol{x})$ l_j
$\mathcal{C}_1$
$\mathcal{C}_2$
$\mathcal{C}_l$

Figure 2.1: Comparison of Binary Relevance (BR) and Classifier Chains (CC) classifier training.

Problem transformation methods

The most popular method from this category is called *Binary Relevance* (BR), which is similar to the one-versus-all approach for multi-class classification. BR assumes the labels are independent and trains a separate model for each label. BR is a simple method that has low computational complexity compared to many other multi-label methods. Its label independence assumption also makes it applicable for adding/removing labels without affecting the rest of the model, suitable for evolving multi-label data, and possible for parallel implementation. On the other hand, BR suffers from ignoring label correlations, sample imbalance, and an increasing number of classifiers for high dimensional labels. Later, to alleviate the label independence problem, *Classifier Chains* (CC) [149] were proposed. CC adds links to the classifiers in BR in such a way that the feature space of each link in the chain is extended with the label associations of all previous links. Figure 2.1 illustrates the difference in training data for BR and CC.

As the order of the chain can influence the performance, an *Ensemble of Classifier Chains* (ECC) was proposed. ECC trains a set of CC classifiers with a random chain order and a random subset of training data. A threshold is applied to the normalized sum of votes for each label to produce the output. Instead of random order, a Bayes optimal way of forming chains based on probabilistic integration of links was proposed in *Probabilistic Classifier Chains* (PCC) [41]. It tests all possible combination of chain orders and selects the best one based on the likelihood of correct prediction for instances. Although the model achieves better accuracy than CC, its time complexity is exponential in terms of label set size and only recommended for small label sets.

Another way to address the label independence assumption of BR is to use two layers of BR as in *Meta-BR* (MBR) [149]. MBR learns a binary meta-learner for each label, which is fed by the output of BR classifiers in the first layer. A pruning strategy is applied to the output of the first layer and only the predictions on those labels whose absolute value of the ϕ coefficient is greater than a pre-defined threshold is considered in the meta-learning:

$$\phi = \frac{ad - bc}{\sqrt{(a+b)(c+d)(a+c)(b+d)}}, \tag{2.16}$$

where a is the frequency of co-occurrence of label i and j, b is the frequency of occurrence of i but not j, c is the frequency of occurrence of j but not i, and d is the frequency of instances where i and j do not occur. Experimental results showed that pruning improves the computational cost substantially while maintaining or improving predictive performance [176].

Ranking and Threshold [177] uses, for each label, a single-label classifier that can produce a score. All labels associated with a confidence greater than a threshold form the set of prediction. To build the training data, a straightforward approach copies one multi-label instance for all its label set (and possibly weigh them). However, this approach increases the number of patterns and makes modeling more complicated. To simplify the procedure, one may select the multi-label instance for a subset of labels. Depending on the selection method, the most frequent, the less common or even random subset could be chosen.

The *Label Powerset* (LP) method generates a new class for each possible combination of labels and solves the resulting multi-class problem. This approach is simple, and it considers label correlations. However, the training data could be limited due to a large number of possible classes, and the classifier cannot predict unseen label sets. The complexity of LP is exponential with the number of labels. *Random k-labelsets* (RAkEL) constructs an ensemble of LP classifiers, each with a random subset of k labels. It aims to reduce the computational complexity and class imbalance of LP when the number of labels is large. Moreover, the method averages over the prediction of the ensemble per label and applies a threshold to assign the label set. In this way, RAkEL overcomes the limitation of LP in dealing with unseen label sets. Experimental results show that employing LP instead of BR or ML-kNN in RAkEL improves the results, and as a single-label classifier, using C4.5 and Support Vector Machines (SVMs) leads to better performance than Naïve Bayes [180].

An approach considering the trade-off approach between the simplicity of BR and the complexity of LP is called LPBR [174]. It is an iterative approach that, in each round, trains an LP on already grouped dependent labels and a BR on independent labels. First, all labels are sorted using the χ^2 score:

$$\chi^2 = \frac{(ad - bc)^2(a+b+c+d)}{(a+b)(c+d)(b+d)(a+c)}. \tag{2.17}$$

Moreover, the most dependent label pair is chosen. Depending on whether the labels of the pair belong to other previous groups of labels, either a new group with the new pair is formed, or they can be joined to a previous group, or two previous groups can be joined together to include this pair. Predictive performance of LPBR is higher than LP and BR; however, training time is relatively long and comparable to RAkEL and MBR.

Pruned Sets (PS) [150] also tries to address the limits of LP by pruning instances with less frequent label sets. To compensate for the information loss, PS reintroduces pruned instances with their disjoint subsets of the pruned label sets that occur more frequently than a threshold in the training set. *Ensemble of Pruned Sets* (EPS) follows the same strategy as RAkEL in classifying instances with new label sets. Experimental results showed that EPS outperformed LP and RAkEL [150].

Another method, *Ranking by Pairwise Comparison* (RPC) [82], follows a similar approach to *pairwise classification* in multi-class learning by transforming an l label problem into $l(l-1)/2$ binary datasets, each for one pair of labels, and builds a binary classifier on each dataset. Each dataset contains instances which belong only to one of the labels, and it considers one of them as class 0 and the other as class 1. This approach has quadratic complexity in terms of space and time, and thus is prohibitive especially for problems with a large number of labels. Fürnkranz *et al.* extend the RPC approach by a calibration label that can be interpreted as a split for relevant and nonrelevant labels [51]. This way, the complexity of queries is even more $(l^2 + l)$.

Label space reduction methods

With an increase in the number of labels, many of the standard multi-label classification methods that work in the original label space (e.g., BR and ECC) become computationally infeasible. Hence, new strategies for reducing the label space have been presented, which are called *Label Space Dimension Reduction* (LSDR) methods [80, 172, 193, 212]. These methods accelerate the learning process by training fewer binary classifiers on compressed label sets. The reduction on the label space and the number of classifiers to be learned makes such methods suitable for multi-label stream classification problems where the time complexity of methods matters.

Random Projection (RP) is a simple and computationally efficient linear dimensionality reduction technique that is data oblivious. The key idea behind that stems from the *Johnson-Lindenstrauss lemma* [89], which states that a set of points in a high-dimensional space can be embedded into a lower-dimensional space, with distances between these points preserved up to a certain multiplicative factor:

Definition 2 (JL-lemma)**.** Let $\epsilon \in (0,1)$ and $\mathbb{A}$ be a set of n points in $\mathbb{R}^d$, and $k = \mathcal{O}(\epsilon^{-2} \log n)$ be an integer. There exists a mapping $f : \mathbb{R}^d \to \mathbb{R}^k$ such that for any $\mathbf{a}, \mathbf{b} \in \mathbb{A}$:

$$(1-\epsilon)||\mathbf{a}-\mathbf{b}||_2 \leq ||f(\mathbf{a})-f(\mathbf{b})||_2 \leq (1+\epsilon)||\mathbf{a}-\mathbf{b}||_2. \tag{2.18}$$

Johnson-Lindenstrauss lemma indicates that any dataset with n instances, regardless of its dimensionality, can be represented in $k = \mathcal{O}(\epsilon^{-2}\log n)$ dimensions such that pairwise distances between any two instances is preserved up to a multiplicative factor of $1 \pm \epsilon$, where ϵ is the distortion.

JL-lemma is an optimal estimation in terms of n and ϵ, which means that without prior knowledge of the dataset, no linear dimensionality reduction technique can improve the guarantee on k. However, the lemma does not specify how to create the mapping f. In their proof, Johnson and Lindenstrauss chose f as a properly scaled dense orthogonal transformation projection matrix, which is neither easy nor efficient for practical applications. Later studies showed the unnecessity of the orthogonality constraint [84]. This observation led to a simple random projection matrix construction that initializes the projection matrix with random numbers drawn from a normal distribution. An orthogonalization algorithm can be applied to the matrix later. One example of such an orthogonalization method is the *Gram-Schmidt* method, which runs in $\mathcal{O}(dk^2)$. We later use this approach in Chapter 4 as an initialization of the proposed RACE method. However, one may leave the random matrix without any orthogonalization step, as studies showed the probability of random vectors being orthogonal or almost orthogonal increases with the vector dimensionality [74].

Gaussian random projection is an easy-to-implement algorithm that produces a high-quality sketch of the original data matrix; however, compared to more recent random projection methods, it needs more computations. Several different projection methods with good embedding quality have been proposed in the literature: Some focus on the fast embedding of potentially dense data [6], while some others concentrate on embedding highly sparse data [39, 129].

Applying random projection methods to multi-label data, an early work in label space reduction family of algorithms used *Compressed Sensing* (CS) approach for multi-label prediction [80]. Compressed sensing is a random projection technique based on the assumption of the sparsity in the signal and the incoherence, which is applied through the isometric property. Hsu *et al.* assumed sparsity in the label set and encoded labels using a small number of linear random projectors. Although the encoding function is linear, the decoder is based on the *Orthogonal Matching Pursuit* (OMP) [117], which is a nonlinear iterative and greedy reconstruction algorithm. For each test instance, CS needs to solve an optimization problem related to its sparsity assumption. Hence, CS can be time-consuming during prediction.

Unlike CS, the projection matrix in *Principle Label Space Transformation* (PLST) [172] captures the correlations between labels using *Singular Value*

Decomposition (SVD) of the label matrix:

$$\min_{\mathbf{o},\mathbf{P}} \frac{1}{n} \sum_{i=1}^{n} ||\boldsymbol{l}_i - \mathbf{o} - \mathbf{P}^T\mathbf{P}(\boldsymbol{l}_i - \mathbf{o})||^2 \text{ such that } \mathbf{P}\mathbf{P}^T = \mathbf{I}, \tag{2.19}$$

where $\mathbf{o} \in \mathbb{R}^l$ is a reference point and $\mathbf{P}$ is an $k \times l$ matrix where $k << l$. This approach guarantees the minimum encoding error on the training set. Both encoding and decoding functions are computed from the SVD, and thus, both are linear. PLST outperforms CS in terms of prediction accuracy and time complexity.

Zhang and Schneider employed *Canonical Correlation Analysis* (CCA) in the multi-label setting [211]. CCA is a classical tool for modeling linear associations between two sets of variables. In the multi-label setting, they consider the feature set as the first variable and the label set as the second. Canonical correlation analysis finds a pair of projection directions $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^l$ such that the correlation between the pair of projected variables $\mathbf{u}^T\boldsymbol{x}$ and $\mathbf{v}^T\boldsymbol{l}$ is maximized:

$$\arg\max_{\mathbf{u},\mathbf{v}} \frac{\mathbf{u}^T\mathbf{X}^T\mathbf{L}\mathbf{v}}{\sqrt{(\mathbf{u}^T\mathbf{X}^T\mathbf{X}\mathbf{u})(\mathbf{v}^T\mathbf{L}^T\mathbf{L}\mathbf{v})}}, \tag{2.20}$$

where $(\mathbf{X}, \mathbf{L})$ indicates the matrix of training instances. Label dependency is characterized as the most predictable directions in the label space, extracted as canonical output variates and encoded into the codeword. Predictions for the codeword define a graphical model of labels with both Bernoulli potentials (from classifiers on the labels) and Gaussian potentials (from a regression on the canonical variates). For a test instance, the exact solution of the decoding function has exponential complexity. Hence, a mean-field approximation is performed on the joint probability to reduce the complexity, which offers a tractable predictive distribution on labels [211]. They further proposed a maximum margin formulation to learn the output coding that is both predictive and discriminative [210]. However, its optimization relies on the cutting plane algorithm, which may not be efficient when there are many labels.

Linear Gaussian random projection is another form of transforming labels, which was used in *Compressed Labeling* (CL) [212]. CL compresses the original label set to improve balance and independence by preserving the signs of its Gaussian random projections. In the decoding phase, the method uses a series of Kullback-Leibler divergence based hypothesis tests on the *distilled label sets* (DLs). DLs are the frequent label subsets extracted from the original labels by performing a recursive clustering algorithm and subtraction on the label vectors. Although the decoding phase is composed of a series of hypothesis tests and is linear in time, extracting distilled label sets through a recursive clustering method is empirically expensive.

MLC-BMaD [193] is another label compression method that uses Boolean Matrix Decomposition (BMaD) to factorize the label matrix into the product of a Boolean code matrix and a Boolean decoding matrix. The technique needs all the training data at once to extract the compressed label set. Another study employs Bloom filters, a space-efficient randomized data structure initially designed for approximate membership testing, for multi-label classification [35].

Instead of transforming label sets to a much smaller space, which may make the problem more difficult to learn, one approach selects a small subset of labels that can approximately span the original space [11, 14]. These methods are based on the assumption that all the output labels can be covered by a small subset ($\mathbf{L} \simeq \mathbf{LW}$):

$$\min_{\mathbf{W}} ||\mathbf{L} - \mathbf{LW}||_F^2 + \lambda_1 ||\mathbf{W}||_{1,2} + \lambda_2 ||\mathbf{W}||_1, \tag{2.21}$$

where $\mathbf{W} \in \mathbb{R}^{l \times l}$ is the coefficient matrix with only a few nonzero rows, λ_1, λ_2 are regularization parameters, $||\mathbf{W}||_{1,2} = \sum_{i=1}^{l} \sqrt{\sum_{j=1}^{l} W_{ij}^2}$ is the $l_{1,2}$ group-sparsity regularizer that encourages the row sparsity of $\mathbf{W}$, and $||\mathbf{W}||_1 = \sum_{i=1}^{l} |W_{ij}|$ is the traditional l_1 regularizer that encourages sparsity over the whole $\mathbf{W}$. While *multiple output landmark selection* [11] requires an expensive optimization problem to select the best labels (columns), a more efficient randomized sampling method based on the *Column Subset Selection Problem* (CSSP) was proposed by Bi and Kwok [14]. Instead of using a predetermined number of sampling trials, the number of trials is adaptive and based on performing partial SVD.

All the previous methods compress the label set regardless of the corresponding feature set. This approach can be considered as an unsupervised manner by not considering the input feature information. Recently, some feature-aware methods have been proposed that find the optimized compression function considering both feature and label sets. *Conditional Principal Label Space Transformation* (CPLST) is a feature-aware variation of PLST that minimizes both the encoding error and the training error in the reduced space [33]. Another feature-aware method (FaIE) [113] learns both the encoding and the linear decoding matrices by jointly maximizing the recoverability of the original space from the latent space and the predictability of the latent space from the feature space. FaIE can be considered as a generalization of PLST and CPLST. Cao and Xu use the *Hilbert-Schmidt independence criterion* to maximize the dependency between features and labels via solving an eigenvalue problem [26]. Their method uses the same decoding matrix as in PLST and CPLST, but the eigenvalues decrease more quickly than in PLST and CPLST, and therefore, their method results in shorter codewords. A more recent study extends CCA to the *Kernel Canonical Correlation Analysis* (KCCA) to reduce labels in a nonlinear feature-aware scheme [111]. Their goal is to capture different types of label correlation patterns in the original label space and increase the label information preserved in the reduced space. As a decoding phase, a sparsity regularized least squares loss minimization problem is solved, and a projected gradient descent algorithm is developed to solve the minimization problem.

2.2 Stream mining

Due to the ever-increasing volumes of data generated today, processing data and extracting meaningful knowledge has become a challenge in many problem domains. In many cases, the complete dataset cannot be saved and processed offline anymore in a way classical data mining methods assume. Consequently, there is a great need for algorithms that can process a continuous and unlimited stream of data. *Data stream classification* has been considered as a challenging problem in many real-world applications such as intrusion detection, user interest, and climate forecast, among others. A stream of data can be viewed as a stochastic process of continuous and independent data/events where data stream algorithms are required to possess some properties [171, 52]:

- Not all the training data is available at once, and thus the algorithms need to operate incrementally.
- The algorithms need to process an unlimited stream of data, which cannot be stored completely. Therefore, they need a forgetting mechanism to keep the most relevant subset of data.
- Because of the scale of data, the algorithms should process instances online and in one pass. Hence, the processing algorithm needs to be simple.
- The most important feature of a data stream classification algorithm is the ability to cope with drifts in the underlying distribution of data. We define *concept drift* as:

Definition 3 (Concept drift). Let an instance $(x_1, \dots, x_m, y)$ be drawn from a feature space $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_m \subseteq \mathbb{R}^m$, where $x_i \in \mathcal{X}_i$ is the i^{th} feature from the corresponding space and $y \in \mathcal{Y}$ is its class label, each drawn from a distribution, which may change over the time. The joint probability of random variables over feature space $\mathcal{X}$ and/or target space $\mathcal{Y}$, $P(X, Y)$, is considered as *concept*, and the change of the joint probability distribution through time is called *concept drift*, where $P_{t_i}(X, Y) \neq P_{t_j}(X, Y)$ [58]. The joint probability can be written as:

$$P(X, Y) = P(Y|X)P(X), \tag{2.22}$$

where the change in the feature variable distribution, $P(X)$, is called *virtual concept drift* or *covariate drift*, and it may occur when the training instances are skewed. On the other hand, the change in the target variable distribution, $P(Y|X)$, is called *real concept drift* or *class shift* [183, 192]. Any of these two changes may or may not lead to a change in posterior distribution, $P(Y|X)$. If the posterior distribution stays the same, the drifts are called *pure class shift* and *pure covariate shift*, respectively.

The ability to distinguish between virtual and real drifts and their pure forms helps to decide if the model has to relearn or revise. Hence, it leads to a more robust model. To this end, there has been much work on identifying types of concept drift [214, 79, 58]. In general, we can categorize a drift according to the way changes happen in time into the following groups:

- Abrupt/Sudden: When the distribution of instances changes instantaneously or near instantaneously. A real-world example of such change happens in a stock market crash, where stock values change almost instantly and follow a new pattern. Many algorithms in the literature investigated to detect sudden drift [195, 57, 29, 53, 16, 130]. They usually track the performance of the algorithm on a buffer/window of recent instances and when there is a significant degradation on the performance from one time slice to the other, the occurrence of drift is indicated.

- Gradual: In case there is a non-deterministic stage and transition time before completing the distribution change, and data may be drawn from both distributions, a *gradual drift* is occurring. Gradual changes may or may not be a steady progression from one concept towards another, where the probability of old distribution decreases and the probability of new distribution increases during the time. This progression can consist of small changes or involve major sudden changes. If the change is a steady progression from c_i toward concept c_{i+1} and the distance from c_i increases and the distance from c_{i+1} decreases in each time step, *incremental drift* is occurring. Thus, incremental drift can be seen as a generalized form of gradual drift in which during the non-deterministic period of distribution change, there are several distributions to draw data from, and the difference between the distributions are small. *Probabilistic drift* is another type of drift where it can be gradual or incremental. Probabilistic drift is when two alternating concepts exist, and one initially predominates, but over time the other comes to predominate. Gradual drifts are usually learned implicitly by updating a single learner [98, 106, 96, 83, 45] or an ensemble of classifiers [114, 115, 196, 49, 171, 170, 188, 106, 99, 159, 60, 199].

- Recurring: If the distribution of data reoccurs after some time, the drift is called *recurrent drift*. If the repetition of the concept(s) occurs in a specific way, a *cyclical* form of drift is emerging. A good example of such case is the weather patterns in one specific location. The focus of our thesis is to detect and learn recurrence of concepts in single-label and multi-label problem setting.

Nonetheless, we should keep in mind that these categories and their definition are rather qualitative and informal. Recently, some studies provided quantitative measures on drifts and tried to establish a formal definition of drifts and how to address the problem of drift detection [123, 192, 62]. Minku *et al.* proposed the *severity* of drift as the proportion of the instance space for which the class labels change between successive stable concepts [123]. *Magnitude* of drift is another measure, which finds the distance between the distribution of concepts at the start and end of the period of drift [192]. Although this measure is simple, the proper function for measuring distribution distance varies from domain to domain and impacts how a learner should respond to the drift. Goldenberg and Webb provided an in-depth survey of different distance measures with respect to their suitability for quantifying and estimating drift magnitude between samples of numeric data [62]. Likewise, Webb *et al.* propose *drift duration* and *drift rate* as two measures, which quantify the elapsed time over a period of drift and how fast the distribution changes in a more formal fashion [192].

An ideal concept drift learning algorithm should support different types of drift, i.e., sudden, gradual and recurring, and at the same time be robust to noise. One challenge in learning from drifting data streams is to distinguish drift from noise. As data is not stationary and drift may occur over time, noise may be recognized as drift and vice versa. Moreover, in the classification of streaming data, there could be cases where a new previously unseen class emerges in time. This problem is called *novel class detection*, and it adds more challenge to the problem of noise detection [119]. However, novel class detection is beyond the scope of this thesis, and we only focus on the problems where all classes are known beforehand.

The validation process of algorithms in conventional data mining with a limited amount of data focuses on maximizing the use of data. *Hold-out*, *cross-validation*, and *leave-one-out* are standard validation methods in those kinds of problems. The hold-out method randomly divides the dataset into two subsets, one for training and one for testing. The $k-$fold cross-validation segments the data into k independent and equally sized sub-samples. One sub-sample is used for testing, and the rest of $k-1$ sub-samples are used for training, and the process is repeated k times so that each sub-sample is used exactly once as test data. The leave-one-out method is a variant of the cross-validation method where the number of folds k is equal to the data size. However, in a data stream environment where data

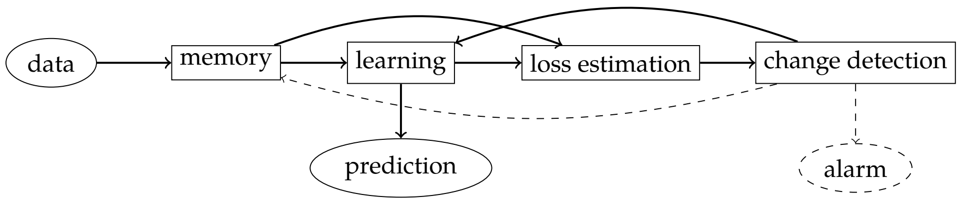


Figure 2.2: A generic four-module schema of an online learning algorithm.

arrives in time and potentially with infinite size, these approaches are not applicable anymore.

A well-known approach is to focus on evaluating the model at various stages by plotting the model's performance over time. This will show how much the model improves with more training data and how well it adapts to drifts. Well-known evaluation measures (i.e., accuracy, misclassification rate, precision, recall, and F-measure) from conventional data classification is applicable here as well. A standard procedure for validation of streaming models is called *prequential* or *Interleaved Test-Then-Train*: Each instance is first treated as a test example, and the current model predicts its class. After receiving its true class, we estimate the loss of our model and incrementally update the model. In fact, the training and the test phases are mixed as opposed to the conventional classification procedure.

Instances arrive in a batch/window or individually at each time slice. Processing data instance by instance releases the need of choosing the window size; however, it obscures the algorithm's performance at a particular time since the model's early mistakes will quickly diminish over time. Processing data in batch allows the model to adapt to the latest changes in the data and is preferred in scenarios with drifts. As a remark, we should consider that the i.i.d (independent identically distribution) assumption is not always valid due to the non-stationary nature of the data stream. However, it is reasonable to assume that the i.i.d assumption holds in small sized windows of data.

2.2.1 Stream mining methods

Gama *et al.* modularize any online adaptive learning algorithm into a four-module schema [58]:

1. **Memory component:** Learning from non-stationary streaming data requires not only to update the predictive model, but also forget the outdated old data. We refer to the memory needed by the model as a long-term memory and to the memory needed for data as a short-term one. Some of the methods in the literature store and process only one instance at a time, while others save multiple instances in a batch and process them once. *WINNOW* [114] is one of the early-proposed well-known examples of the former approach, which is a

robust linear classifier using a multiplicative weight update scheme. STAGGER [157], DWM [98], and GT2FC [22] are other examples of single instance models. *FLORA* [195] is an example of the latter approach, which uses the *first-in-first-out* data structure to store the most recent data and builds a new model on them.

The key challenge in window based methods is the size of the window: a small window of data ensures the data to be stationary, while larger windows get better performance results in stable periods. To benefit from data most, some approaches use a variable window depending on the indications of a change detector (e.g., FLORA2 [195] and others [56, 96, 16]). One study presented a theoretically supported method for finding an appropriate window size for Support Vector Machines based on the estimate of the generalization error [97]. Also, Kuncheva and Zliobate provide a stepping stone towards a theoretical basis of choosing the window size for streams with abrupt drifts and Gaussian classes [104]. In all these approaches, the most recent data is assumed to contain the most relevant and the most important information. However, this assumption is not always correct, especially when the data is noisy or a concept is recurring. Some approaches store the instances not based on their age but based on their distribution usage [154]. Some others do not entirely discard instances but assign weights based on their importance [96]. Single instance algorithms like WINNOW [114] and VFDT [45] do not contain any explicit forgetting mechanism and adapt to the new concept slowly over time by visiting new instances of the new concept.

2. **Change detection component:** Some adaptive learning approaches use an explicit drift detection method, which helps them with providing information about the dynamics of the data process. A typical strategy is to monitor the performance indicators or statistics of data and compare them to a fixed baseline [195, 92]. In general, the methods can be categorized according the following groups:

   - Sequential analysis: The *Sequential Probability Ratio Test* (SPRT) [186] is the basis of several drift detection methods. Assume a sequence of instances where a subset of them is generated from an unknown distribution P_0 and the other subset is generated from another unknown distribution P_1. When the distribution changes from P_0 to P_1 at point t_1, the probability of observing certain subsequences under P_1 is expected to be *significantly* higher than that under P_0. Having independent observations X_i, the null hypothesis for change occurrence is tested by:

$$T_{t_1}^n = \log \frac{P(\mathbf{x}_{t_1} \dots \mathbf{x}_n | P_1)}{P(\mathbf{x}_{t_1} \dots \mathbf{x}_n | P_0)} = \sum_{i=t_1}^{n} \log \frac{P_1[\mathbf{x}_i]}{P_0[\mathbf{x}_i]} = T_{t_1}^{n-1} + \log \frac{P_1[\mathbf{x}_n]}{P_0[\mathbf{x}_n]}, \tag{2.23}$$

where a change is detected if $T_{t_1}^n > \theta$ and θ is a user-defined threshold. The *Cumulative sum* (CUSUM) [136] is a memoryless test that uses SPRT to detect changes. The test outputs an alarm when the mean of the current data significantly deviates from zero. The *Page-Hinkley* (PH) test [136] is a variant of CUSUM for a Gaussian signal. The test variable is the cumulative difference between the observed values and their mean until the current time.

- Statistical process control (SPC): SPC methods are the standard statistical techniques that monitor the evolution of the learning process. The error for each instance is a random variable from Bernoulli trials. For each instance i, the *error rate* is the probability p_i of observing an error with the standard deviation $\delta_i = \sqrt{\frac{p_i(1-p_i)}{i}}$. For a sufficiently large number of instances, the Binomial distribution can be approximated by the Normal distribution with the same mean and standard deviation. The $1 - \delta/2$ confidence interval for p_i will be $p_i \pm \alpha \times \delta_i$. Keeping two parameters p_{min} and δ_{min}, a common threshold for *warning* is 95% with the threshold $p_i + \delta_i \geq p_{min} + 2\delta_{min}$, and for *out of control* is 99% with the threshold $p_i + \delta_i \geq p_{min} + 3\delta_{min}$. We can use the *rate of change* by measuring the time between *warning* and *out of control*. One widely used SPC method is called *Drift Detection Method* (DDM) [56]. Other examples of drift detection methods based on SPC can be found in [65, 22]. *Early Drift Detection Method* (EDDM) [10] improves DDM in better detecting gradual drifts by considering the distance between classification error instead of classification errors. This way, EDDM detects change faster, without increasing the rate of false positives.
- Distribution differences: These methods use two windows of data, one a reference window as a summarization of previous information, and a sliding window over the most recent data. A null hypothesis of equality of distributions is applied to compare these two windows. If the null hypothesis is rejected, a drift is signaled at the start of the current sliding window. These two windows are conceptually different from training windows, and they can be of equal size or progressive sizes, and different window positioning strategies can be employed [1]. In some studies, a statistical test based on the Chernoff bound is used

to compare two distributions [93, 53]. Some other methods use entropy-based metrics [185] or the Kullback-Leibler (KL) divergence [40]. In some methods, the accuracies of the windows are monitored [131] and in some, the summarization of data [16]. The main advantage of this group of drift detection is their more precise localization of the change point compared to the previous approaches, although they need more memory to store instances in two windows.

- Contextual: *Splice* [72] employs a meta-learning technique, which aims to identify stable intervals and induce local concepts associated with these intervals. Some other methods maintain a set of prototypes, which is to represent a class with several clusters [22, 160].

3. **Learning component:** Early learning models emulated incremental learning with batch learning algorithms by discarding the current model and retraining a new model from scratch using buffered data [97, 171]. Following methods attempted to incrementally update the current model using the most recent data. Based on the memory component, the model may be updated by one instance or a buffer of instances. The former is called *online learning* and the latter is known as *incremental learning*. WINNOW [114] and MBW [28] are examples of online learning, and SEA [171] and Learn++.NSE [47] are examples of incremental learning. The ability to continuously learn from a stream of data while preserving previously learned knowledge is known as the *stability-plasticity* dilemma: The model should be on the one hand stable to handle noise and on the other hand being able to learn new knowledge and patterns [27].

   Learning methods follow different adaptation strategies toward drifts. Some methods make a proactive *blind* adaptation by updating the model based on the loss function [195, 97]. VFDT [45] is also an example of such approaches: new examples update statistics in the leaves of the current model; and as the tree grows, the leaves reflect the most recent concepts. The major limitation of this group of methods is their slow reaction to concept drift. Hence, some methods use reactive *informed* strategies based on the triggers from a change detector [83, 15], or data discreptors [195, 92] as in recurring concept methods (cf. Section 2.2.2). The reaction to the drift signal may apply to the whole model or only some regions of the generalization of examples. Most of the existing methods reconstruct the whole model from scratch after detecting a drift [171, 10]. However, some granular methods have the possibility of decomposing the data space and change only the specific affected sections. CVFDT [83] is an example of such methods where with the detection of a change, it starts growing an alternate decision tree in parallel with the root of the newly-invalidated node. When the alternate tree is more accurate than the original tree, it replaces the latter in the main model. Nodes of a Hoeffding tree capture statistics from a data window. The root and the nodes closer to the root are generated by the older instances, while the leaf nodes are generated by the recent instances. By comparing the distribution of the errors at leaves to the distribution of errors at upper nodes, drift is detected. The reaction is to push up all the information of the descending leaves to the corresponding subroot.

From the perspective of the number of learners, some methods train an ensemble of classifiers to obtain a weighted combination of distributions [159, 47]. The weights reflect the performance of individual models on the most recent data. Ensemble updates can be done in three ways [103]:

(a) Dynamic combination, where base learners are trained once and only their combination rule changes over time [114, 115, 195]. It is clear that fixing the classifiers to the ones trained on a limited data may lead to inaccurate prediction in the long run.

(b) Continuous update of the learners, where the learners are either retrained in a batch mode or updated online [171, 134, 59]. This group of methods can be assumed as a variation of the next group.

(c) Structural update or replacement of the poorly performed methods, where new learners are added, and the inefficient ones are removed/deactivated [99, 106].

Studies show that before the occurrence of drift, ensembles with less diversity obtain lower test error, but shortly after drift, highly diverse ensembles reach better performance [123].

4. **Loss estimation:** As discussed before in the validation process of data streams, methods may use different performance measures for measuring the loss (e.g., the leave-one-out error [97]). Many of the methods also use two windows of data, one from the most recent data and one as a reference, to monitor the performance of the algorithm based on their difference [131].

2.2.2 Recurrent concept methods

As training and test phases are mixed in stream mining and the learner is updated iteratively, the old unused concepts learned by the learner are likely to be forgotten, and new concepts are learned as they emerge. After a sufficiently long time, when the old concept reappears, the learner treats it as a new concept and may mislabel most of its instances. This makes the learner inefficient in those real-world problems with such potential. If the learner could avoid forgetting the previously learned knowledge in the classification algorithm, this issue can be resolved appropriately. The methods supporting recurring concepts, try to extract concepts from instances and maintain them in a pool of concepts [55, 65, 92, 106, 125]. When a new instance is received, its similarity to the concepts in the pool is measured, and one or some of the available concepts are selected to predict its class.

Early work on handling recurrent concepts in data streams goes back to FLORA3 [195], which deals with categorical attributes and uses a disjunctive normal form (DNF) language to represent a concept and to train the rule-based algorithm. Ramamurthy and Raj proposed an ensemble model where each classifier was built on a window [145]. Once a classifier is created, it never gets deleted. Instead, at each point, the algorithm chooses relevant classifiers for the ensemble. To filter the classifiers from the global list, the classification error of the classifier on the immediately preceding window is taken into account. All the classifiers which perform better than a random classifier are included in the ensemble set. A classifier is random if the probability of classifying an instance depends solely on the class distribution in the current window. As no classifier is deleted, the algorithm supports recurring concepts.

Inspired by the context space model [135], Gomes *et al.* extract concepts for each classifier [65]. $N-$tuple of form $R = (a_1^R, a_2^R, \dots, a_N^R)$ is called a *context space*, where a_i^R is the acceptable regions of feature a_i. The classifier and its corresponding context space are maintained in the pool, and an explicit drift detection method is used to detect stable concepts. This approach is similar to our approach proposed in Chapter 6, as it extracts the context of a batch of data and trains a classifier on it; however, the representation of context is different from our concept representation.

Later, Lazarescu proposed a method supporting continuous features [106]. The technique keeps an ensemble of classifiers and assigns a concept representation to each classifier. The representation consists of the average of attributes, and any distance measure can measure the similarity. Morshedlou and Barforoush propose to employ the information of mean and standard deviation used in the conceptual features of numeric attributes [125]. The approach uses a pro-active behavior, which means that the next concept's probability is calculated conditional to the current concept. If the concept is likely to occur (its occurrence probability is greater than a threshold), it will be added to a buffer. If a drift is detected and the algorithm decides to behave proactively, the first concept from the buffer is selected. If the chosen concept matches the recent batch, it is updated by batch instances. Otherwise, the algorithm selects the next concept in the buffer, or if it decides to behave reactively, a new classifier is created on the recent batch. A decision on acting reactively or proactively is pursued by a heuristic method. Selecting a suitable probability threshold and proactive or reactive behavior are very time-consuming. Meanwhile, this algorithm supports only the datasets with numeric features; not nominal ones.

A similar approach to Lazarescu's was followed in the *Conceptual Clustering and Prediction* (CCP) framework with some modifications [92]. The CCP framework describes continuous attributes by their means and standard deviations, and nominal features with the probability of attribute values given the class, instead of keeping only the average of attributes. This summary of data is called *conceptual vector* $Z = (z_1, z_2, ..., z_m)$ and is extracted from each batch of data, where z_i is a conceptual feature and is calculated from:

$$z_i = \begin{cases} P(f_i = v|y_j) : i = 1..m, j = 1..k, v \in V_i & \text{if } f_i \text{ is nominal} \\ (\mu_{i,j}, \sigma_{i,j}) : j = 1..k & \text{if } f_i \text{ is continuous,} \end{cases} \tag{2.24}$$

where m is the number of features, k the number of classes, f_i the i^{th} feature and V_i the set of possible values for the nominal feature f_i. $\mu_{i,j}$ and $\sigma_{i,j}$ are mean and standard deviation of the j^{th} class of feature i. By receiving a new batch of data, the method decides whether to update one existing classifier or add a new classifier to the pool. In the CCP framework, the Euclidean distance is used as a similarity measure for conceptual vector comparisons. If the similarity of recent conceptual vector to a concept available in the pool is higher than a predefined threshold, its corresponding classifier will be updated by instances of the recent window. Otherwise, a new classifier and concept is added. As the threshold parameter is problem specific, one major shortcoming of the CCP framework is how to determine the threshold. In some problems, features are not independent of each other, in which case CCP's conceptual vectors fail to extract this information from data. Including feature correlations can help in a more accurate approximation of the data distribution and better drift detection in case of drifts in feature correlations. We have extended the idea of representing dependencies between the features in conceptual vectors in Chapter 6. Furthermore, instead of using the Euclidean distance, we first present a heuristic method in Chapter 5, and then we propose a multivariate statistical test to check if two conceptual vectors are drawn from different distributions in Chapter 6.

Another approach keeps only a pool of classifiers, but no concept representation [201]. Instead of comparing representations, *RePro* [201] takes the number of similar predictions of the classifiers on the current batch as the similarity measure. This method uses a Markov chain to learn concept transition patterns. In contrast to our proposed method in Chapter 6, this method follows a proactive approach and updates the transitions in the Markov chain whenever drift has been detected; then it moves to the next state. Data is predicted using a proactive-reactive method: If the accuracy of the next classifier is better than a threshold, it will be used as the next state; otherwise, a new classifier will be trained on the current batch. Our experimental results showed that RePro takes more time in finding similar classifiers on each new batch of data than methods using concept representations.

A recent framework called *Recurring Concept Drift* (RCD) [66] creates a new classifier for any new concept and keeps a fixed size buffer for data samples used to build the classifier. If a drift is detected, the method calculates a non-parametric multivariate statistical test to compare the similarity of a new concept with all the previous concepts in the pool. If the new data is similar to one of the earlier concepts, it is counted as a concept recurrence, and the old classifier is reused. If not, a new classifier and its buffer are added to the pool. This approach is different from our approach in Chapter 6 in several aspects: Besides the different statistical test used in finding similar concepts, this method keeps a fixed size FIFO buffer instead of the conceptual vectors in our approach. We do not use any separate drift detection method in a way RCD does. Moreover, RCD does not extract transitions between the concepts of the pool; it just keeps a list of classifiers and their sample buffers.

Gama and Kosina presented a meta-learning approach that meta-classifiers select which base classifier is used when a drift is detected [55]. If the output of the meta-learner is greater than a predefined threshold, the base learner will be used to label the instance; otherwise, it is excluded. Here also a pool is used to keep all base and meta-learners. There also have been other frameworks that add a meta-learning layer to the pool of classifiers to characterize the domain of applicability of the learned models [54, 9]. The meta-layer is a control layer that monitors the evolution of learning algorithms. *MM-PRec* [9] uses a *Hidden Markov Model* (HMM) as a meta-model to predict if a drift will happen as well as to choose the best concept if there is a recurrence. The framework uses an explicit drift detection module: In the case of a warning, the current classifier is compared to the classifiers in the repository using a fuzzy similarity function. If the current classifier or a similar one is not available in the repository, it is added to the repository as a new concept. In order to get the prediction of the meta-model, MM-PRec keeps a buffer of multi-instance data and will send it to the HMM whenever the buffer gets full. This may delay the training process of HMM, and it is considered as the main drawback of MM-PRec. When drift is signaled, a model is trained on a stable sub-window of instances, and it is compared to the models in the repository. In case the newly trained model is detected as a recurrent model, the one existing in the repository is used in the future, and the new model is discarded. Moreover, the recurrent models are not incrementally trained with the new data anymore.

2.2.3 Multi-label stream learning methods

Although multi-label classification has received a lot of attention, classifying multi-label data streams is relatively new and not very well investigated. A *multi-label data stream* is defined as:

Definition 4 (Multi-label data stream). Let $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^m$ and $\boldsymbol{l} = (l_1, l_2, \ldots, l_l) \in \{0,1\}^l = \mathcal{L}$ denote an instance of an m-dimensional feature space and the relevant binary label set of size l, respectively. At each time stamp, a new batch of multi-labeled instances of size n is received: $(X(t), L(t)) = \{(\boldsymbol{x}_1^t, \boldsymbol{l}_1^t), (\boldsymbol{x}_2^t, \boldsymbol{l}_2^t), \ldots, (\boldsymbol{x}_n^t, \boldsymbol{l}_n^t)\}$, where $(\boldsymbol{x}_i^t, \boldsymbol{l}_i^t)$ indicates the i^{th} multi-labeled instance in batch t. We assume that instances are generated independently from a joint distribution $P(X, L)$ over $\mathcal{X} \times \mathcal{L}$.

We can extend the definition of concept drift to the multi-label scenario, considering:

$$P(X, L) = P(L|X)P(X) = P(X|L)P(L). \tag{2.25}$$

The change in the joint probability distribution can be derived from the change in the prior feature distribution $P(X)$, or the change in the likelihood label distribution given the feature set, $P(L|X)$, or the change in the posterior distribution over the feature set given the label set, $P(X|L)$, or the change of distribution among class labels $P(L)$, or any combination of them. The general categorizations of drift in single-label streams (such as abrupt, gradual, recurrent, etc.) are still applicable here [192].

As instances in data streams are available only for a short period, and their underlying distribution may vary over time, special algorithms are developed to deal with their features as explained in Section 2.2.2. However, in the multi-label context, apart from some well-known multi-label methods like Binary Relevance (BR) and Classifier Chains (CC) that can easily be upgraded to the online scenario by using an updateable classifier as their base learner, there are limited studies that address the problem of multi-label stream classification.

Early work provides a stacking modification of the BR algorithm (MBR) to learn the dependencies among the labels by adding each classified label as a new feature [141]. To cope with drifts, an implicit strategy is followed by taking the weighted majority vote of a dynamic classifier ensemble. *Multiple Windows Classifier* (MWC) is a modified version of k-Nearest Neighbor and considers the class imbalance problem by maintaining two fixed-sized windows per label, one for positive and one for negative examples [168]. This method also has an implicit strategy for dealing with drifts. The same implicit strategy is used in other work [191, 190].

Later, Read *et al.* proposed a multi-label version of Hoeffding trees (MLHT) by modifying the definition of entropy and training multi-label prune sets at the leaf nodes [147]. MLHT uses the definition of multi-label entropy proposed by Clare and King [36] in adapting C4.5 to the multi-label setting (equation (2.13)). MLHT also benefits from the ADWIN bagging method [18] as an implicit change detector, thus, upon the occurrence of a drift, the worst performing classifier of the ensemble is replaced by a new one. Experimental results show that the MLHT method outperforms MBR and MWC [147]. Shi *et al.* [161] also presented a multi-label version of entropy as a simple drift control strategy to monitor the change of distribution between features and labels. In their work, they capture the label correlation and interdependence based on two different approaches: the Apriori association rule mining algorithm and the Expectation-Maximization clustering method. Then each instance is annotated with some subsets of the class label as the new class labels. There may exist overlapping or non-overlapping subsets. The experimental results demonstrated the advantage of the method based on label grouping and entropy over the method only based on entropy. This indicates that considering label dependence is effective in detecting concept drift for multi-label data streams.

Multi-Label Dynamic Ensemble (MLDE) integrates an ensemble of *Multi-Label Cluster-based Classifiers* (MLCCs) and measures the appropriateness of them for classifying a new concept by the subset accuracy weight [166]. This approach uses a sophisticated and time-consuming multi-label cluster-based classifier, which makes it impractical for larger datasets. A recent method monitors the average local log-likelihood of nodes in a multi-dimensional Bayesian network classifier (MBC) using the Page-Hinkley test [20]. However, none of these studies consider recurrent concepts in multi-label streams.

2.3 Deep neural networks

Neural networks have received so much attention from researchers recently mainly because they do not require handcrafted features, and can discover the features during the learning process. Nevertheless, for a long time, training networks with a larger number of layers was not successful, and methods like Support Vector Machines were more useful in practical applications. Last decade witnessed impressive progress with *Deep Neural Networks* (DNN), and the proposed approaches significantly pushed the state of the art on many difficult problems such as image recognition [162] and speech recognition [76]. The network architectures vary from feedforward neural networks to radial basis networks, to recursive and recurrent neural networks, and to convolutional neural networks, or a combination of them.

Deep feedforward networks are the backbone of modern deep learning methods. One important architecture in this category is the *multi-layer perceptron* (MLP). The goal here, the same as in other classification problems, is to find the best approximation of a function f that maps the input to the desired output. The mapping is implemented by multiple layers of neurons. A neuron is a simple computation unit that makes a weighted connection to the neurons of a subsequent layer:

$$y = \phi(\sum_i w_i x_i + b), \tag{2.26}$$

where x_i is the i^{th} input, w_i is its corresponding weight, b is the bias, ϕ is the activation function, and y is the output of the neuron. Historically, common activation functions were the logistic sigmoid function ($\phi(x) = \frac{1}{1+e^{-x}}$) or the hyperbolic tangent function ($\phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$). An output unit determines the type of loss function. The most popular choices for classification purposes are sigmoid and softmax units. Sigmoid units are suitable for binary classification since the result of a sigmoid function can be interpreted as class probability. In this case, an appropriate cost function is the *cross entrpy loss*:

$$J_i(\mathbf{x}_i, y_i; \theta) = -\big(y'_i \ln y_i + (1 - y'_i)\ln(1 - y'_i)\big), \tag{2.27}$$

where y_i, y'_i are the desired output and the sigmoid output, respectively. For multi-class problems, however, the *softmax* function is a perfect choice:

$$y(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \tag{2.28}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ is the input vector to the softmax layer. The cross entropy loss function can be easily extended to c classes:

$$J_i(\mathbf{x}_i, y_i; \theta) = -\sum_{j=1}^{c} y'^{j}_{i} \ln y^{j}_{i}. \tag{2.29}$$

To find an accurate mapping function f, the parameters should be adjusted. For that, the objective function, i.e. the average loss over training instances, is to be minimized. The procedure starts from the output layer, and by applying the chain rule to derivatives, it propagates the error to the previous layers. The method is called *backpropagation* [152]. When the gradients are known, a gradient-based optimization algorithm can find the best parameter for the objective function. In practice, the *Stochastic Gradient Descent* (SGD), a stochastic approximation of the gradient optimization method over training instances, is calculated. However, updating the parameters with every single instance is an extremely inefficient method, therefore, many methods use a mini-batch update approach. In addition, the number of parameters in a neural network can be very high, which makes the training prone to overfitting. One way to avoid such issue is to apply *regularization*. One simple regularization method is to add a penalty on the magnitudes of the parameters:

$$\bar{J}(\mathbf{X}, \mathbf{y}; \theta) = J(\mathbf{X}, \mathbf{y}; \theta) + \lambda w(\theta), \tag{2.30}$$

where λ is the regularization hyperparameter controling. The most common penalty norms are the L2 regularization ($w(\theta) = \frac{1}{2}||\mathbf{w}||_2^2$) and the L1 regularization ($w(\theta) = ||\mathbf{w}||_1$). Parameter norm penalty regularization is not a specific method for neural networks. Recently, *dropout* regularization has been proposed specifically for reducing overfitting in deep neural networks [169]. It randomly disables a subset of neurons each time a minibatch is processed. The probability of keeping a hidden neuron is usually set to 0.5, and the input neuron is preserved with 0.8. In the test phase, all the neurons are involved. MLP is a feedforward network because there are no backward connections and the flow of computations is only from input towards the output. In this aspect, the feedforward network differs from recurrent neural networks. In this section, we briefly review three other famous categories of neural networks: *convolutional neural networks*, *recursive neural networks*, and *autoencoders*.

2.3.1 Convolutional neural networks

A Convolutional Neural Network (CNN) [108] can be thought of a *multilayer perceptron* for processing spatially structured data (e.g., image). Assuming that the data is organized in a grid-like structure, CNNs can greatly reduce the number of parameters, and hence, speed up training. Apart from the classical fully-connected layers, CNNs use two other types of layers: convolutional layers and pooling layers. Neurons in convolutional layers are grouped in feature maps to detect local features in the input data. Thus, instead of being connected to all the units from the previous layer, each neuron is connected only to a small region of its input, called the *receptive field*. Weights of these connections form a filter, which is convolved with the input to produce the *activation map*. A well-known activation function is the *rectifier linear* function:

$$\phi = \max\{0, \sum_i w_i x_i + b\}, \tag{2.31}$$

where x_i is the i^{th} feature of input instance, w_i is its corresponding weight, and b is the bias. Neurons with this activation function are commonly called *Rectifier Linear Units* (ReLUs). Convolution layers are interleaved with pooling layers to reduce the computation burden for subsequent layers and control overfitting. A pooling layer downsamples the data representation by combining the output of the previous feature map into non-overlapping clusters. The most common pooling method is *max pooling*, which then returns the maximum value of each cluster. A CNN could be a repeat of one or several layers convolutional layers followed by a pooling layer, before employing one or many fully-connected layers. A CNN can be trained with the *Stochastic Gradient Descent* (SGD) and backpropagation, as all the operations performed by the layers and neurons are differentiable. Convolutional neural networks existed long before the advent of deep learning; however, their popularity escalated after the domination of deep CNNs in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition in 2012 [102].

2.3.2 Recursive neural networks

An MLP can only map from input to output vectors, whereas a recursive neural network (RecNN) in principle can map from the entire history of previous inputs to each output. A recursive neural network applies the same set of weights recursively over a structured input [164]. In its basic architecture, nodes are combined into parents using a shared weight matrix, and a non-linear activation function:

$$p(i,j) = \phi(\mathbf{W}[\mathbf{c}_i; \mathbf{c}j] + b), \tag{2.32}$$

where $\mathbf{c}_i$ and $\mathbf{c}_j$ are the $n-$dimensional vector representation of nodes i and j, and $p(i,j)$ is their parent's vector representation with the same dimensions. $\mathbf{W}$ is the learned $n \times 2n$ weight matrix, b is the bias, and a typical example for ϕ is $tanh$. With this representation, we can compute a local score using an inner product with a row vector $\mathbf{W}_{score} \in \mathbb{R}^{1\times n}$:

$$a(i,j) = \mathbf{W}_{score}p(i,j). \tag{2.33}$$

This type of network is useful when dealing with variable-size inputs and when the topological structure of the input is important.

Recurrent Neural Networks (RNNs) [78] are a type of recursive networks with a linear chain structure. While a recursive neural network operates on any hierarchical structure, recurrent neural networks combine the previous time step and a hidden representation into the current representation, hence, operate on a linear progression of time. The forward pass of RNNs is the same as an MLP with a single hidden layer, except that a hidden layer receives input from both the current external input and the hidden layer of one step back in time:

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{j=1}^{H} w_{jh} \phi_h(a_h^{t-1}), \tag{2.34}$$

where I and H are the number of input and hidden units of the RNN, x_i^t is the i^{th} input value at time t, a_j^t is the input to unit j at time t, and ϕ is a non-linear activation function. The complete sequence of activations can be computed by starting at $t = 1$ and recursively applying equation (2.34). For the backward pass, we can apply backpropagation through time [198].

2.3.3 Autoencoder

The autoencoder is an MLP whose goal is to reconstruct the input at the output. Therefore, it has the same number of input and output neurons. Middle layers have a smaller number of neurons, and the smallest layer stands exactly in the middle of the architecture. The autoencoder consists of two parts, the encoder, and the decoder. The first part of the network tries to compress the input to an encoded representation $\boldsymbol{c} = e(\boldsymbol{x})$, and the second half calculates the reconstruction $d(\boldsymbol{c})$. The objective function is to minimize the dissimilarity between $\boldsymbol{x}$ and $d(\boldsymbol{c})$. One may add additional regularization terms to include various desired properties, such as sparsity [146] or robustness to minor changes [151].

Autoencoders were developed long before the advent of deep networks [23]; however, the discovery of *generative pretraining* [77] opened the possibility of training much deeper models.

2.3.4 Multi-label deep learning methods

Deep neural network architectures have opened their way into multi-label learning in recent years. Read and Perez-Cruz showed that employing *Deep Belief Networks* (DBNs) [77] yields a better feature space representation and hence, better performance [148]. Wicker *et al.* apply autoencoders as a label space dimension reduction approach to extract non-linear dependencies among the labels [194]. *Canonical-Correlated Autoencoder* (C2AE) [203] is another deep method that combines deep canonical correlation analysis (DCCA) and autoencoders to learn a feature-aware latent subspace for label embedding. It also enhances the loss function at the decoding output to be label-correlation aware, and hence, better exploit cross-label dependencies. However, the training of such networks is intensely time-consuming, and thus only applicable to very small datasets with not many labels.

Recently, some methods combine the approach in Classifier Chains with recurrent neural networks. *CNN-RNN* [189] replaces the memory mechanism in Classifier Chains with an RNN-based approach. It learns a linear joint image-label embedding to characterize both the semantic label dependency and image-label relevance. To better extract hidden correlations, *RethinkNet* [200] modifies Classifier Chains by forming a chain of multi-label classifiers. It adopts RNN for making memory-based predictions by fully memorizing the temporary predictions from all classifiers. Nam *et al.* [128] present an alternative formulation of Probabilistic Classifier Chains using recurrent neural networks by only focusing on the positive labels. Lately, an order-free visually attended RNN method (*Att-RNN*) [32] is proposed to solve the problem of pre-determining the label order in CC and CNN-RNN. It furthermore improves the performance on image classification by introducing a confidence-ranked Long Short-Term Memory (LSTM) network. Att-RNN approximates the optimal order of labels with a time-consuming beam search.

Chapter 3

Deep Learning for Text Classification

The advent of social media such as Twitter, blogs, ratings, and reviews has created a surge of research on analyzing text data for different purposes such as sentiment analysis. Much of this data is in the form of short texts such as sentences. Training a well-performing classifier for a specific task on a single sentence that has a limited amount of contextual data is a challenging task. To effectively solve this problem, one may model sentences to analyze and represent their semantic content. The goal of modeling sentences is to accurately represent their meaning and semantic content for different tasks including classification. Recent years have seen a variety of different deep learning architectures considered especially to model sentences [121, 165, 90]. Their significant advantages lie on the removed requirements for feature engineering, and preservation of the order of words and syntactic structures, in contrast to the traditional bag-of-words model, where sentences are encoded as unordered collections of words. The success of such neural network-based methods is also based on the progress in learning distributed word representations in semantic vector spaces [122, 138], where a real-valued word embedding represents each word. Word embeddings are learned by projecting words onto a low-dimensional vector space that encodes both semantic and syntactic features of words. Given word embeddings, different methods have been proposed to learn word compositions and to model sentences. These neural network approaches range from basic *Neural Bag-of-Words* (*NBoW*), which ignores word orderings to more representative compositional approaches such as *Recursive Neural Networks* (*RecNNs*) [163, 85], *Convolutional Neural Networks* (*CNNs*) [94, 90], and *Recurrent Neural Networks* (*RNNs*) [110, 213] or a combination of them (e.g. CNN and RNN [116, 126], or RecNN and RNN [173, 25]).

Recursive neural networks are a generalization of classic sequence modeling neural networks to tree structures and have shown excellent abilities to model word combinations in a sentence [140, 63, 75]. They work by feeding a parse tree to the network. At every node in the tree, the composition is done in a bottom-up fashion by a weight matrix shared over all nodes of the tree. However, they depend on well-performing parsers to provide the topological structure. These are not available for many languages and do not perform well in noisy domains. Further, they often require labeling of all phrases in sentences to reduce the so-called *vanishing gradient problem* [86]. Recurrent neural networks are a special case of recursive networks where their structure is linear instead of a tree [78, 121]. An in-depth comparison of RecNN and RNN showed that when long-distance semantic dependencies play a role, recursive models offer useful power [110]. Although Recursive neural networks implicitly model the interaction among input vectors, *Recursive Neural Tensor Networks* (*RNTNs*) have been proposed to allow more explicit interactions [165].

On the other hand, convolutional neural networks models, as the alternative models for sentence modeling, apply one-dimensional convolution kernels sequentially on word vectors using sliding windows to extract local features. Each sentence is treated individually as a bag of n-grams, and long-range dependency information spanning multiple sliding windows is therefore lost [208]. Recently, new architectures have been proposed to resolve the limitation of CNNs in losing long-range dependency information [116, 208], or to overcome the fixed structure of CNNs for one input length [90]. A recent comparison of CNNs and RNNs has shown that in those tasks where recognizing a keyphrase is essential (e.g., sentiment analysis or question-answer matching), CNNs perform better [204]. Another limitation of CNN models is their requirement for the exact specification of their architecture and hyperparameters [209].

Although there are few studies on comparing different deep network architectures, little is known about the comparative performance of recursive neural networks and convolutional neural networks on a common ground, across a variety of datasets, and on the same level of optimization. In this chapter, we compare convolutional neural networks to a well-performing variation of recursive neural networks called recursive Neural Tensor Networks (RNTNs). Although RNTNs have shown to work well in many cases, they still suffer from the need for intensive manual labeling to overcome the vanishing gradient problem. In this chapter, we conduct extensive experiments over a range of benchmark datasets to compare the two network architectures. Our goal is to provide an in-depth analysis of how these models perform across different settings. Such a comparison is missing in the literature, likely because recursive networks often require labor-intensive manual labeling of phrases. Such annotations are unavailable for many benchmark datasets. We employ two methods to automatically label the internal nodes: a rule-based method and (this time as part of the RNTN method) a convolutional neural network. This enables us to compare these RNTN models to

$$p_3 = f\left(\begin{bmatrix} w_1 \\ p_2 \end{bmatrix}^T \mathbf{V}^{[1:d]} \begin{bmatrix} w_1 \\ p_2 \end{bmatrix} + \mathbf{W} \begin{bmatrix} w_1 \\ p_2 \end{bmatrix}\right)$$

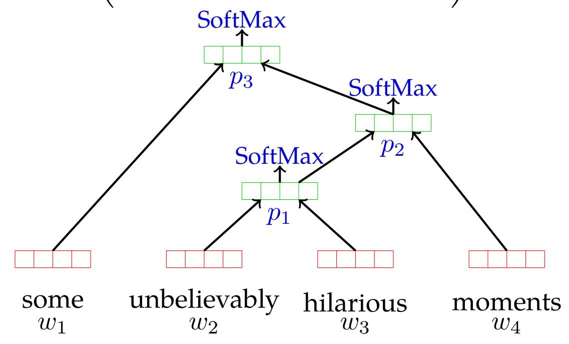


Figure 3.1: An example of an RNTN architecture with word vector dimension of size 4 for sentiment classification of a given input sequence, which is parsed by a constituency parser. **V** and **W** are the tensor matrix and the recursive weight matrix, respectively.

a relatively simple CNN architecture. We also investigate whether there is an effect of using constituency parsing instead of dependency parsing in the RNTN model. In this way, we aim to contribute to a better understanding of the limitations of the two network models and provide a foundation for their further improvement.

In this following sections, we first explain our approach to the automatic labeling of RNTNs and then explain our proposed architecture for the CNN. The presented architectures are then evaluated on a set of benchmark datasets, described in Section 3.3.

3.1 Recursive neural tensor network architecture

RNTNs [165] are a generalization of RecNNs where the interactions among input vectors are encoded in a single composition function (Figure 3.1). Here, we propose two methods for the automatic labelling of the phrases for RNTNs:

- **Rule-based method:** The RNTN model was first proposed for sentiment analysis purposes. Hence, our first approach uses a rule-based method to determine the valence of a phrase. We use four types of dictionaries: A dictionary of sentiments carrying terms (from unigrams to phrases consisting of n-gram words) with a corresponding sentiment score in the range of $[-k, +k]$, a negation dictionary, a dictionary of intensifier terms with a weight range of $[1, +k]$, and a dictionary of diminishers with a weight range of $[-k, -1]$.

  The analysis of a phrase is conducted from the end, backward to the

ALGORITHM 1: Rule-based labeling method

```
Input: a sequence of $< w_1 w_2 \ldots w_n >$ words;
       dictionary of sentiment with $[-k, +k]$ range,
       dictionary of intensifiers with $[+1, +k]$ range,
       dictionary of diminishers with $[-k, -1]$ range,
       and dictionary of negation
Output: $sentiment$ of the input sequence
1  $sentiment := 0\,; i := n$
2  while $i > 1$ do
3      if $w_i$ is a term in the dictionary of sentiments then
4          $sentiment :=$ sentiment value of $w_i$
5          $i := i - 1\,; flag := false$
6          if $w_i$ is a term in the dictionary of intensifiers then
7              $|sentiment| := min(|sentiment| + k_i, k)$
8              $i := i - 1\,; flag := true$
9          else if $w_i$ is a term in the dictionary of diminishers then
10             $|sentiment| := max(1, |sentiment| - k_i)$
11             $i := i - 1\,; flag := true$
12         if $w_i$ is a term in the dictionary of negations then
13             if $\sim flag$ then
14                 $sentiment := -sentiment$
               else
15                 $sentiment := -1$
       else
16         continue the search for $< w_1 \ldots w_{i-1} >$
```

beginning (Algorithm 1)[1] : If any sentiment term is found (line 3), we update the sentiment of the phrase from neutral to the value of the sentiment term in the dictionary (line 4). Then we search backward for an intensifier or diminisher term, which may also consist of more than one word. We increase or decrease the absolute value of the sentiment based on the weight of the intensifier/diminisher term, and if required, we adjust the score to a pre-defined range (line 6– 11). In the next step, we adjust the score for a negation term. If one is found and there is no intensifier/diminisher before the sentiment term, the sentiment is reversed (lines 13– 14); otherwise, if the phrase includes both the negation term and an intensifier/diminisher, the sentiment is set to weak negative (line 15). As an example, consider both *"not very good"* and *"not very bad"* terms, where both sentiments are weak negative.

- **CNN-based method:** An alternative approach to labeling the phrases

[1]This algorithm was developed as a collaboration with the NLP team in the company PRIME Research International AG & Co. KG.

is to use a pre-trained CNN model. We use the architecture proposed here (see below for the description) to train a model on the sentence level and use the resulting model to label the internal phrases for the RNTN. In this way, the RNTN can be applied to domains other than sentiment classification as well. The CNN model receives the complete sentences and their label as training data and will label the internal phrases in the test phase.

3.2 Convolutional neural network architecture

Deep convolutional neural networks have led to a series of breakthrough results in image classification. Although recent evidence shows that network depth is of crucial importance to obtain better results [73, 37], most of the models in the sentiment analysis and sentence modeling literature use a more shallow architecture, e.g., Kim uses a one-layer CNN [94]. Inspired by the success of CNNs in image classification, our goal is to expand the convolution and Max-Pooling layers in order to achieve better performance by deepening the models and adding higher non-linearity to the structure. However, deeper models are also more difficult to train [73]. To reduce the computational complexity, we choose small filter sizes. In our experiments, we use a simple CNN model that consists of six layers (Figure 3.2): The first layer applies $1 \times d$ filters to the word vectors, where d is the word vector dimension. The essence of adding such a layer to the network is to derive more meaningful features from word vectors for every single word before feeding them to the rest of the network. This helps us achieve better performance since the original word vectors capture only sparse information about the words' labels. In contrast to our proposed layer, a related approach [94] uses a so-called *non-static* approach, which modifies the word vectors through the training phase.

The second layer of our CNN model is again a convolution layer with the filters of size $2 \times d$. The output of this layer is fed into a Max-Pooling layer with pooling size and stride 2. The reason for applying such a Max-Pooling layer in the middle layers of the network is to reduce the dimensionality and to speed up the training phase. This layer does not have a notable effect on the accuracy of the resulting model. Next, on the fourth layer, convolving filters of size $2 \times d$ with a padding size of 1 are again applied to the output of the previous layer. Padding preserves the original input size. The next layer applies Max-Pooling to the whole input at once. Using bigger pooling sizes leads to better results [209]. Finally, the last layer is a fully connected SoftMax layer, which outputs the probability distribution over the labels.

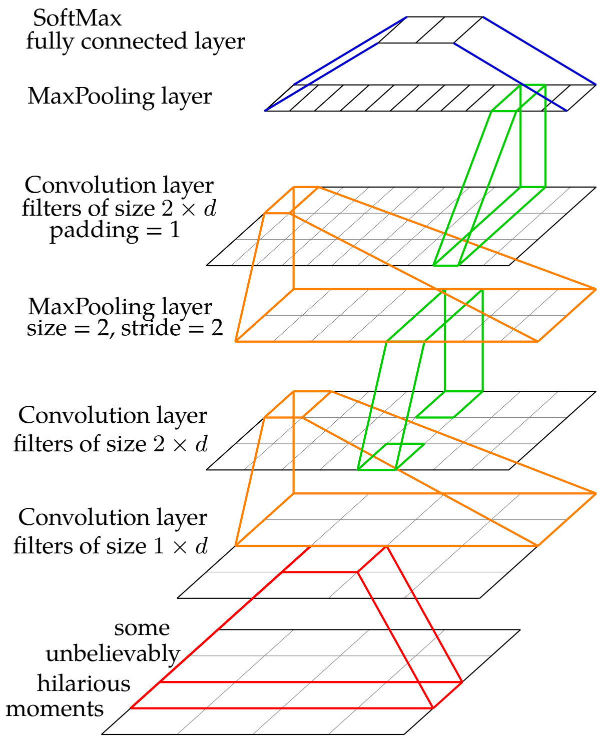


Figure 3.2: Our proposed 6-layered CNN architecture. d is the dimension of the word vector.

3.3 Experimental results

In this section, we first introduce the experimental settings, then we investigate the variants of RNTNs and compare their performance to the proposed CNN architecture on different tasks.

3.3.1 Experimental settings

In our experiments, we use the pre-trained Glove [138] word vector models[2]: On the SemEval-2016 dataset, we use Twitter specific word vectors that were trained on 2 billion tweets. On other datasets, we use the model trained on the web data from Common Crawl, which contains a case-sensitive vocabulary of size 2.2 million. Experiments show that RNTNs work best when the word vector dimension is set between 25 and 35 [165]. Hence, in all the experiments, the size of the word vector, the minibatch, and the epochs was set to 25, 20 and 100, respectively. We use $f = tanh$ and a learning rate of 0.01 in all the RNTN models. In CNN models, the number of filters in the convolutional layers are set to 100, 200 and 300, respectively; and the maximum length of the sentences is 32. For shorter sentences, they are padded

[2]`http://nlp.stanford.edu/projects/glove/`

Table 3.1: Summary statistics for the sentiment datasets. c, N_{tr}, N_{tu} and N_{ts} indicate the number of labels, number of training sentences, number of tuning sentences and the number of test sentences, respectively.

Dataset	c	N_{tr}/N_{tu}	N_{ts}
MR	2	10662	CV
SemEval-2016	3	12644/3001	20632
SST-5	5	8544/1101	2210

with zero vectors. In RNTN models which use constituency parsers, we use the Stanford parser [95]. For those models which use dependency parsers, we use the Tweebo parser [100] – a dependency parser specifically developed for Twitter data – for the SemEval-2016 dataset and on the rest of the datasets, we use the Stanford neural network dependency parser [31]. In rule-based methods, we use a dictionary of sentiments consisting of $6,360$ entries with a maximum 2-grams words and a sentiment range of $[-3, +3]$, a negation dictionary consisting of 28 entries, a dictionary of intensifier terms consisting of 47 words with a weight range of $[1, 3]$, and a dictionary of diminishers consisting of 26 entries with a weight range of $[-3, -1]$.

3.3.2 Task 1: sentiment analysis

In this section, we present the results of automatic labeling of phrases, the effect of the selected parser type, and describe the overall evaluation results for the presented RNTN and CNN models. Next, we discuss the effect of automatic labeling on the performance of the RNTN. We compare the models on a set of commonly used sentiment analysis benchmark datasets (Table 3.1): The Movie Review (**MR**) dataset[3] was extracted from Rotten Tomato reviews [137], where the reviews can be positive or negative. As the MR dataset does not have a separate test set, we use 10-fold cross-validation in the experiments. An extended version of the MR dataset relabeled by Socher *et al.* [165] in the Stanford Sentiment Treebank (**SST-5**)[4] has five fine-grained labels: negative, somewhat negative, neutral, somewhat positive and positive. The **SemEval-2016**[5] dataset is a set of tweets labeled as either of the three negative, neutral and positive labels.

- **Comparison of automatic labeling methods:** We first use the manually labeled SST-5 dataset to test the effectiveness of our automatic labeling methods. We extract all the possible phrases of the whole dataset with respect to their parse trees and use our rule-based method to label them. In the next step, we train the CNN model

[3]`https://www.cs.cornell.edu/people/pabo/movie-review-data/`

[4]`http://nlp.stanford.edu/sentiment/Data`

[5]`http://alt.qcri.org/semeval2016/task4/`

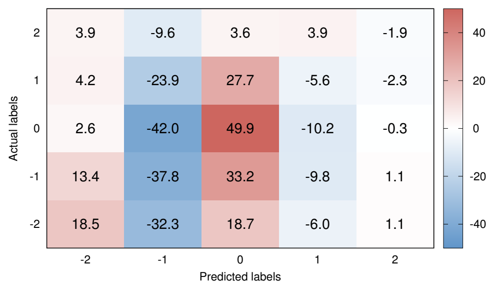


Figure 3.3: Heatmap of difference of rule-based RNTN and CNN-based RNTN confusion matrices on the SST-5 phrase set. The numbers are the percentage of normalized differences based on the total number of phrases for each label.

on the set of training instances and use the resulting model to label the phrases. The accuracy of the rule-based method and the CNN model labeling are 69% and 40%, respectively. As we see, the overall accuracy of the CNN-based model is significantly lower than that of the rule-based method. To have a better understanding of the classification performance on each type of label, we look into their confusion matrices. We subtract the corresponding elements of the CNN-based confusion matrix from that of the rule-based variant and normalize them by dividing by the total number of phrases for each label (i.e., $\frac{conf^{i,j}_{rule} - conf^{i,j}_{cnn}}{total_i}$ where i and j are the actual and predicted labels, respectively). Figure 3.3 shows the resulting heatmap. Red color indicates cases where more phrases are predicted by the rule-based method than by the CNN-based method, while the blue color indicates the opposite case. We observe that CNN is a better model to correctly classify somewhat positive (1) and somewhat negative (-1) classes than the rule-based method. In turn, the rule-based method is superior in the classification of the neutral (0) and negative (-2) classes. To have a better interpretation of the numbers in the heatmap, it is beneficial to look at the distribution of labels in the whole population: 2.6%, 11.3%, $67,7\%$, 14.3% and 4.1% for -2 to $+2$ labels.

- **Constituency parser vs. dependency parser:** The output of a dependency parser is a Directed Acyclic Graph (DAG). However, RNTNs accept a binary-branching parse tree as input. Therefore, we have binarized the output of dependency parsers by using the method presented in Algorithm 2. We start from a word which does not point to any other word as its parent, and recursively binarize its children list by adding empty nodes when necessary.

  When analyzing the effect of using a dependency parser instead of a constituency parser in RNTNs (Table 3.2), for some datasets (e.g.,

ALGORITHM 2: Binarize dependency DAG method

```
Function(binarizeTree)
Input: dependency graph with $root$ node
1  if $root$ has no children then
2      return $root$
3  else if $root$ has one child then
4      BinarizeTree($root.child$)
5      return $root$ with binarized sub-tree
6  else if $root$ has two children then
7      make $tempRoot$ node with $root$ data
8      $tempRoot.leftChild :=$ BinarizeTree($root.child(0)$)
9      $tempRoot.rightChild :=$ BinarizeTree($root.child(1)$)
10     return $tempRoot$
11 else if $root$ has more than two children then
12     make $tempRoot$ node with $root$ data
13     if $tempRoot.data = empty$ then
14         $tempRoot :=$ binarizeSubTree($root.children$)
       else
15         $tempRoot.leftChild :=$ binarizeSubTree($root.children$)

Function(binarizeSubTree)
Input: list of $children$ nodes
16 if $children.size = 1$ then
17     return binarizeTree($children.remove(0)$)
18 make $tempRoot$ node
19 $tempRoot.leftChild := children.remove(0)$
20 $tempRoot.rightChild :=$ binarizeSubTree($children$)
21 return $tempRoot$
```

MR) a significant loss of performance is visible. This is particularly noticeable when the labeling method is CNN (e.g., 70% to 49% in MR). The reason for this could be the difference of the word order resulting from a dependency parser compared to the n-gram features extracted by the CNN.

- **RNTN vs. CNN:** Table 3.2 shows a detailed comparison of the RNTN automatic labeled variants to the CNN model and the rule-based method. We have reported the average accuracy and F-measure over all classes. With the same settings of parameters, we see a better performance of the CNN model on the MR and SemEval-2016 datasets. The most noteworthy performance (in terms of F-measure) improvement can be observed on the SemEval-2016 dataset, 0.51 to 0.56, for the best performing RNTN and CNN approaches. The possible reasons may be related to the enormously large number of parameters that have to be optimized in the tensor and the effects

Table 3.2: Performance comparison on all datasets. Accuracy and F-measure are averaged over all the classes. n/a indicates non-defined cases as one of the classes was misclassified completely resulting in an undefined value. If an experiment was not applicable, the cell is left with a dash.

Dataset	RNTN										CNN		CNN (Kim model)		Rule-based	
	Constituency parser						Dependency parser									
	Rule		CNN		Manual		Rule		CNN							
	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1
MR	0.63	0.63	0.70	0.70	-	-	0.50	0.50	0.49	0.49	**0.71**	**0.71**	**0.71**	**0.71**	0.64	0.64
SemEval-2016	0.53	0.45	0.52	0.51	-	-	0.52	0.45	0.50	0.49	0.56	0.56	**0.60**	**0.57**	0.53	0.52
SST-5	0.30	0.28	0.34	0.21	**0.41**	**0.32**	0.30	0.29	0.30	n/a	0.37	0.26	0.39	**0.32**	0.31	0.29
TREC	-	-	0.72	n/a	-	-	-	-	0.33	n/a	**0.86**	**0.86**	0.54	0.57	-	-
Subj	-	-	0.76	0.76	-	-	-	-	0.42	0.42	**0.89**	**0.89**	0.88	0.88	-	-

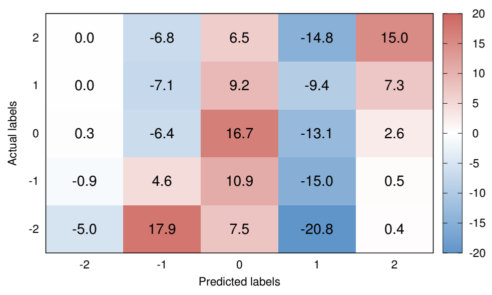


Figure 3.4: Heatmap of difference of the manually labeled RNTN and the CNN model confusion matrices on the SST-5 test set. The numbers in each cell indicate the percentage of normalized differences based on the total number of sentences for each label.

of the applied automatic labeling of phrases used on the RNTN. Therefore, a future research direction could try to reduce this space and find a better initialization.

- **Effect of automatic labeling on RNTN performance:** Table 3.2 also presents the performance of the manually labeled RNTN on the SST-5 dataset. As we can see, automatic labeling results in significant degradation of performance on SST-5. Comparing the results with the CNN model shows that the manually labeled RNTN outperforms the CNN architecture in terms of overall accuracy and F-measure. To have a closer look into the confusion matrix of both methods, we generate a heatmap similar to Figure 3.3, this time subtracting the CNN confusion matrix elements from that of the RNTN method (i.e. $\frac{conf_{rntn}^{i,j} - conf_{cnn}^{i,j}}{total_i}$). Blue color indicates more prediction of sentences by the CNN model than by the RNTN, while the red color indicates the reverse case. Figure 3.4 indicates that the RNTN has a tendency to classify more instances into neutral (0) and positive (2) labels and it is better at correct prediction of somewhat negative (-1), neutral and positive labels, while the CNN is better at classifying negative (-2) and somewhat positive (1) labels. Here, the distribution of sentences over labels is closer to the uniform distribution: 12.6%, 28.6%, 17.6%, 23.1% and 18.1% for -2 to $+2$ labels. Unfortunately, currently, no other dataset is manually labeled at the phrase level. A future direction includes further evaluation of the impact of the phrase labeling accuracy on various datasets.

3.3.3 Task 2: sentence categorization

We test this task on two datasets: (1) the TREC question dataset[6], where the goal is to classify a question into six coarse-grained question types (whether

[6]`http://cogcomp.cs.illinois.edu/Data/QA/QC/`

a question is about an entity, a person, a location, numeric information, abstract concepts or an abbreviation); (2) the Subj dataset[7], where the goal is to classify a sentence as being objective or subjective. The TREC dataset has 5452 training instances and 500 test sentences. The Subj dataset contains $10,000$ sentences in total, but it does not have a separate test set. Therefore we use 10-fold cross-validation. The results are reported in the bottom section of Table 3.2. In these experiments only CNN-based methods are applicable. We observe that the CNN model outperforms RNTN versions, and dependency parsing drastically reduces the performance of the RNTN.

3.3.4 Comparison of CNN architectures

In the next experiment, we compare our proposed deep CNN architecture to a one layer CNN to find out the cases where the deep structure is beneficial. The one layer CNN architecture [94] has several parallel filters of different sizes and a max-pooling layer. In our experiments, we have used 100 filters of size 3, 4, and 5. Classification results (see next to the last column of Table 3.2) indicate that the performance of the one layer architecture is comparable to the proposed deep architecture on the MR dataset and that it performs better on the rest of sentiment datasets. The performance of Kim's architecture on the SST-5 dataset is comparable to the RNTN based on manual labeling. These results highlight the importance of keyphrase recognition in sentiment tasks, where applying larger filters is more beneficial than having several layers of small filters. However, on the other sentence categorization datasets, i.e., TREC and Subj, the proposed deep CNN outperforms the flat architecture.

3.4 Conclusion

In this chapter, we studied two well-known deep architectures, CNNs and RNTNs, in the context of sentence modeling. To avoid the labor-intensive task of manually labeling the internal phrases for recursive networks, we proposed two methods to automatically label them for training and tuning phases: a rule-based method, which is specifically used for sentiment prediction and a CNN based method for general purposes. Considering this part of the study, the evaluation results on the SST-5 dataset indicate that the CNN method tends to assign a positive or negative polarity to the phrases, while the rule-based method classifies many of them as neutral.

Based on the presented two methods for automatic labeling of internal nodes, we conducted a novel in-depth study of the RNTN model and compared the model to a relatively simple deep CNN architecture. Experimental results conducted on an extensive set of standard benchmark datasets

[7]`https://www.cs.cornell.edu/people/pabo/movie-review-data/`

demonstrate that the proposed CNN model outperforms the RNTN variants with automatic phrase labeling, whereas the RNTN with manual labeling (if available) outperforms the CNN. However, in that case, a one layer CNN with several filters of different sizes is comparable to the manually labeled RNTN. These results demonstrate that the syntactic structure of a sentence will help in the classification performance when it is possible to label the internal nodes of a parse tree accurately, otherwise, CNN is more successful at representing the meaning of the sentence with respect to the task. The findings show that there is still room for improvement of RNTN variants in terms of determining tensor functions in a more informed manner.

Despite the progress of deep covolutional neural networks in several fields, its extension for solving multi-label classification problems is still a direction to explore, especially in the streaming scenario where the models need to be simple and anytime. Section 2.3.4 reviewed recent advances of CNNs in multi-label streams. One possible future work is to extend our proposed CNN architecture to the multi-label setting. However, for the rest of this thesis, we focus on proposing a faster algorithm, thus, we develop an analytical approach.

Chapter 4

Multi-label Stream Classification

Many modern applications deal with multi-label data such as functional categorizations of genes, image labeling, and text categorization. Classification of such data with a large number of labels and latent dependencies among them is a challenging task, and it becomes even more challenging when the data is received online and in chunks. Multi-label classification can be viewed as a generalization of multi-class classification where labels do not exclude each other and may have unknown dependencies among each other as well as with the features. One goal of multi-label classification is thus to take advantage of hidden label correlations to improve classification performance. Besides multiple interdependent class labels, we are facing a massive increase in the size of data becoming available. In many cases, this data is received as a stream, e.g., a stream of sensor data or an email text stream. Because of the evolving nature of data streams, we cannot record all the instances and cannot assume previous data can be scanned an arbitrary number of times. This makes multi-label classification on data streams even more challenging. Most of the current multi-label classification methods require a lot of time and memory, which make them infeasible for practical real-world applications. Hence, there is an increasing need for efficient multi-label methods in terms of time and space complexity.

As mentioned in Chapter 2, a common approach to multi-label classification is to transform the problem into one or more single-label problems. The important advantage of problem transformation methods is the possibility of using any available single-label base classifier. This makes these approaches more flexible and generally applicable. However, they may suffer from high computational complexity (as in the Label Powerset method [182]) or ignore the label dependencies (as in the Binary Relevance method [177]); both are important features in the classification of multi-label streams. One successful type of transformation methods was

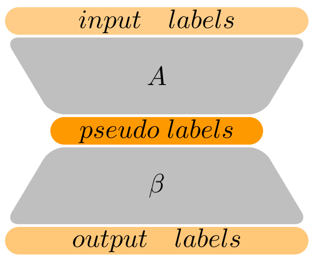


Figure 4.1: General framework of the RACE approach.

introduced that compresses the original label space to a compressed representation. These methods accelerate the learning process by training fewer binary classifiers on compressed label sets, which makes them suitable for multi-label stream classification. However, all the existing methods need to access the whole training data at once.

Although multi-label classification has received much attention, classifying multi-label data streams is relatively new and not very well investigated. Only some well-known methods like Binary Relevance (BR) and Classifier Chains (CC) and a variation of them [141, 168] have been upgraded to the online scenario by using an updateable classifier as their base learner. A multi-label version of Hoeffding Trees, a popular decision tree classifier in single-label stream mining, was also presented [147]. However, none of the available label space reduction methods in the literature on multi-label classification has been adapted for the streaming scenario yet. Most of them are computationally expensive, which makes them infeasible for streaming data.

In this chapter, we propose a novel online linear label compression approach, called RACE (Random Compression), that transforms the labels into a reduced encoded space and trains classifiers on the obtained pseudo labels. Additionally, it provides an analytical method to update the decoding matrix, which maps the labels into the original space and is used during the test phase. For each new batch of data, we (1) compress the label space into a smaller random space, then (2) update the single-label online classifiers or regressors on this compressed label set (we call them pseudo labels), and in the end (3) update the existing decompressing function analytically with the recent data batch. Figure 4.1 represents the general framework of RACE. The main feature of this approach is to process each data batch once and analytically; hence it is not iterative in finding encoding/decoding functions, which leads to a faster method compared to other offline label space compression methods in the literature. To make our method efficient, we select the encoding matrix randomly. Choosing random features or projections has been an active topic in machine learning. Random features have been reasonably successful in scaling kernel methods when dealing with large data sets [142, 107, 38]. Training neural networks with random weights has also been elaborated in the literature [158, 197]. Studies show that there is significant redundancy in the parametrization of several deep learning methods and many of them even need not be learned at all [43]. Subsequent studies propose using random features to reduce the number of parameters for deep convolutional neural networks without sacrificing predictive performance [202]. Another study shows that features from a one-layer convolutional pooling architecture with completely random filters could achieve an average recognition rate that is just slightly worse than unsupervised pre-trained and fine-tuned filters [88]. Further investigations show that a surprising fraction of the performance can be attributed to the architecture alone [155]. In addition to the fast random encoding function, our proposed method does not have many control parameters to be set; the only parameter is the dimension of the compressed space.

4.1 Online label compression method

The key idea of our method is to compress labels into a smaller random space. As mentioned earlier in this chapter, it has been shown that using random features or weights has been a successful component of many existing methods. Using a fixed random encoder will help in accelerating the method and make it scalable for streaming data. However, it may be too restrictive for multi-label stream classification. Therefore, we propose two variants of the compression function: one fixed and one adaptive encoder. After receiving the first batch of data, we first map the label space to the reduced space. As the encoding is a real-valued mapping, the reduced labels will be real-valued. One may then use an updateable regressor or binarize the labels and train one updateable classifier for each pseudo label. The decoding function is calculated as a least squares solution and can be updated incrementally. The next batch of data is first used as test data: We obtain the prediction by the classifiers/regressors and use the decoding function to obtain predicted labels in the original space. After receiving the actual labels of the batch, the batch is used as training data to update the models and the decoding function accordingly. The remainder of this section will explain the algorithm in detail.

Recall from Section 2.1 that the target of multi-label classification is to minimize the risk function as in equation (2.1). In this chapter, we propose a multi-label stream classifier $\mathcal{C} : \mathcal{X} \rightarrow \{0,1\}^l$ that minimizes a label-wise decomposable loss function. To do so, we decompress the reduced label set by a decoding matrix (β), which minimizes the least squares error in each batch:

$$\underset{\beta_t}{\operatorname{argmin}} \sum_{i=1}^{n} \mathbf{e}_t^2(i), \tag{4.1}$$

where $\mathbf{e}_t(i)$ is the error of predicted labels for the i^{th} instance of batch t based on the decoding matrix β_t. Linear models obtained by least squares, equation (4.1), are the same as those obtained by optimizing the Hamming loss, equation (4.2), and can be interchangeably used [42]:

$$f(\boldsymbol{l}, \mathcal{C}(\boldsymbol{x})) = \frac{1}{l}\sum_{k=1}^{l} [\![l_k \neq \mathcal{C}_k(\boldsymbol{x})]\!]. \tag{4.2}$$

4.1.1 Label compression with least squares solution

Compressing the label space to a fixed random space was inspired by the idea of Extreme Learning Machines (ELMs) [81, 112]. Let $q = t.n$ be the total number of instances up to time t and $L(q)$ denote the original label set, which is encoded into a smaller random space by $H_q = L(q)A$, where A is an $l \times k$ fixed encoder and H_q is the $q \times k$ resulting pseudo label matrix and k is the reduced label space size. To decode the reduced label predictions to the original label space, we have:

$$H_q\beta_q = Y(q), \tag{4.3}$$

where β_q is a $k \times l$ decoding matrix after observing q instances. In order to find β_q in equation (4.3), we use the least squares approach to make it faster than iterative optimization methods (such as stochastic gradient descent). Let $E(q) = [\mathbf{e}(1)\mathbf{e}(2)\dots\mathbf{e}(q)]^T = L(q) - Y(q)$ be the error of predicted labels for all instances up to time t according to the parameters at time t. By rewriting equation (4.1), we obtain:

$$\begin{aligned}\xi &= E^T(q)E(q)\\ &= (L^T(q) - Y^T(q))(L(q) - Y(q))\\ &= (L^T(q) - \beta_q^T H_q^T)(L(q) - H_q\beta_q)\\ &= L^T(q)L(q) - 2(H_q^T L(q))^T\beta_q + \beta_q^T H_q^T H_q\beta_q.\end{aligned}$$

Setting the gradient of ξ with respect to β_q equal to zero, we obtain:

$$\begin{aligned}(H_q^T H_q)\hat{\beta}_q &= H_q^T L(q),\\ \hat{\beta}_q &= (H_q^T H_q)^{-1} H_q^T L(q). \end{aligned}\tag{4.4}$$

Consequently, the resulting decoder matrix ($\hat{\beta}_q$) is the least squares solution of equation (4.3). Experimental results show that this approach has better generalization performance at higher learning speed on both classification and regression problems [81, 48].

4.1.2 Random compression of multi-label streams

The least squares solution provided by equation (4.4) is of little interest in mining a stream of data, as it requires all the past samples at each iteration.

We need to transform the solution in equation (4.4) into an incremental form. Hence, for the first batch of data, we calculate the decoding matrix as:

$$\beta_0 = (H_0^T H_0)^{-1} H_0^T L(0). \tag{4.5}$$

Suppose we are at step $t+1$ and receive a new batch of data. The new decoding matrix will be obtained by:

$$\begin{aligned}\beta_{t+1} &= \left(\begin{bmatrix} H_t \\ H_{t+1} \end{bmatrix}^T \begin{bmatrix} H_t \\ H_{t+1} \end{bmatrix} \right)^{-1} \begin{bmatrix} H_t \\ H_{t+1} \end{bmatrix}^T \begin{bmatrix} L(t) \\ L(t+1) \end{bmatrix} \\ &= \left(H_t^T H_t + H_{t+1}^T H_{t+1}\right)^{-1} \left(H_t^T L(t) + H_{t+1}^T L(t+1)\right).\end{aligned} \tag{4.6}$$

To simplify equation (4.6), we estimate $L(t)$ with $H_t \beta_t$, where β_t is the decoding matrix at time t. The second part of the right-hand side of equation (4.6) becomes:

$$\begin{aligned}\left(H_t^T L(t) + H_{t+1}^T L(t+1)\right) &= H_t^T H_t \beta_t + H_{t+1}^T L(t+1) \\ &= (H_t^T H_t + H_{t+1}^T H_{t+1} - H_{t+1}^T H_{t+1})\beta_t + H_{t+1}^T L(t+1) \\ &= (H_t^T H_t + H_{t+1}^T H_{t+1})\beta_t - H_{t+1}^T H_{t+1}\beta_t + H_{t+1}^T L(t+1).\end{aligned} \tag{4.7}$$

Substituting equation (4.7) in (4.6), β_{t+1} can be written as:

$$\begin{aligned}\beta_{t+1} &= \left(H_t^T H_t + H_{t+1}^T H_{t+1}\right)^{-1} \left(H_t^T L(t) + H_{t+1}^T L(t+1)\right) \\ &= \left(H_t^T H_t + H_{t+1}^T H_{t+1}\right)^{-1} \Big(\left(H_t^T H_t + H_{t+1}^T H_{t+1}\right) \beta_t - H_{t+1}^T H_{t+1} \beta_t + H_{t+1}^T L(t+1) \Big) \\ &= \left(H_t^T H_t + H_{t+1}^T H_{t+1}\right)^{-1} \Big(\left(H_t^T H_t + H_{t+1}^T H_{t+1}\right) \beta_t + H_{t+1}^T (L(t+1) - H_{t+1}\beta_t) \Big) \\ &= \beta_t + \left(H_t^T H_t + H_{t+1}^T H_{t+1}\right)^{-1} H_{t+1}^T \left(L(t+1) - H_{t+1}\beta_t\right).\end{aligned} \tag{4.8}$$

In order to avoid multiple calculations of matrix inversions in equation (4.8), we use the Sherman-Morrison-Woodbury formula, where rather than keeping $M_t = H_t^T H_t$, we keep $K_t = M_t^{-1}$. We can rewrite β_{t+1} as the following:

$$\begin{aligned}M_{t+1}^{-1} &= (M_t + H_{t+1}^T H_{t+1})^{-1} \\ &= M_t^{-1} - M_t^{-1} H_{t+1}^T (I + H_{t+1} M_t^{-1} H_{t+1}^T)^{-1} H_{t+1} M_t^{-1} \\ K_{t+1} &= K_t - K_t H_{t+1}^T (I + H_{t+1} K_t H_{t+1}^T)^{-1} H_{t+1} K_t \\ \beta_{t+1} &= \beta_t + K_{t+1} H_{t+1}^T \left(L(t+1) - H_{t+1}\beta_t\right).\end{aligned} \tag{4.9}$$

The recursive method for updating the least squares solution of β_{t+1} is similar to the Recursive Least Squares (RLS) algorithm [48]. Algorithm 3 indicates the main steps of our proposed online label compression (RACE) method. The first step is to generate the encoding matrix (line 1). We generate k uniformly distributed random hyperplanes in the space of labels, where k is the dimension of the reduced label space. In order to have more informative pseudo labels, we use the Gram-Schmidt algorithm [175] to rotate the k hyperplanes orthogonally. The Gram-Schmidt algorithm (line 2–6) subtracts from vector $\mathbf{a_i}$ its components along previously determined orthonormal directions $\mathbf{a_1}, ..., \mathbf{a_{(i-1)}}$ to obtain the new orthogonal direction $\mathbf{v} = \mathbf{a_i} - \sum_{j=1}^{i-1} (\mathbf{a_j} \times \mathbf{a_i})\mathbf{a_j}$. Then, scale $\mathbf{v}$ to obtain a unit norm vector: $\mathbf{a}_i = \mathbf{v}/||\mathbf{v}||$. These orthonormal hyperplanes are kept as the encoding matrix or as an initialization of the adaptive variant.

For the first batch of instances, the RACE algorithm finds the pseudo labels by multiplying the label matrix with the encoding matrix (line 7), and if the binary relevance models are chosen to be classifiers, the resulting real-valued pseudo labels are converted to binary values (line 8– 9). It then trains a binary relevance model, either a regressor or a classifier, on the encoded data (line 10). Finally, the decoding matrix β is initialized based on equation (4.5). Lines 12– 21 repeat this procedure for the following batches of data. Before updating the models and the decoding matrix, we use the current model and β to predict the labels of the newly arrived batch: The model predicts the labels in the reduced space (line 13), and the decoding matrix of the previous step (β_t) transfers pseudo labels to the original space (line 14). To update the model with the new batch, we first update the encoding matrix to the transpose of the decoding matrix from the previous step for the adaptive encoding variants (line 15– 16). The pseudo labels of the new batch are extracted from its true labels (line 17– 19) and used to update the learning model (line 20). In the end, β is incrementally updated by equation (4.9).

4.2 Experimental results

Experiments[1] were performed in a prequential manner, where each data batch is first treated as test data and then as training data. All methods were developed within the Mulan framework [181], the experiments were repeated ten times to reduce the effect of random parameters (e.g., A in RACE or the number of chains in OECC), and the average values are reported.

4.2.1 Experimental setting

Benchmark datasets. We evaluate our proposed method on several multi-label dataset benchmarks (Table 4.1). We have chosen these datasets in order to cover different types of multi-label datasets: datasets with high la-

[1]The RACE source code is available at `https://github.com/kramerlab/RACE`

ALGORITHM 3: RACE algorithm

```
Input: $B(t) = (X(t), L(t))$ is the $t^{th}$ batch of data,
        $l$ and $k$ are the size of original and reduced label space respectively
Output: encoder matrix $A_{l \times k}$, decoder matrix $\beta_{k \times l}$, and trained model
1  Generate a random matrix $A_0 = (\mathbf{a_1}, \mathbf{a_2}, ..., \mathbf{a_k})$ of size $l \times k$, where $\mathbf{a_i}$ is a
     vector of length $l$
   // Gram-Schmidt orthogonalization
2  for $i = 1$ to $k$ do
3      $\mathbf{v} = \mathbf{a_i}$
4      for $j = 1$ to $i - 1$ do
5          $\mathbf{v} = \mathbf{v} - (\mathbf{a_j} \cdot \mathbf{v}) \cdot \mathbf{a_j}$
6      $\mathbf{a_i} = \mathbf{v}/||\mathbf{v}||$
   // Initialization step
7  $H_0 = L(0)A_0$
8  if $Method = classification$ then
9      $H_0^{i,j} = \begin{cases} 1 & \text{if } H_0^{i,j} \geq 0 \\ 0 & \text{otherwise} \end{cases}$
10 Train a Binary Relevance updateable model on $(X(0), H_0)$
11 Initialize label decoders:
   $K_0 = ({H_0}^T H_0)^{-1}$ , $\beta_0 = K_0 H_0^T L(0)$
   // Sequential step
12 while more batches of data do
       // test new batch
13     get $P_{t+1}$ the prediction of model on $X(t+1)$
14     $Y(t+1) = P_{t+1}\beta_t = \begin{cases} 1 & \text{if } Y^{i,j}(t+1) \geq 0 \\ 0 & \text{otherwise} \end{cases}$
       // Update with new batch
15     if $Encoding = adaptive$ then
16         $A_{t+1} = {\beta_t}^T$
17     Get pseudo labels by $H_{t+1} = L(t+1)A_{t+1}$
18     if $Method = classification$ then
19         $H_{t+1}^{i,j} = \begin{cases} 1 & \text{if } H_{t+1}^{i,j} \geq 0 \\ 0 & \text{otherwise} \end{cases}$
20     Update Binary Relevance model with batch $(B(t+1), H_{t+1})$
21     Update $K_{t+1}$ and $\beta_{t+1}$ using equation (4.9)
22     $t := t + 1$
```

bel density (e.g., CAL500), datasets with a lot of labels (e.g., delicious), datasets with a large feature space (e.g., rcv1v2), and datasets with a large sample size (e.g., mediamill and NUS-WIDE). We report the measurements of Example-based accuracy, the Example-based F-measure, Hamming loss, the Micro-averaged and Macro-averaged F-measure, and the running time for all methods.

Table 4.1: Multi-label benchmark datasets used in experiments. |D|, |X|, |L|, LC, LD and UL indicate number of instances, number of features, number of labels, label cardinality, label density, and unique label sets respectively.

Name	|D|	|X|	|L|	LC	LD	UL
CAL500	502	68	174	26.044	0.150	502
delicious	16105	500	983	19.020	0.019	15806
enron	1702	1001	53	3.378	0.064	753
mediamill	43907	120	101	4.376	0.043	6555
NUS-WIDE	269648	500	81	1.869	0.023	18430
rcv1v2(subset1)	6000	47236	101	2.880	0.029	1028

Baseline and comparison methods. We evaluate four variants of RACE, where the encoding may be fixed or adaptive, and the learning method may be regression or classification. We have developed an online version of Binary Relevance (OBR) and Ensemble of Classifier Chains (OECC), two well-known multi-label methods, to compare with RACE. ECCs are an extension of CCs that reduce the chance of poorly ordered chains and create more scalable classifiers for batch learning [149]. Our experimental results in the online setting are in line with previous findings in the batch setting so that only OECC results are reported in our experiments. As many multi-label datasets have a very sparse label space, we implemented an always negative classifier (Negative) to see how well the classifier learns the available labels. Besides, we implemented the majority prediction (Majority) baseline method that takes the cardinality of the current data batch as a threshold for the classification of the following batch. Hence, if the cardinality of the current batch is c, the top $\lfloor c \rfloor$ labels are predicted as positive, and the rest is predicted as negative.

Parameter Settings. Naïve Bayes Updateable is used as a simple generative updateable base classifier for OBR, OECC, and RACE (classification variants). For PLST and regression variants of RACE, we used stochastic gradient descent with the squared loss function and a learning rate of 10^{-4}. The size of the ensemble in OECC is set to 5, and the size of the reduced label space in RACE to $k = \lceil \log_2 l \rceil$, where l is the size of the original label space. The window size is set to 50 instances for CAL500, 100 for enron, and 500 for the rest.

4.2.2 Comparison of RACE variants

We first compare the four variants of RACE, where the encoding matrix can be fixed or adaptive, and the learning method is either regression or classification. The results are reported in Table 4.2. The worst performance belongs to the classification variant with fixed encoding. This can be due to the very confined structure of the model. On the other hand, the regression variant

Table 4.2: Comparison of RACE variants with respect to different measures per dataset. Each cell indicates the mean and standard deviation of all runs including $(rank)$.

Dataset	Method	Encoding	Ex.-based Accuracy	Ex.-based F-measure	Hamming loss	Macro-avg. F-measure	Micro-avg. F-measure	**Average rank**	Running time (s)
CAL500	Classification	Fixed	**0.22 ± 0.01** (1)	**0.35 ± 0.02** (1)	0.15 ± 0.01 (2)	**0.21 ± 0.01** (1)	**0.35 ± 0.02** (1)	1.2	2.69
		Adaptive	0.21 ± 0.02 (2)	0.33 ± 0.02 (2)	0.24 ± 0.04 (4)	0.19 ± 0.02 (2)	0.33 ± 0.03 (2)	2.4	2.57
	Regression	Fixed	0.20 ± 0.01 (3)	0.33 ± 0.01 (2)	0.15 ± 0.00 (2)	0.17 ± 0.01 (3)	0.32 ± 0.01 (3)	2.6	4.29
		Adaptive	0.19 ± 0.01 (4)	0.31 ± 0.01 (4)	**0.14 ± 0.00** (1)	0.16 ± 0.01 (4)	0.31 ± 0.01 (4)	3.4	4.20
delicious	Classification	Fixed	**0.10 ± 0.04** (1)	**0.17 ± 0.06** (1)	0.03 ± 0.00 (3)	0.07 ± 0.02 (3)	**0.17 ± 0.06** (1)	1.8	278.54
		Adaptive	**0.10 ± 0.01** (1)	0.16 ± 0.02 (2)	0.05 ± 0.00 (4)	0.05 ± 0.01 (4)	0.11 ± 0.02 (2)	2.6	278.17
	Regression	Fixed	0.01 ± 0.00 (3)	0.02 ± 0.01 (4)	**0.02 ± 0.00** (1)	**0.08 ± 0.00** (1)	0.02 ± 0.01 (3)	2.4	2343.54
		Adaptive	0.01 ± 0.00 (3)	0.03 ± 0.01 (3)	**0.02 ± 0.00** (1)	**0.08 ± 0.00** (1)	0.02 ± 0.00 (3)	2.2	2815.23
enron	Classification	Fixed	0.14 ± 0.07 (4)	0.21 ± 0.09(4)	0.15 ± 0.03 (4)	0.19 ± 0.08 (4)	0.18 ± 0.09 (4)	4	21.56
		Adaptive	**0.26 ± 0.03** (1)	0.35 ± 0.03 (2)	0.09 ± 0.00 (3)	0.21 ± 0.03 (3)	0.35 ± 0.04 (2)	2.2	20.39
	Regression	Fixed	**0.26 ± 0.02** (1)	**0.36 ± 0.03** (1)	**0.06 ± 0.00** (1)	**0.33 ± 0.01** (1)	**0.38 ± 0.03** (1)	1	96.58
		Adaptive	0.23 ± 0.04 (3)	0.32 ± 0.06 (3)	**0.06 ± 0.00** (1)	0.32 ± 0.01 (2)	0.34 ± 0.06 (3)	2.4	86.65
mediamill	Classification	Fixed	0.22 ± 0.03 (4)	0.33 ± 0.03 (4)	0.11 ± 0.02 (3)	0.34 ± 0.05 (3)	0.30 ± 0.04 (3)	3.4	401.29
		Adaptive	0.25 ± 0.01 (3)	0.34 ± 0.02 (3)	0.16 ± 0.00 (4)	0.32 ± 0.00 (4)	0.22 ± 0.02 (4)	3.6	298.44
	Regression	Fixed	**0.34 ± 0.01** (1)	**0.45 ± 0.01** (1)	**0.03 ± 0.00** (1)	**0.45 ± 0.00** (1)	**0.45 ± 0.01** (1)	1	504.65
		Adaptive	**0.34 ± 0.00** (1)	**0.45 ± 0.00** (1)	0.04 ± 0.00 (2)	0.44 ± 0.00 (2)	**0.45 ± 0.00** (1)	1.4	531.56
NUS-WIDE	Classification	Fixed	0.17 ± 0.02 (4)	0.21 ± 0.02 (4)	0.08 ± 0.02 (4)	0.13 ± 0.08 (4)	0.17 ± 0.02 (2)	3.6	2157.70
		Adaptive	**0.23 ± 0.01** (1)	**0.26 ± 0.01** (1)	0.03 ± 0.00 (3)	0.40 ± 0.02 (3)	**0.23 ± 0.04** (1)	1.8	2629.84
	Regression	Fixed	**0.23 ± 0.00** (1)	0.23 ± 0.00 (3)	**0.02 ± 0.00** (1)	**0.42 ± 0.00** (1)	0.02 ± 0.01 (4)	2	2699.95
		Adaptive	**0.23 ± 0.00** (1)	0.24 ± 0.00 (2)	**0.02 ± 0.00** (1)	**0.42 ± 0.00** (1)	0.04 ± 0.01 (3)	1.6	2415.93
rcv1v2	Classification	Fixed	**0.10 ± 0.03** (1)	**0.16 ± 0.05** (1)	0.10 ± 0.04 (3)	0.04 ± 0.02 (3)	**0.15 ± 0.04** (1)	1.8	22739.73
		Adaptive	0.08 ± 0.02 (2)	0.12 ± 0.02 (2)	0.18 ± 0.03 (4)	0.04 ± 0.02 (3)	0.07 ± 0.02 (2)	2.6	22723.76
	Regression	Fixed	0.01 ± 0.00 (4)	0.02 ± 0.01 (4)	**0.03 ± 0.00** (1)	**0.19 ± 0.00** (1)	0.03 ± 0.01 (4)	2.8	59603.26
		Adaptive	0.04 ± 0.01 (3)	0.06 ± 0.02 (3)	**0.03 ± 0.00** (1)	**0.19 ± 0.00** (1)	0.06 ± 0.03 (3)	2.2	57300.81
Average rank	Classification	Fixed	2.5	2.5	3.17	3	2	**2.63**	
		Adaptive	1.67	2	3.67	3.17	2.17	**2.54**	
	Regression	Fixed	2	2.5	1.17	1.33	2.67	**1.93**	
		Adaptive	2.67	2.67	1.17	1.83	2.83	**2.23**	

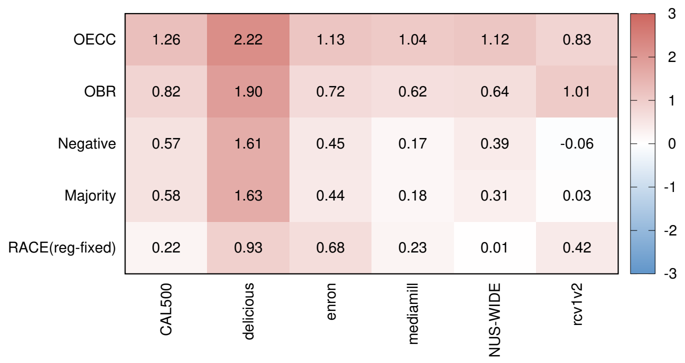


Figure 4.2: Runtime comparison of different algorithms to RACE (cls-adap), results shown as log ratio.

with adaptive encoding does not perform well, possibly due to the overfitting of so many real-valued parameters. The average ranks over all datasets indicate that the classification variant with adaptive encoding exhibits the best performance in terms of Example-based accuracy and Example-based F-measure and nearly the best in terms of Micro-averaged F-measure. Conversely, the regression variant with fixed encoding achieves the best results in terms of Hamming loss and Macro-averaged F-measure. However, its Example-based measures and Micro-averaged F-measure on some datasets (i.e., delicious and rcv1v2) are poor. Concerning running time, the classification variants are faster than their regression counterparts as their base learners are Naive Bayes Updateable, which is faster than the stochastic gradient descent regression model.

4.2.3 Comparison to online baseline methods

In Table 4.3 we compare two variants of RACE, the classification method with adaptive encoding (cls-adap) and the regression method with fixed encoding (reg-fixed), to other online baseline methods and report the average values for different measures over all batches and runs. We left out the standard deviation of different runs in this table as the corresponding values for OECC were negligible, and the ones for the RACE variants were reported in Table 4.2. Again, the average ranks over all datasets indicate that RACE (cls-adap) achieves the best Example-based accuracy and Example-based F-measure and nearly the best Micro-averaged F-measure; and hence, with the average rank of 2.73 over all measures, it stands on the second place, after RACE (reg-fixed). RACE (reg-fixed) achieves the best results in terms of Hamming loss and Macro-averaged F-measure. It gives the best performance across all evaluated measures on enron and mediamill. However, its poor behavior in terms of Example-based measures and the Micro-averaged F-measure on delicious and rcv1v2 is quite similar to the Negative baseline.

The last column of Table 4.3 presents the running time of all algorithms.

Table 4.3: Comparison of RACE to OBR, OECC, and the Negative and Majority baseline methods across different measures per dataset $(rank)$.

Dataset	Method	Ex.-based accuracy	Ex.-based F-measure	Hamming loss	Macro-avg. F-measure	Micro-avg. F-measure	**Average rank**	Running time (s)
CAL500	RACE (cls-adap)	0.21 (2)	0.33 (2)	0.24 (3)	0.19 (3)	0.33 (3)	2.6	2.57
	RACE (reg-fixed)	0.20 (4)	0.33 (2)	**0.15** (1)	0.17 (4)	0.32 (4)	3	4.29
	Majority	0.03 (5)	0.06 (5)	0.28 (5)	0.11 (6)	0.06 (5)	5.2	9.79
	Negative	0.00 (6)	0.00 (6)	**0.15** (1)	0.12 (5)	0.00 (6)	4.8	9.64
	OBR	**0.22** (1)	**0.35** (1)	0.27 (4)	**0.25** (1)	**0.35** (1)	1.6	17.14
	OECC	0.21 (2)	0.33 (2)	0.31 (6)	0.24 (2)	0.34 (2)	2.8	46.84
delicious	RACE (cls-adap)	**0.10** (1)	0.16 (2)	0.05 (4)	0.05 (5)	0.11 (2)	2.8	278.17
	RACE (reg-fixed)	0.01 (4)	0.02 (4)	**0.02** (1)	**0.08** (1)	0.02 (4)	2.8	2343.54
	Majority	0.01 (4)	0.01 (5)	0.04 (3)	**0.08** (1)	0.01 (5)	3.6	11760.25
	Negative	0.00 (6)	0.00 (6)	**0.02** (1)	**0.08** (1)	0.00 (6)	4	11324.38
	OBR	**0.10** (1)	**0.17** (1)	0.16 (5)	0.07 (4)	**0.13** (1)	2.4	22117.97
	OECC	0.04 (3)	0.07 (3)	0.61 (6)	0.04 (6)	0.04 (3)	4.2	46525.20
enron	RACE (cls-adap)	**0.26** (1)	0.35 (2)	0.09 (3)	0.21 (4)	0.35 (2)	2.4	20.39
	RACE (reg-fixed)	**0.26** (1)	**0.36** (1)	**0.06** (1)	**0.33** (1)	**0.38** (1)	1	96.58
	Majority	0.07 (5)	0.11 (5)	0.11 (4)	0.27 (3)	0.12 (5)	4.4	56.52
	Negative	0.00 (6)	0.00 (6)	0.07 (2)	0.29 (2)	0.00 (6)	4.4	57.51
	OBR	0.23 (3)	0.33 (4)	0.18 (5)	0.14 (5)	0.29 (3)	4	108.12
	OECC	0.23 (3)	0.34 (3)	0.18 (5)	0.13 (6)	0.29 (3)	4	275.51
mediamill	RACE (cls-adap)	0.25 (2)	0.34 (2)	0.16 (4)	0.32 (4)	0.22 (2)	2.8	298.44
	RACE (reg-fixed)	**0.34** (1)	**0.45** (1)	**0.03** (1)	**0.45** (1)	**0.45** (1)	1	504.65
	Majority	0.07 (5)	0.13 (5)	0.07 (3)	0.42 (3)	0.13 (5)	4.2	450.23
	Negative	0.04 (6)	0.04 (6)	0.04 (2)	0.43 (2)	0.00 (6)	4.4	444.24
	OBR	0.10 (3)	0.17 (3)	0.30 (5)	0.15 (5)	0.17 (3)	3.8	1258.34
	OECC	0.08 (4)	0.15 (4)	0.35 (6)	0.15 (5)	0.15 (4)	4.6	3295.56
NUS-WIDE	RACE (cls-adap)	**0.23** (1)	**0.26** (1)	0.03 (3)	0.40 (4)	**0.23** (1)	2	2629.84
	RACE (reg-fixed)	**0.23** (1)	0.23 (2)	**0.02** (1)	**0.42** (1)	0.02 (4)	1.8	2699.95
	Majority	0.02 (6)	0.02 (6)	0.04 (4)	0.41 (3)	0.01 (5)	4.8	5338.37
	Negative	0.22 (3)	0.22 (3)	**0.02** (1)	**0.42** (1)	0.00 (6)	2.8	6506.92
	OBR	0.06 (4)	0.10 (5)	0.26 (6)	0.08 (5)	0.11 (2)	4.4	11481.38
	OECC	0.06 (4)	0.11 (4)	0.24 (5)	0.08 (5)	0.11 (2)	4	34511.24
rcv1v2	RACE (cls-adap)	0.08 (3)	0.12 (3)	0.18 (4)	0.04 (6)	0.07 (3)	3.8	22723.76
	RACE (reg-fixed)	0.01 (4)	0.02 (4)	**0.03** (1)	**0.19** (1)	0.03 (4)	2.8	59603.26
	Majority	0.00 (5)	0.00 (5)	0.05 (3)	0.18 (3)	0.00 (5)	4.2	24487.15
	Negative	0.00 (5)	0.00 (5)	**0.03** (1)	**0.19** (1)	0.00 (5)	3.4	19761.63
	OBR	**0.13** (1)	**0.20** (1)	0.43 (5)	0.13 (4)	**0.12** (1)	2.4	234679.01
	OECC	**0.13** (1)	0.19 (2)	0.46 (6)	0.13 (4)	**0.12** (1)	2.8	154585.83
Average rank	RACE (cls-adap)	1.67	2	3.5	4.33	2.17	**2.73**	
	RACE (reg-fixed)	2.5	2.33	1	1.5	3	**2.07**	
	Majority	5	5.17	3.67	3.17	5	**4.4**	
	Negative	5.33	5.33	1.33	2	5.83	**3.96**	
	OBR	2.17	2.5	5	4	1.83	**3.1**	
	OECC	2.83	3	5.67	4.67	2.5	**3.73**	

For a better visualization of time complexity, we plotted a heat map that represents the log ratio of each method's time, i.e. $\log_{10} \frac{t_{method}}{t_{RACE(cls-adap)}}$ (Figure 4.2). The heat map illustrates the difference in the order of the time needed to finish for each method in comparison to RACE (cls-adap) on each dataset. We observe that the space reduction method is efficient and the running time of RACE (cls-adap) is orders of magnitude smaller than the one of other ensemble methods, especially when the original label space is very large (see the results for delicious).

4.2.4 Comparison to offline label compression methods

In this section we compare RACE (cls-adap) to PLST[2] [172], a popular offline label compression method, which uses a projection method based on singular value decomposition. Here, we used hold-out evaluation, i.e., 33% of each dataset was chosen randomly as the test set and the rest as the training set. RACE (cls-adap) received training data in batches, and after updating for each batch, the test was performed on the test data, and the average values of Example-based accuracy and Hamming loss are reported. In addition to present each batch once to RACE, we repeated the experiment by showing every batch several times consecutively. Table 4.4 presents the results for different measures and datasets, and reports the iterative results for 3 iterations. We observe that while RACE has an adaptive nature, Example-based accuracy and Hamming loss are comparable to PLST and on some datasets (CAL500 and delicious) RACE reaches an even higher accuracy. Moreover, on all datasets, due to its random compression nature, RACE has an order of magnitude smaller running times, and when the dataset is large (e.g., rcv1v2 and NUS-WIDE), PLST does not even finish within reasonable time[3]. Looking into iterative RACE, we observe that this method achieves more stable results over time, however, on some datasets it leads to better accuracy (CAL500 and delicious), while on some others the accuracy is reduced (mediamill, NUS-WIDE, and rcv1v2).

4.2.5 Impact of pseudo label set size

RACE has one parameter to set: the size of the reduced label space. We have changed the number of pseudo labels from $\lceil \log_2 l \rceil$ to $\lceil \log_2 l \rceil^2$, where l is the size of the original label space. Figure 4.3 shows the impact of this parameter on the performance of RACE (cls-adap) and RACE (reg-fixed) for the CAL500 and the enron datasets. As we can see in both datasets, while increasing the number of pseudo labels in RACE (cls-adap) does not change Example-based accuracy and Hamming loss notably, the Micro-averaged and Macro-averaged F-measure are improved up to some point (at 30 pseudo labels in enron and at 40 pseudo labels in CAL500), but then they drop again. This is the case for RACE (reg-fixed) for Example-based accuracy and the Micro-averaged F-measure in the enron dataset, however, in the CAL500 dataset, different measures do not change notably for RACE (reg-fixed). Comparing these improvements to the average ranks from Table 4.3, we can see that each of these variants can be improved by choosing a properly fine-tuned reduced label space size.

[2]We have used the implementation provided by the Meka framework at `https://github.com/Waikato/meka/tree/master/src/main/java/meka/classifiers/multilabel`.

[3]The experiment was not finished after 120 hours on the same system.

Table 4.4: Comparison of RACE (cls-adap), its iterative version and PLST on various datasets.

	CAL500			delicious			enron			mediamill			NUS-WIDE			rcv1v2		
	RACE	RACE $(iter = 3)$	PLST	RACE	RACE $(iter = 3)$	PLST	RACE	RACE $(iter = 3)$	PLST	RACE	RACE $(iter = 3)$	PLST	RACE	RACE $(iter = 3)$	PLST	RACE	RACE $(iter = 3)$	PLST
Ex.-based accuracy	0.23 ± 0.01	**0.25 ± 0.01**	0.19	0.10 ± 0.01	**0.12 ± 0.04**	0.03	0.33 ± 0.04	0.33 ± 0.04	**0.34**	0.31 ± 0.02	0.29 ± 0.03	**0.37**	**0.17 ± 0.03**	0.15 ± 0.02	-	0.10 ± 0.02	**0.12 ± 0.02**	-
Hamming loss	0.17 ± 0.01	0.16 ± 0.01	**0.14**	0.04 ± 0.00	0.04 ± 0.01	**0.02**	0.07 ± 0.00	0.08 ± 0.04	**0.06**	0.06 ± 0.01	0.06 ± 0.00	**0.03**	**0.04 ± 0.00**	**0.04 ± 0.00**	-	**0.15 ± 0.04**	0.19 ± 0.09	-
Running time (s)	**1.79**	4.46	2.34	**1.43×10^3**	3.08×10^3	2.28×10^4	**20.20**	50.82	640.07	**1.22×10^3**	3.13×10^3	1.52×10^4	**6.97×10^4**	1.62×10^5	-	**1.84×10^4**	2.28×10^4	-

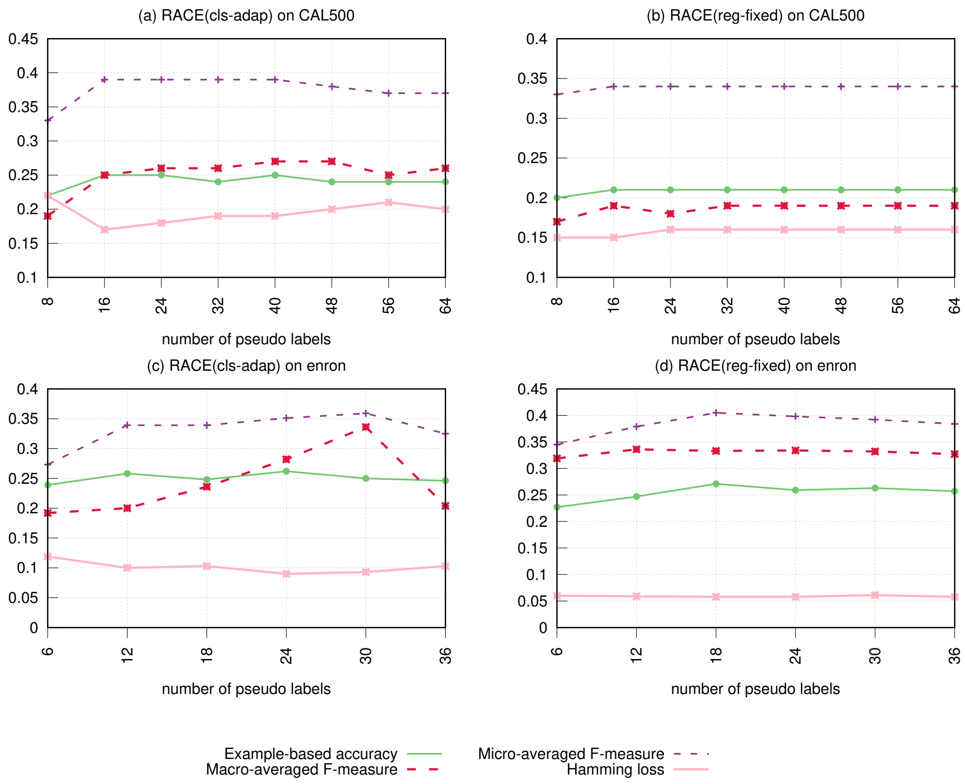


Figure 4.3: Effect of different numbers of pseudo labels on the performance of RACE variants on the CAL500 and enron datasets.

4.3 Conclusion

This chapter addressed the problem of online label compression for multi-label data stream classification. The main contribution of the proposed algorithm is to encode the original label space by a random projection method to a much smaller space and decode the output of models by an incremental analytical approach inspired by Extreme Learning Machines. Different variants of the proposed method, RACE, were tested. The experimental evaluations showed that the approach works well on different datasets across a variety of measures, and outperforms other existing online multi-label baselines in terms of accuracy, F-measure, Hamming loss and notably running time. However, there is still room for further improvement. We can extend the current framework to cover various types of concept drift. An ensemble of RACE models also can be used to reduce the variance due to its random initialization.

Chapter 5

Pool-based Stream Classification

Classifying a stream of non-stationary data with recurrent drift is a challenging task and has been considered as an interesting problem in recent years. Early work on stream classification goes back to the 1990s and the FLORA framework [195]. Since then, there have been extensive studies on learning algorithms that can handle data streams [52, 58, 2, 101, 215, 21]. Most of the work in the literature focuses on handling gradual and sudden drift. That is, they are trying to adapt to a new concept and forget the previous, outdated ones [83]. Such a case cause problems, especially in adversarial systems (e.g., spam filtering and intrusion detection), where the adversary might take advantage of the forgetting mechanism and beats the learner by presenting new instances from a previously seen concept that belongs to a long time ago, which the learner has already forgotten. In some other problems, e.g., weather forecast, remembering the previously seen concepts will help in future predictions. In these cases, the drift can imply the recurrence of previously seen concepts. The recurrent concepts challenge has been faced only quite recently [54, 64, 66, 145, 153]. As reviewed in Chapter 2, all of the existing approaches handling recurrent concepts maintain a pool of concepts/classifiers and use that pool for future classifications to reduce the error on classifying the instances from a recurring concept.

In this chapter, we extend the idea of exploiting recurring concepts using a pool of classifiers, in which each classifier represents a concept. We call the learning framework *Pool and Accuracy based Stream Classification* (*PASC*). The pool is updated iteratively after receiving new batches of data. If the similarity of the recent batch and one of the classifiers of the pool is high enough, the corresponding similar classifier is updated. If not, a new classifier is added to the pool. In the predecessor approach, the CCP framework [92], the pool size is fixed, and after reaching the limit, the most similar classifier is updated regardless of its level of similarity. This can cause the classifier to represent multiple concepts. To solve the problem, we present a merging procedure that finds the most similar concepts in the pool and merge them. The benefit of a merging procedure becomes clear once we notice the small size of data batches and the fact that one concept may appear in several batches of instances. At this stage, the value of the similarity threshold becomes critical, and it may lead to several classifiers in the pool that represent one concept. Once the pool reaches its maximum size, we can search the pool and merge these classifiers to release space for a new classifier. Merging can also improve the generalization performance of a classifier. In case there are not such classifiers in the pool, the most similar classifier to the current batch will be updated. We propose novel methods to update the pool and merge classifiers: exact Bayesian, Bayesian and Heuristic methods. Our proposed framework improves the CCP framework in several ways:

1. CCP uses a pure Euclidean distance as a distance measure. However, in cases where the range of features is not similar, some features impact more than others. PASC solves this issue by using a normalized distance measure.
2. To classify a new batch of instances, PASC uses a weighted majority approach and compares it to the so-called active classifier method in CCP.
3. PASC proposes three new batch assignment methods and a merging process to manage the pool.

5.1 Pool and accuracy based stream classification

As explained above, the proposed PASC framework is a generalization of the CCP framework [92]. The algorithm keeps a pool of classifiers, where each classifier corresponds to a concept. When a batch of instances is received, the algorithm first predicts the corresponding class of the instances. After receiving the true class of new instances, the algorithm either chooses the most appropriate classifier and updates it, or creates a new classifier and adds it to the pool. Usually, there is a finite number of possible concepts in a dataset. Hence, in order to avoid redundancies and an ever-growing pool size, we set a maximum size for the number of classifiers in the pool. Algorithm 4 presents the general steps of the PASC framework.

Let $X(t) = (\boldsymbol{x}_1^t, \boldsymbol{x}_2^t, \ldots, \boldsymbol{x}_n^t)$ and $Y_{(t)} = (y_1^t, y_2^t, \ldots, y_n^t)$ be instances of batch t and their corresponding true class, where $\boldsymbol{x}_i^t$ and y_i^t indicate the i^{th} instance of batch t and its true class respectively. Lines 9-12 are the main steps of Algorithm 4. This loop iteratively classifies a batch and updates the pool. The algorithm has three main phases that are shown in lines 10-12 and will be explained in the following subsections. Lines 1–8 are the initialization parts of the method: classifier $\mathcal{C}$ is built on the first batch of data and

ALGORITHM 4: PASC main framework

```
Input: $B(t) = (X(t), Y(t))$: instances of data batch $t$,
       instance $\boldsymbol{x}_i^t$ to be classified
Output: Predicted class of instance $\boldsymbol{x}_i^t$
1  $Pool = \emptyset$
2  Initialize the $RDC$ classifier
3  Train a classifier $\mathcal{C}$ on $(X(0), Y(0))$
4  $Pool = Pool \cup \{\mathcal{C}\}$
5  $w_0 = 1$
6  $c_{active} = 1$
7  $X_{RDC} = sum\_data(X(0))$
8  Update $RDC$ with $(X_{RDC}, 0)$
9  for $j \geq 1$ do
10     Classify $X(t)$
11     Update $Pool$ with $B(t)$
12     Update the active classifier or classifier weights
```

its weight is set to 1. c_{active} indicates the index of the active classifier for the variants of the framework that use the so-called *active classifier*. *Raw Data Classifier* (*RDC*) is a classifier that aims to predict the concept, from which $X(t)$ is drawn. It makes the prediction only based on the instance features and not the true class of data. Training input of the RDC classifier, X_{RDC}, is a summarization of each data batch and the index of its corresponding concept in the pool.

5.1.1 Classify new data

As explained in the CCP framework [92], we can classify the instances by the current active classifier. However, there may be more than one classifier in the pool, which represents the current concept. Hence, we propose a weighted classification method. Both methods are described in the following:

Active classifier: The *active classifier* is a classifier which was updated by the previous batch of instances. If the data is stationary and there is no concept drift between the last consecutive batches, this classifier is a reasonable choice. The variable pc in line 3 of Algorithm 5 stores the predicted class by the active classifier c_{active}.

Weighted classifiers: We use an adaptive method to choose a combination of classifiers from the pool. At the beginning of the iteration, we initialize the weight of each classifier according to the current state of the pool and the last batch of data. After predicting the class of $\boldsymbol{x}_i^t$, we update the weights of classifiers using y_i^t:

$$w'_j = w_j \beta^{M(j,i)}, \tag{5.1}$$

ALGORITHM 5: Classify new data

```
Input: $X(t)$: instances of data batch $t$ with size $n$
Output: Predicted class of $X(t)$
1 if $mode = active\ classifier$ then
2     for $i$=1 to $n$ do
3         $pc[\boldsymbol{x}_i^t]$ = classify $\boldsymbol{x}_i^t$ with $c_{active}$
  else if $mode = weighted\ classifiers$ then
      // makes a subsample of size $m$
4     $S_t = sub\_sample(X(t), m)$
5     for $i$=1 to $m$ do
          // Uses the highest weighted classifier
6         $pc[\boldsymbol{x}_i^t] = weighted\_classify(Pool, w, S_{t,i})$
7         for $j$=1 to Pool.size do
              // error is a binary function
8             $w_j = w_j \beta^{Pool[j].error(S_{t,i}, y_i^t)}$
```

where w_j and $w'(j)$ are the current and the new weight of the j^{th} classifier, respectively, and $\beta \in [0, 1)$ is a penalty parameter. $M(j, i)$ is 0 if the j^{th} classifier predicts the class correctly and 1 otherwise. Equation (5.1) is inspired from an online prediction modeling of a two-player repeated game problem [50]. The first player, here the pool of classifiers, is the learner. The actions of the first player include choosing a classifier among the pool of classifiers. A mixed strategy P is used by the first player to choose its actions. P is a probability distribution function on each possible action to be selected. The learner computes a mixed strategy P by normalizing the weights, which are assigned to the classifiers. This is equivalent to use a majority vote among the classifiers of the pool according to their weights. The second player, here the source of producing the batches, is the environment. The actions of the second player include choosing the instances, which are given to the learner for classification. The second player uses the mixed strategy Q to choose its actions. The strategies P and Q change themselves as the game proceeds. P is updated according to the loss of the last iteration by updating the classifiers' weights, and Q can be updated by the environment arbitrarily. If the number of instances is sufficiently large and the learner uses equation (5.1) to update the weights of classifiers, the prediction error converges to the best classifier's error on the last batch [50]. Considering this, if the size of the input batch is large enough, the accuracy of our ensemble classifier on the last batch of instances is as good as using the best classifier in the pool. However, in the context of concept drift, the size of the batch should not grow arbitrarily since the i.i.d assumption can quickly be violated.

For the sake of efficiency, our classification algorithm is slightly different from [50]: First, instead of getting a majority vote, we classify the instance

ALGORITHM 6: Update the classifier pool

```
Input: A $Pool$ of classifiers,
       $C_{best}$: Best concept describing batch $B(t) = (X(t), Y(t))$,
       $S_{max}$: The similarity value of $C_{best}$ and $X(t)$,
       $\theta$: Threshold parameter
1 $(S_{max}, C_{best}) = batch\_assignment(Pool, B(t))$
2 if $S_{max} > \theta$ then
3     $Pool[C_{best}].update(B(t))$
  else if $Pool.size < C_{max}$ or $merge\_procedure(Pool, C_{best}, B(t))$ then
4     Train $\mathcal{C}$ on $B(t)$
5     $Pool = Pool \cup \{\mathcal{C}\}$
6     $C_{best} = Pool.size$
  else
7     $Pool[C_{best}].update(B(t))$
  // only in (exact) Bayesian
8 $RDC.update(X(t), C_{best})$
```

with the classifier that has the highest weight. Second, we do not update our classifiers for every instance; instead, we use a subsample of the batch. In our experiments, we chose the square root of the batch size as the number of elements for updating the classifiers. In line 4 of Algorithm 5, we make a subsample of instances and make sure to update the weights only for the members of the subsample S_t (line 6).

5.1.2 Update the classifier pool

Given $X(t)$ and $Y(t)$, our goal is to find the most likely concept for the current batch (line 1 of Algorithm 6). For this purpose, we propose three different batch assignment methods: *Exact Bayes* that uses the Bayes theorem to find the probabilities, the *Bayes* method, which is similar to the Exact Bayes method but makes some simplifying assumptions to decrease the time complexity, and the *Heuristic* method. These methods find the most similar concept/classifier of the pool to the labeled batch (C_{best}) and the similarity between them (S_{max}). If the similarity is greater than a predefined threshold (θ_1, θ_2 or θ_3 for exact Bayesian, Bayesian and Heuristic methods, respectively), the C_{best} classifier is updated with the batch (line 3). Once there is free space in the pool or the $merge\_procedure$ can create a free space, a new classifier is trained on the new batch (line 4– 6). If the pool is full and there are no similar concepts to be merged, the best classifier is updated (line 7). Finally, RDC is updated according to $X(t)$ and C_{best} (line 8). We explain the merge procedure and various batch assignment methods in the following:

Merge procedure Assume the concept n_C refers to the nearest concept to C_{best}. If the distance between n_C and C_{best} is less than the distance between C_{best} and the new batch $(B(t))$, a merging procedure will merge C_{best} and n_C together. Merging of C_{best} and n_C should be possible even in the absence of their training data. In this framework, we use Naïve Bayes, a simple generative classifier, as the base classifier and implement a simple method to merge two Naïve Bayes classifiers by combining the probability density functions (*pdf*) of each attribute. For each nominal attribute f_i and class y_j, a probability $P(f_i = v|y_j)$ is maintained for each possible value v of f_i. The corresponding probability of the merged classifier is the weighted average of the two probabilities with respect to the number of instances. For each numeric attribute f_i and class y_j, a normal distribution $P(f_i|y_j)$ is maintained. The mean value of one of the two normal distributions is used to update the mean value and the standard deviation of the other normal distribution according to its number of instances. The result distribution is then used as the corresponding distribution of the merged classifier. The pdfs maintained for the class distributions are merged similarly. Moreover, in the Bayesian or exact Bayesian batch assignment methods, the RDC classifier must be updated, which will be discussed later.

The distance measure used here is a normalized version of distance measure used in CCP and we name it as $\mathcal{D}_{norm}$. It compares the concept representative vectors of the instance batches or the ones of a classifier, Z and Z', as follows:

$$\mathcal{D}_{norm}(Z, Z') = \sqrt{\mathcal{D}(z_1, z'_1) + \cdots + \mathcal{D}(z_m, z'_m)} \tag{5.2}$$

If f_i is numeric, $\mathcal{D}(z_i, z'_i)$ is defined as:

$$\mathcal{D}(z_i, z'_i) = \sum_{j=1}^{k} \min\big((\frac{\mu_{ij} - \mu'_{ij}}{\sigma_{ij} + \sigma'_{ij}})^2, 1\big), \tag{5.3}$$

where k is the number of classes, and μ_{ij} and σ_{ij} indicate the mean and standard deviation of the i^{th} feature for the instances of class j. If f_i is nominal, $\mathcal{D}(z_i, z'_i)$ can be written as:

$$\mathcal{D}(z_i, z'_i) = \sum_{j=1}^{size(z_i)} \mathcal{D}(z_{ij}, z'_{ij})^2, \tag{5.4}$$

where $z_{ij}(X)$ is the j^{th} element of $z_i(X)$ and $\mathcal{D}(z_{ij}, z'_{ij})$ is simply the difference of the nominal values. In the following, we explain the batch assignment methods in detail:

Exact Bayesian batch assignment method: One way to find the most similar classifier of the pool to a batch of data, $B(t) = (X(t), Y(t))$, is to compute the likelihood of the batch and the concept described by the classifier

$\mathcal{C}_i$ from Bayes rule:

$$P(\mathcal{C}_i|B(t)) = \frac{P(B(t)|\mathcal{C}_i)P(\mathcal{C}_i)}{P(B(t))}. \tag{5.5}$$

To derive the best concept using equation (5.5) we should find:

$$\arg\max_i P(\mathcal{C}_i|B(t)) = \arg\max_i P(B(t)|\mathcal{C}_i)P(\mathcal{C}_i), \tag{5.6}$$

where the best $\mathcal{C}_i$ is assumed to be independent of the occurence probability of $B(t)$. In fact, $P(\mathcal{C}_i)$ can depend on the previous concepts and the underlying distribution of data. The former can vary in the context of concept drift and cannot be modeled appropriately without making any specific assumptions. Moreover, considering the first kind of dependencies in calculating $P(\mathcal{C}_i)$ will lead to a late detection of concept drifts, since the concepts which appeared in the previous batches can get higher probabilities. The second kind of dependencies (the underlying distribution of data) is unknown. If for some datasets, we have domain knowledge related to these dependencies, we can include them in equation (5.6). Thus, we assume the term $P(\mathcal{C}_i)$ to be identical for all concepts and discard it in our calculations:

$$\arg\max_i P(\mathcal{C}_i|B(t)) = \arg\max_i P(B(t)|\mathcal{C}_i). \tag{5.7}$$

We can expand equation (5.7) with respect to the elements of $B(t)$:

$$\begin{aligned} &= \arg\max_i P((\boldsymbol{x}_1^t, y_1^t), (\boldsymbol{x}_2^t, y_2^t), \ldots, (\boldsymbol{x}_n^t, y_n^t)|\mathcal{C}_i) \\ &= \arg\max_i P((\boldsymbol{x}_1^t, y_1^t)|\mathcal{C}_i)P((\boldsymbol{x}_2^t, y_2^t)|\mathcal{C}_i, (\boldsymbol{x}_1^t, y_1^t)) \ldots \\ &\qquad P((\boldsymbol{x}_n^t, y_n^t)|\mathcal{C}_i, (\boldsymbol{x}_1^t, y_1^t), \ldots, (\boldsymbol{x}_{n-1}^t, y_{n-1}^t)). \end{aligned} \tag{5.8}$$

Suppose we define $\mathcal{C}_{i,j}$ as the hypothesis $\mathcal{C}_i$, which is updated by the first j labeled instances of the batch:

$$\mathcal{C}_{i,j} = \mathcal{C}_i, (\boldsymbol{x}_1^t, y_1^t), \ldots, (\boldsymbol{x}_j^t, y_j^t)$$

$$P(\boldsymbol{x}_{j+1}^t, y_{j+1}^t|\mathcal{C}_{i,j}), 0 \leq j \leq n-1 \tag{5.9}$$

Now the task is transformed to the estimation of likelihoods of the form:

$$P(\boldsymbol{x}_{j+1}^t, y_{j+1}^t|\mathcal{C}_{i,j}) = P(y_{j+1}^t|\mathcal{C}_{i,j}, \boldsymbol{x}_{j+1}^t)P(\boldsymbol{x}_{j+1}^t|\mathcal{C}_{i,j}), 1 \leq j \leq n. \tag{5.10}$$

The term $P(y_{j+1}^t|\mathcal{C}_{i,j}, \boldsymbol{x}_{j+1}^t)$ can be estimated by classifier $\mathcal{C}_i$. Note that this term equals to the probability that the label of $\boldsymbol{x}_{j+1}^t$ is predicted as y_{j+1}^t given that true concept $\mathcal{C}_i$ and the j labeled instances of equation (5.11) are visited. Hence, it is sufficient to update the i^{th} classifier with the j labeled instances mentioned above and then output the posterior probability for the instance

$\boldsymbol{x}_{j+1}^t$. To estimate the second term of the right hand side of equation (5.10), we assume:

$$P(\boldsymbol{x}_{j+1}^t|\mathcal{C}_{i,j}) = P(\boldsymbol{x}_{j+1}^t|\mathcal{C}_i, \boldsymbol{x}_1^t, \dots, \boldsymbol{x}_j^t), 0 \le j \le n-1. \tag{5.11}$$

This assumption can be valid as we already know that j labeled instances have been drawn from $\mathcal{C}_i$, and therefore, we can use it to estimate instances independent of their labels. The probabilities can be estimated as follows: We initialize a classifier for this purpose namely raw data classifier (RDC). It gets one instance as an input and predicts its corresponding classifier in the pool. We train RDC after receiving the true labels of each batch $X(t)$ and determining the concept of the labeled instances of this batch. The instances, and not their class values, and their corresponding concept ID are fed to the RDC to update itself. Since the prior probability of the concept, $P(\mathcal{C}_i)$, is assumed to be identical for all concepts, to calculate $P(\boldsymbol{x}_{j+1}^t|\mathcal{C}_i)$, we can compute only the posterior probability of RDC on $\boldsymbol{x}_{j+1}^t$, $P(\mathcal{C}_i|\boldsymbol{x}_{j+1}^t)$. To estimate $P(\boldsymbol{x}_{j+1}^t|\mathcal{C}_i, \boldsymbol{x}_1^t, \dots, \boldsymbol{x}_j^t)$, it is more convenient to update RDC with j instances $\boldsymbol{x}_1^t, \dots, \boldsymbol{x}_j^t$ and the concept ID i as the class label and then output the posterior probability. Finally, the most relevant concept to the newly arrived batch, $B(t)$, can be determined:

$$\begin{aligned}\arg\max_i P(y_1^t|\mathcal{C}_i, \boldsymbol{x}_1^t)P(y_2^t|\mathcal{C}_{i,1}, \boldsymbol{x}_2^t)\dots P(y_n^t|\mathcal{C}_{i,n-1}, \boldsymbol{x}_n^t)\\ P(\boldsymbol{x}_1^t|\mathcal{C}_i)P(\boldsymbol{x}_2^t|\mathcal{C}_i, \boldsymbol{x}_1^t)\dots P(\boldsymbol{x}_n^t|\mathcal{C}_i, \boldsymbol{x}_1^t, \dots, \boldsymbol{x}_{n-1}^t).\end{aligned} \tag{5.12}$$

To prevent underflow, we use the logarithmic version of equation (5.12):

$$\arg\max_i \sum_{j=1}^{n} \big(\log P(y_j^t|\mathcal{C}_i, \boldsymbol{x}_j^t) + \log P(\boldsymbol{x}_j^t|\mathcal{C}_i, \boldsymbol{x}_1^t, \dots, \boldsymbol{x}_{j-1}^t)\big). \tag{5.13}$$

Calculating equation (5.13) for all instances is very time-consuming, therefore, we randomly choose a subsample of the square root size of the batch to make the computations. Lines 2–3 of Algorithm 7 summarizes the explained procedure.

Bayesian batch assignment method: This method is similar to the exact Bayesian method in the sense of estimating the expression in the right hand side of equation (5.7). However, in the Bayesian method, we make a simplifying assumption and estimate the equation as:

$$\begin{aligned}\arg\max_i P(\mathcal{C}_i|B(t)) &= \arg\max_i P(B(t)|\mathcal{C}_i)\\ &= \arg\max_i P(X(t)|\mathcal{C}_i)P(Y(t)|X(t), \mathcal{C}_i).\end{aligned} \tag{5.14}$$

$P(Y(t)|X(t), \mathcal{C}_i)$ is the conditional probability that the predicted labels of $(\boldsymbol{x}_1^t, \boldsymbol{x}_2^t, \dots, \boldsymbol{x}_n^t)$ be $(y_1^t, y_2^t, \dots, y_n^t)$ using classifier $\mathcal{C}_i$. $P(X(t)|\mathcal{C}_i)$ is the likelihood of the batch $X(t)$ belonging to the i^{th} concept/classifier. As mentioned

ALGORITHM 7: Update the classifier pool with batch assignment methods

```
Input: $B(t) = (X(t), Y(t))$: instances of data batch $t$ with size $n$,
       A $Pool$ of classifiers,
       $RDC$: the Raw Data Classifier
1  if $mode = exact\ Bayesian$ then
2      $(S_{x,t}, S_{y,t}) = sub\_sample((X(t), Y(t)), r)$
       // store the labels of $S_{x,t}$
3      $(S_{max}, C_{best}) =$
       $(\max, \arg\max)_{c:1..Pool.size} ExactBayesian(Pool, RDC, c, (S_{x,t}, S_{y,t}))$
       $from\ equation$ (5.13)
4  else if $mode = Bayesian$ then
5      $X_{RDC} = sum\_data(X(t))$
6      $(S_{x,t}, S_{y,t}) = sub\_sample((X(t), Y(t)), r)$
       // stores the labels of $S_{x,t}$
7      $(S_{max}, C_{best}) =$
       $(\max, \arg\max)_{c:1..Pool.size} Bayesian(Pool, RDC, c, (S_{x,t}, S_{y,t}), X_{RDC})$
       $from\ equation$ (5.18)
8  else if $mode = Heuristic$ then
9      $(S_{x,t}, S_{y,t}) = sub\_sample((X(t), Y(t)), r)$
       // stores the labels of $S_{x,t}$
10     $(S_{max}, C_{best}) = (\max, \arg\max)_{c:1...Pool.size} Pool[c].accuracy(S_{x,t}, S_{y,t})$
```

earlier, if there is a concept drift, the i.i.d assumption will not hold anymore. To assure the i.i.d assumption holds between the instances of a batch, its size should be small enough. In this approach, we make the i.i.d assumption:

$$P(Y(t)|X(t), \mathcal{C}_i) = \prod_{j=1}^{n} P(y_j^t|\boldsymbol{x}_j^t, \mathcal{C}_i),$$
$$P(X(t)|\mathcal{C}_i) = \prod_{j=1}^{n} P(\boldsymbol{x}_j^t|\mathcal{C}_i). \tag{5.15}$$

The $P(y_j^t|\boldsymbol{x}_j^t, \mathcal{C}_i)$ term can be calculated directly from the i^{th} classifier. To calculate $P(\boldsymbol{x}_j^t|\mathcal{C}_i)$, we create an RDC classifier similar to the one used in the exact Bayesian method. It gets one instance as input and predicts the corresponding concept/classifier ID in the pool. The training procedure is the same as that described in the exact Bayesian section. To reduce the time complexity of RDC training, instead of individual instance update, we use $X_t = \sum \boldsymbol{x}_j^t$, i.e. sum of all elements of batch t. Assume that RDC predicts the instances of batch t in concept i with the probability p_i:

$$P(X(t)|\mathcal{C}_i) \propto p_i^n. \tag{5.16}$$

To determine the best classifier, equation (5.14) turns into:

$$\arg\max_i P(\mathcal{C}_i|B(t)) = \arg\max_i p_i^n \prod_{j=1}^{n} P(y_j^t|\boldsymbol{x}_j^t, \mathcal{C}_i). \tag{5.17}$$

ALGORITHM 8: Determine classifier weights

```
Input: $B(t) = (X(t), Y(t))$: instances of data batch $t$ with size $n$,
       A $Pool$ of classifiers,
       $\beta$: penalty parameter
1 $(S_{x,t}, S_{y,t}) = sub\_sample((X(t), Y(t)), r)$
2 for j=1 to Pool.size do
3     $e_c = Pool[j].error(S_{x,t}, S_{y,t})$
4     $w_j = \beta^{2^{e_c}}$
```

To prevent underflow of the products, we use the following equation to find the best concept:

$$\arg\max_i P(\mathcal{C}_i|B(t)) = \arg\max_i n \log p_i + \sum_{j=1}^{n} \log P(y_j^t|\boldsymbol{x}_j^t, \mathcal{C}_i). \tag{5.18}$$

Still this equation leads to an inefficient approach as calculating the posterior for all $\boldsymbol{x}_j^t$ is time-consuming. Assuming the i.i.d condition, we again can use a subsample of the batch. Lines 5–7 of Algorithm 7 presents the Bayes batch assignment method.

Heuristic batch assignment method: The heuristic approach follows a simple idea, and that is to calculate the accuracy of all classifiers on $X(t)$ and update the most accurate one. Hence, the similarity measure is the accuracy of the classifiers on $X(t)$. Line 10 of Algorithm 7 finds the best classifier according to this approach.

5.1.3 Update parameters for the next iteration

In the last step, we should update some parameters for the following batches. If we use the so-called active classifier, we should keep the classifer that is updated in the current iteration (i.e. C_{best} in Algorithm 4). If we use the weighted classifier method, the weights of the classifiers are updated by:

$$w_0(i) = \beta^{2^{e(i)}}, \tag{5.19}$$

where $e(i)$ is the error of the i^{th} classifier. The more erroneous classifier, the less initial weight it has for the next iteration. Some locality assumption is used in equation (5.19) for setting the initial weights, which do not work properly when a sudden concept drift occurs. As mentioned in Section 5.1.1, that drift is handled by updating the weights during batch processing (Algorithm 8).

5.2 Experiments

In order to evaluate the performance of the proposed framework, experiments on some standard datasets are conducted. We first introduce the datasets in Section 5.2.1. These datasets contain recurring drifts. Then in Section 5.2.2, we discuss the parameter setting of PASC and other methods of comparison. Extensive experiments are explained in Section 5.2.3. The experiments show the effectiveness of the proposed framework. It can be seen that the weighted classification method outperforms the active classifier in datasets that contain sudden concept drifts. The proposed batch assignment methods outperform the CCP batch assignment method in datasets with arbitrary features. Moreover, the effectiveness of the merging procedure is shown for large datasets, and a sensitivity analysis of the algorithm to its parameters is carried out.

5.2.1 Datasets

We employ three real-world and one synthetic datasets in our experiments. The synthetic dataset is the moving hyperplanes that contains sudden concept drift. Emailing list, spam filtering, and sensor data are the real-world datasets used in our experiments. The emailing list and the spam filtering datasets are counted as high dimensional datasets, and the sensor dataset has a large number of instances. Emailing list and hyperplane datasets contain sudden concept drift, while spam filtering and sensor data contain gradual drift.

- Emailing List [92, 47]: In this dataset, a stream of email from different topics were collected and labeled as interesting or junk with respect to the user's interest [92]. Usenet posts data[1], which contains 20 newsgroups collection is used to construct this dataset, and three of its topics are selected. In each time interval (concept), the user is interested in one or two topics and labels the emails according to his/her interest. User interests may vary from time to time, so the dataset contains recurring concepts and sudden drifts. The labels and the existing drifts of this dataset are artificial. So elist is not a pure real-world dataset, but its instances are derived from a real-world application. The dataset has 1500 instances and 913 attributes and is divided into five time-intervals of equal width (Table 5.1).

- Spam Filtering [47]: The spam filtering dataset is extracted from the Spam Assassin[2] collection. It contains 9324 email messages with 500 attributes and two possible labels. This dataset consists of gradual concept drift.

[1] `http://archive.ics.uci.edu/ml`

[2] The Apache SpamAssassin Project `http://spamassassin.apache.org/`

Table 5.1: Emailing List Dataset (Elist) [92]

	1-300	300-600	600-900	900-1200	1200-1500
Medicine	+	-	+	-	+
Space	-	+	-	+	-
Baseball	-	+	-	+	-

- Hyperplane: This dataset aims to predict the class of a rotating hyperplane. A hyperplane decision surface is represented by equation $g(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} = 0$, where $\boldsymbol{w}$ is an n-dimensional vector showing the orientation of the surface and $\boldsymbol{x}$ is the instance. If for an instance we have $g(\boldsymbol{x}) > 0$, we classify it as 1, otherwise it is classified as 0. The hyperplane is moving over time and simulates sudden concept drift. We generated 8000 instances with 30 numeric attributes. After receiving 2000 instances, concept drift occurs suddenly. There is a reappearance of concepts after the first 4000 instances to simulate the recurring concept problem.

- Sensor: This dataset is a real-world dataset, which consists of the information collected from 54 sensors deployed in Intel Berkeley Research laboratory over two months[3]. The class label is the sensor ID with 54 possible values. The dataset has five attributes, and 2,219,803 instances. The type and place of concept drift are not specified in the dataset, but it is evident that there are some drifts. For example, lighting or the temperature of some specific sensors during the working hours are much stronger than during nights or weekends.

5.2.2 Parameter setting

The proposed learning algorithm is designed in such a way that most of its parameters can be set according to the general properties of the datasets. This is not the case in the CCP framework, where setting θ inaccurately leads to significant degradation of the performance. For example, θ should be set to 4 for elist and 2 for hyperplane datasets. However, if we set θ to 4 instead of 2 for hyperplane, its accuracy will decrease by 10% (from 78% to 68%). Besides, there is no knowledge for the proper range of this parameter in the CCP framework, and small or large values of this parameter will lead to poor performance. Similar to the θ parameter in the CCP framework, our proposed framework has three parameters θ_1, θ_2 and θ_3, but their selection is straightforward. We set the θ_i parameters of the proposed methods for all four datasets equally, and obtain acceptable results.

The parameters of the performed experiments are set as what follows, except in the specified cases: If we use the weighted classifiers method, a

[3]`http://www.cse.fau.edu/~xqzhu/stream.html`

penalty parameter, $\beta \in (0, 1)$, should be set. The smaller the parameter β is, the faster the updating of the weights of the classifiers is done in response to the potential concept drift. Hence, if the dataset contains sudden concept drift, this parameter should be small. Larger values lead to more robustness to noise. In our experiments, we set this parameter to 0.1. C_{max} is another parameter that indicates the maximum number of classifiers in the pool. Setting this parameter to a proper number needs either some domain knowledge or a trial and error technique. In the following experiments, we set C_{max} to 10 for all datasets. The accuracy threshold parameter, θ_3, in the Heuristic batch assignment method is set to 0.95 for the first three datasets and 0.8 for the sensor dataset. The values are chosen by trial and error so that the overall accuracy of the method will nearly be best. θ_1 and θ_2, the thresholds of the Bayesian and exact Bayesian methods, are set to $2r \log 0.75$ and $2r \log 0.65$, respectively, where r is the number of subsampled instances. Thus, if none of the probabilities in the relevancy equations is less than 0.75 for the Bayesian method and 0.65 for the exact Bayesian method, the concept with this property is relevant to the new batch. Note that by changing the values inside the logarithms $(0.75$ and $0.65)$ to unusually large or small values, the algorithm will not deliver good results. The batch size is set to 500 for hyperplane and 50 for the other datasets. These batch sizes do not violate the stationary property of the data.

The CCP framework has two parameters, namely C_{max} and θ. In order to provide a fair comparison between the proposed method and the CCP framework, we set C_{max} to 10 for all datasets as in PASC, and the parameter θ is chosen by trial and error so that the best accuracy is obtained. θ is set to 4, 2.5, 0.1 and 2 for the elist, spam filtering, hyperplane, and sensor datasets, respectively. As a result, the parameter setting of the PASC framework is more straightforward than the CCP framework, as the parameters can be set according to general properties of the datasets. Moreover, almost the same parameter values are shown to work well on various datasets. These datasets are different in their features and nature, but the applied methods only care about the correctness of the classifiers of the pool and do not depend on the inherent features of the datasets. Therefore, parameters are less dependent on the datasets.

5.2.3 Results and discussion

We perform three experiments in order to evaluate the PASC method. First, we compare the performance of different variations of PASC with each other and with the CCP framework. Then, we study the effect of the merging procedure and the sensitivity of the method to the threshold parameters. Finally, we evaluate the effect of the parameter β in the performance of PASC. The two following subsections discuss the results obtained from the first experiment and the two last subsections discuss the second and third experiments.

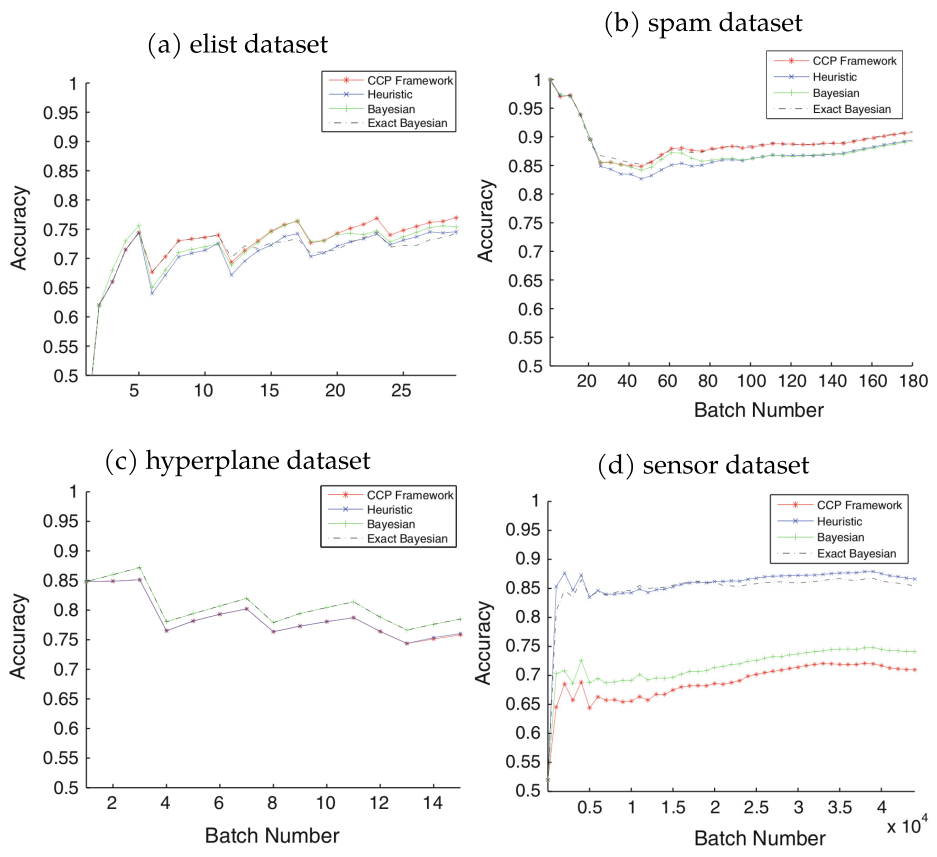


Figure 5.1: Results of the active classifier method on all datasets.

Performance comparison of methods

We compare different variations of the PASC framework with the CCP framework in terms of accuracy, precision, recall, F-measure and running time. The average accuracy values of the methods on the consecutive batches (except for the first batch, which is ignored in all methods) of elist, spam filtering, hyperplane, and sensor are shown in Figure 5.1 and 5.2. The figures consist of four parts each showing the plot of the accuracy of using the four batch assignment methods and the active classifier or the weighted classifiers methods on a given dataset. Hence, for each dataset, two plots are shown for the two classification methods. In addition, the accuracy, precision, recall and runtime of the different methods are shown in Table 5.2. Minor differences in the accuracy can be seen in this table. We can see that the precision and recall of different methods are proportional to their accuracy in most cases. Thus, instead of comparing the F-measure of different methods, their accuracy is enough.

Parts (a)–(c) of Figure 5.1 and 5.2 show that the four batch assignment methods are almost similar in the performance on the first three datasets.

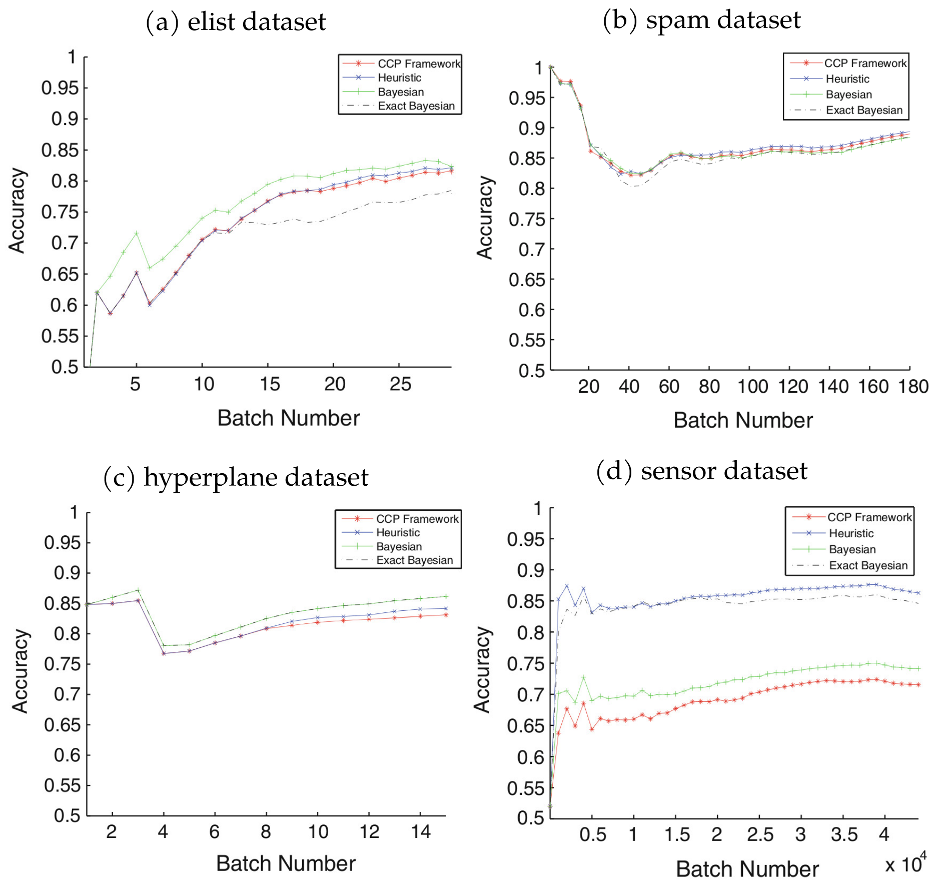


Figure 5.2: Results of weighted classifiers on all datasets.

However, in part (d), where the results on sensor dataset are illustrated, the CCP and the Bayesian batch assignment methods show worse performance (between 9% and 15% of accuracy) than the Heuristic and the exact Bayesian methods. This means that the CCP framework and the Bayesian methods have some problems in determining the true concept of a batch in the sensor dataset. One problem with the CCP framework is the use of the Euclidean distance as a measure of similarity. The distance measure in CCP is dependent on the magnitude of the attribute values, and an attribute with large values can reduce the effects of the other attributes in the distance calculation. The issue with the Bayesian method could be possibly related to its i.i.d assumption, since the exact Bayesian method does not suffer from this problem. Nonetheless, the Bayesian method still outperforms the CCP framework (about 3%). Considering the plots in Figure 5.1 and 5.2, we observe that regardless of minor differences in the accuracy of the different batch assignment methods, all batch assignment methods are affected by the drifts in the datasets in the same way. Furthermore, we observe that the weighted classifiers outperform the active classifier approach in datasets

Table 5.2: Results of all methods on different datasets.

dataset	batch assignment	classification method	Accuracy	Precision	Recall	F-measure	Runtime
elist	CCP	Active	0.77	0.73	0.81	0.77	1004
		Weighted	**0.82**	0.79	0.83	0.81	1274
	Heuristic	Active	0.75	0.71	0.77	0.74	1816
		Weighted	**0.82**	**0.80**	0.83	0.81	1843
	Bayes	Active	0.75	0.71	0.80	0.75	2089
		Weighted	**0.82**	**0.80**	**0.84**	**0.82**	2462
	Exact Bayes	Active	0.74	0.73	0.71	0.72	32039
		Weighted	0.78	0.78	0.76	0.77	32339
spam	CCP	Active	**0.91**	0.91	0.84	**0.94**	2217
		Weighted	0.89	**0.92**	0.87	0.93	2820
	Heuristic	Active	0.89	0.91	0.84	0.93	3942
		Weighted	0.89	**0.92**	0.89	0.93	4112
	Bayes	Active	0.89	0.90	0.86	0.93	4537
		Weighted	0.88	0.91	**0.91**	0.92	5405
	Exact Bayes	Active	**0.91**	**0.92**	0.86	**0.94**	109857
		Weighted	0.89	0.91	**0.91**	0.92	112266
hyperplane	CCP	Active	0.76	0.72	0.84	0.78	868
		Weighted	0.83	0.81	0.87	0.84	947
	Heuristic	Active	0.76	0.73	0.84	0.78	974
		Weighted	0.84	0.81	0.89	0.85	970
	Bayes	Active	0.78	0.75	0.86	0.80	876
		Weighted	**0.86**	**0.83**	**0.91**	**0.87**	899
	Exact Bayes	Active	0.78	0.75	0.86	0.80	1135
		Weighted	**0.86**	**0.83**	**0.91**	**0.87**	1178
sensor	CCP	Active	0.71	0.78	0.72	0.75	370560
		Weighted	0.71	0.77	0.73	0.75	813398
	Heuristic	Active	**0.87**	**0.91**	**0.89**	**0.90**	929289
		Weighted	0.86	**0.91**	**0.89**	**0.90**	846226
	Bayes	Active	0.74	0.78	0.75	0.76	883682
		Weighted	0.74	0.79	0.75	0.77	1299652
	Exact Bayes	Active	0.84	0.85	0.88	0.86	1596031
		Weighted	0.83	0.84	0.87	0.85	2184393

with sudden drift (5% to 8% difference is observable in part (a) and (c) of Figure 5.1 and 5.2). However, when the concept drift is gradual, the two classification methods work almost the same. This is intuitive, because as of occurrence of a sudden concept drift while processing a batch (especially in the beginning of a batch), the weighted classifiers method quickly adapts the weights according to the drift and therefore the performance does not degrade substantially.

Runtime comparison of methods

The runtime of different methods is listed in the last column of Table 5.2. Apart from expensive training and testing parts of the algorithms, CCP has to construct conceptual vectors and cluster them. The time complexity for updating the most similar classifier by the current batch is linear in the number of instances, and it is the same for all methods. Also, in the classification task of all batch assignment methods, each data is classified once. However, the complexity of the posterior probability calculations and updates for an instance is different for batch assignment methods.

Four terms can express the running time of the methods: T_0, T_1, T_2, and T_{RDC}, where T_0 is the time required to classify an instance, T_1 is the necessary time to find the posterior probabilities, T_2 is the needed time to update a classifier and T_{RDC} is the necessary time to use and update the raw data classifier (RDC) in the Bayesian method. To find the most similar classifier to the current batch, a subsample of size r from data in the recent window is used in our three batch assignment methods. Using all of the classifiers in the pool, each of the r instances are classified once in the Heuristic method, and its posterior probabilities estimation is measured in the Bayesian method. T_{RDC}, the computation time in updating and using RDC in Bayesian method, is constant for each batch. This term includes the time to construct X_{RDC} for the batch $X(t)$ and find the posterior probabilities for X_{RDC} using RDC, and also to update the RDC classifier with $X(t)$ and the corresponding classifier ID. The exact Bayesian method needs an update between each two posterior probability distribution computations in addition to the computation of the posterior probabilities in the Bayesian method. Besides, the same computation time for updating and posterior probabilities distribution computations are needed for RDC. Hence, the most required time for the Heuristic method is $rC_{max}T_0$, for the Bayesian method is $rC_{max}T_1 + T_{RDC}$ and for the exact Bayesian method is $2(rC_{max}T_1 + (r-1)C_{max}T_2)$, where C_{max} is the maximum number of classifiers of the pool.

As expected, the exact Bayesian method takes the most time among all. It is clear from the definition that T_1 and T_0 are almost the same for the Naïve Bayes classifier. Consequently, the Bayesian method is expected to take more time than the Heuristic method. Results in Table 5.2 uphold this fact; however, there are some minor inconsistencies for this rule in some of the methods and datasets. The reason for these inconsistencies could be due to parameter settings and the number of classifiers. In addition, it is important to note that the above expressions are written using the maximum number of classifiers and in fact, they only determine the upper bounds. It is notable to mention that the CCP framework takes the least runtime among all batch assignment methods, but the time complexity of the Heuristic and weighted classifiers methods is not much more than the running time of the CCP framework and weighted classifiers methods. As the weighted classifiers method needs updating the classifiers' weights, which in turn requires classification of some instances, it is expected to take more time than the active classifier approach. This can be seen for all batch assignment methods, except for the Heuristic method, because to obtain time-saving in the Heuristic and the weighted methods, we used the same subsample for both tasks of updating the classifiers and determining their weights. By this optimization task, the running time of the Heuristic method is not much more than the CCP framework for the weighted classifiers method. Finally, in the weighted classifiers method, the exact Bayesian and Bayesian methods take the most time. The Heuristic and the CCP framework are almost the same for some datasets, but Heuristic takes more time for the others. However, the overall results of the Heuristic method are much better than those of the CCP framework.

Impact of the merging process and the threshold parameters

To evaluate the impact of the merging process, we use the sensor dataset, as it is large enough to clearly show the impact of the different parameters. Larger datasets are more likely to assign more than one classifier to one concept, because this possibility can be more expected for the larger number of batches of instances. Parts (a) through (d) of Figure 5.3 show the accuracy of the four batch assignment methods for different values of the threshold parameters. In each part, the accuracy of the corresponding method in conjunction with the merging process is compared with the accuracy obtained when the merging process is not used. The parameters used in these experiments are the same as those used in the previous sections for the sensor dataset, except that parameter C_{max} is set to 40 instead of 10. By reducing the C_{max} parameter, it is more likely to assign more than one classifier to a concept, and there are more available classifiers that should be reduced. Therefore, the impact of the merging process is more visible.

We used a wide range of values for the threshold parameters, and observed a relatively low dependency of the methods to the threshold. This is the case with and without using the merging process. However, this does not imply that the correct setting for these parameters is not crucial. In fact, using very large or small values for the threshold parameters is not recommended. For example, part (a) of Figure 5.3, which shows the accuracy for the CCP framework batch assignment, illustrates that the method does not work well when the threshold parameter θ is very large. It is naturally expected because all the batches of instances will be assigned to one classifier in this case and no use of recurring concepts is made. Only incremental learning will be done, which is not expected to perform well on the concept drifting datasets. Although the CCP batch assignment has low dependency on the threshold parameter, its performance (accuracy) degrades considerably without using the merging process. Figure 5.3 indicates that the merging process improves the accuracy substantially for the sensor dataset. We claim that the merging process can be effective for other large datasets, too. This result can be obtained by comparing the two curves of the parts (a) through (d) of Figure 5.3. The accuracy of the methods with the merging process is more than about 60% - 70% better in comparison with the methods without the merging process. For reasonable threshold parameters (not very small or large), no value of the threshold parameters leads to a competitive accuracy to the ones with the merging process. This means that using the merging process is crucial even if the parameters are tuned very well.

The sensitivity of the threshold parameters for the other datasets can be seen in Figure 5.4. Parts (a) through (d) of Figure 5.4 shows the accuracy

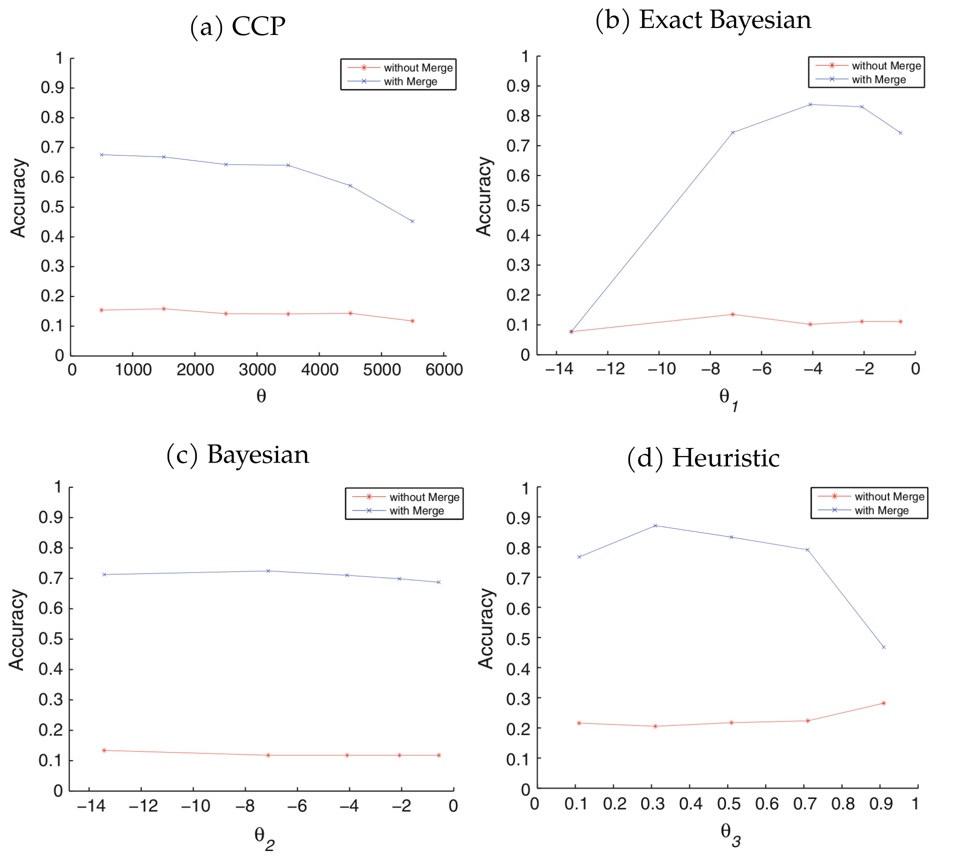


Figure 5.3: Effect of the merging process on sensor dataset for all batch assignment methods.

of the different batch assignment methods in conjunction with the merging process and the weighted classifiers method for classification. For the exact Bayesian and the Bayesian methods, θ'_1 and θ'_2 are included instead of θ_1 and θ_2, where $\theta_i = 2r\log(\theta'_i)$ and r is the batch size. The sensitivity is more than that of the sensor dataset. The parameters should not be set to very large or small values. However, the parameters of the proposed method can be chosen from a specified range of values.

Sensitivity of the weighted classifiers method to parameter β

Figure 5.5 (a) and (b) show the accuracy of the weighted classifiers method for different values of the parameter β and on the elist and spam filtering datasets. Because of the similarity of their trends, we do not include the other two datasets plots. The results are shown for the four batch assignment methods. As can be seen, the sensitivity of the weighted classifiers method to β is very low for both datasets. Only when this parameter is set to 1, the accuracy will be different, but the interval consists of the val-

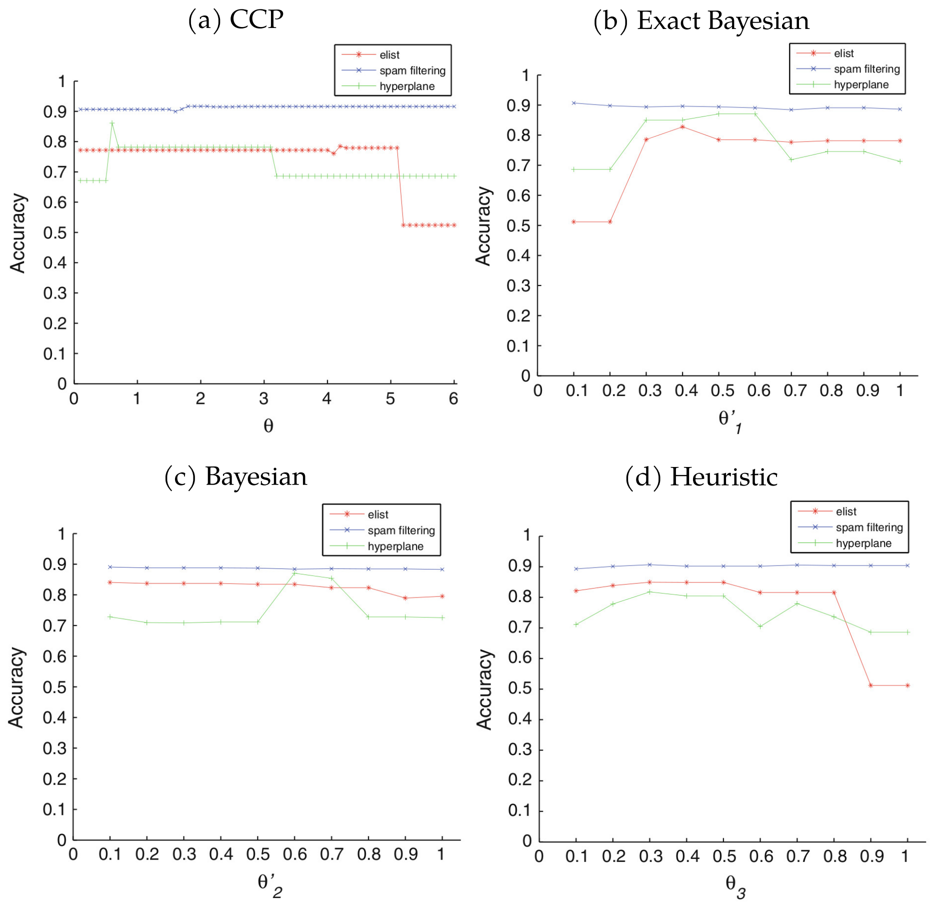


Figure 5.4: Sensitivity of the different methods to the threshold parameters. (a) Pure CCP, (b) – (d) The proposed batch assignment methods in conjunction with the merging process and the weighted classifiers method.

ues in $(0, 1)$. The results of the weighted classifiers method for the elist dataset, which contains sudden concept drift, outperform the active classifier method, for all values of β. When β is less than 1, the results are even better. For the spam filtering dataset, which contains gradual concept drift, the results of the weighted classifiers method for all values of β and the active classifier method are almost the same.

5.3 Conclusion

In this chapter, we proposed the PASC framework that aims to classify data streams in the presence of recurring concepts. A pool of classifiers is updated according to consecutive batches of instances. Each classifier in the pool is a representative of a concept. One new classification method and three batch assignment methods were introduced and compared with the

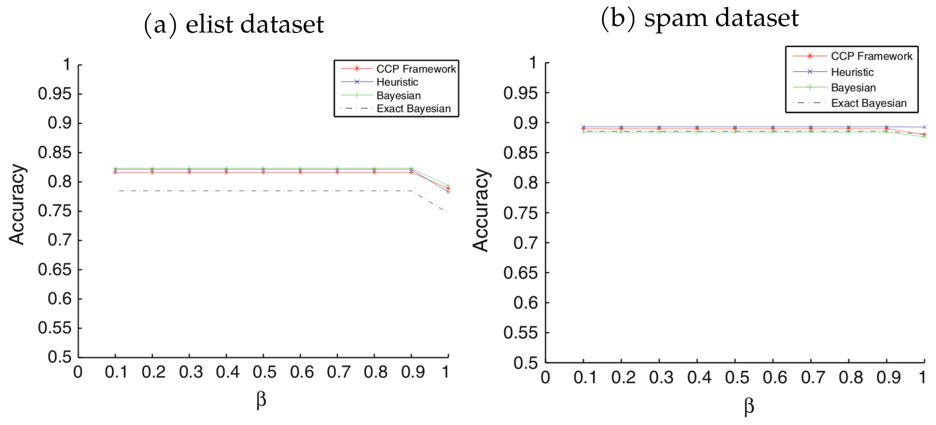


Figure 5.5: Accuracies of weighted classifiers for different values of β for the (a) elist and (b) spam datasets.

existing approaches in the CCP framework. Also, a merging process was introduced. Experimental results showed that our batch assignment, classification and merging methods improve the results on datasets with arbitrary attributes, large size, and containing sudden concept drift. In addition, the parameter setting of our method was shown to be simple according to the general properties of the datasets or a specified range of values. We used almost the same parameters for all different datasets. We can still improve the proposed framework by introducing more automatic parameter selection approaches, and more dynamic pool management operations rather than just a simple merging procedure. One may think of other similarity measures with less time complexity or better performance for the batch assignment or in the merging process.

Chapter 6

Graphical Modeling of Recurrent Concept Streams

As explained in the previous chapter, all of the existing approaches handling recurrent concepts maintain a pool of concepts/classifiers and use that pool for future classifications to reduce the error on classifying the instances from a recurring concept. While this approach is useful, it does not keep the knowledge of transitions between the concepts. Keeping the transitions between the concepts can help us in a faster and more accurate prediction, as well as in extracting patterns of behavior in a given application domain. To make it more clear, we give an example of concept transition in an online store: Most of the users tend to buy specific items on Valentine's day, and probably they come back in a few weeks to buy other items for Mother's day. Once the learner has detected these concepts and trained on some users, it can predict their behavior and suggest relevant items for Mother's day whenever Valentine's day occurs in the following years.

On the other hand, the number of classifiers in the pool usually grows very fast as the accurate detection of an underlying concept is a challenging task in itself. Current recurring concept methods manage the memory by taking a fixed size classifier pool. Many of the methods create a new classifier for each data batch, which leads to a filled pool already at very early stages. Thus, they need to deal with removing one or several of classifiers in the pool once new data batches arrive. As the data is provided in small chunks, there may be many concepts in the pool representing the same underlying concept. There has been several studies in the literature on managing the size of classifier ensembles [98, 99, 124, 87]; however, none of them consider merging the concepts instead of removing the least effective classifiers. This is where our proposed merge method comes into play. The intuition behind merging concepts is that, as each classifier is initially based on a small set of instances, there may be several classifiers in the pool distinguished as separate concepts, but all belong to one stable concept after receiving enough data. Also, one larger concept may be seen in several smaller subconcepts, and by receiving more data, the subconcepts can be merged together.

This chapter proposes a new classification framework to deal with the recurrent concepts challenge and to refine the pool of concepts by applying a merging mechanism whenever necessary. GraphPool not only keeps the concepts but also maintains the transition among concepts via a first-order Markov chain. Keeping these transitions helps to quickly recover from drifts in some real-world problems with periodic behavior. The proposed framework consists of several steps:

1. A conceptual representation is used to extract a comparable summarization of concepts. This representation is a generalization of the conceptual vectors proposed in the CCP framework [92]. In contrast to the CCP framework, it takes into account the correlations in feature space. Unlike the CCP framework that uses the Euclidean distance and a user-defined domain-specific similarity threshold, we use a likelihood multivariate statistical test to check the similarity of the new batch and the existing conceptual representations. The proposed statistical test measures whether the entropy of the combined concepts is different from the entropy of each concept alone.

2. All the conceptual vectors and their corresponding classifiers are stored in a first-order Markov chain graph. Keeping these transitions will help in classifying recurrent concepts correctly. The weights of links are computed based on the frequency of traversing from one concept to another, adjacent one. Thus, the graphical model keeps some extra information about the drifts in addition to the concepts. We use the resulting graphical pool to classify the test data by taking the most likely concept or the majority vote of all adjacent concepts to the current state.

3. To manage the pool size and also to enhance the generalization of the concepts, a merging method is proposed. Merging concepts aims to increase the efficiency of the proposed method while generalizing the concepts that are detected to be similar to each other by the similarity measure. To the best of our knowledge, this is the first work that, instead of adding and removing concepts, actively uses a merger to control the pool size.

There is another closely related task, the so-called sequential supervised learning, that one may find some similarities with our problem, but it is still different in nature. Sequential classification methods also consider non-stationary data with discrete transitions among distributions with the Markov property [44]. However, in sequential supervised learning, the entire sequence is available before we make any predictions of the class. That is totally different from data stream mining, where we only have seen instances up to the current point in time. One successful method in sequence classification is called Input-Output Hidden Markov Models (IOHMM) [13]. IOHMMs are non-homogeneous Markov chains, i.e., the emission and transition probabilities depend on time and input. Unlike traditional HMMs that represent only the distribution of output sequences, IOHMMs use a supervised discriminative training algorithm to represent the conditional sequence of output sequences given corresponding input sequences. IOHMMs typically use complex nonlinear emission and transition distributions based on neural networks and apply Expectation-Maximization (EM) to train all the hidden variables; therefore, the method suffers from high computational complexity. Also, the topological structure of an IOHMM, i.e., the number of states and admissible transitions between different states, needs to be determined prior to its training, which is not possible in our problem setting as we are not aware of the number of concepts in advance.

6.1 Graphical representation of classifier pool

As introduced above, the basic idea of our method is to keep track of concepts and their transitions by graphical modeling. Graphical modeling of concepts is one way of modeling the behavior of an environment and should help to detect recurring concepts quickly. It could also be beneficial for extracting patterns in some applications, e.g., the weather forecast or the stock market. Each concept in the model is represented by a concept representative vector extracted from the data and a corresponding classifier. We assume that the batch size is small enough to contain only stationary instances. Then we find the best normal distribution approximation on the data batch; hence, the concept representative vector keeps the mean and covariance matrix of the data for each class. We extract the conceptual vector for each batch of data and compare it to the other concepts in the pool. The comparison is made by a multivariate statistical test. If the test indicates high statistical significance, a new concept is added to the pool. Otherwise, if the current batch is similar to only one concept in the pool, the corresponding concept will be updated. There may be some cases that the new batch is similar to more than one concept in the pool. In these cases our method benefits of a merge procedure. The merge procedure helps to generalize the concepts and combine those concepts which in the first attempt were misdetected as different concepts due to the lack of sufficient data. This may happen because data batches should be stationary and contain only one concept, and thus, the batch size is chosen to be small. Therefore, it is very likely to encounter a concept in several batches, but each time some different part of the concept. This leads to detect each batch as a new concept in the early stages. However, after some time, by receiving more and more data, the batch may contain some instances, which connect these separated parts. At this point, there may be two or more concepts in the pool, which are similar to the current batch. When this happens, we will merge all those concepts and update the graphical representation by reducing vertices, updating the conceptual vector, the classifier, and merging their edges and edge weights. In the following subsections, we will explain our conceptual representation and graphical pool modeling. Then the statistical test that checks the similarity of conceptual vectors will be explained. Finally, we show how the merging procedure works.

6.1.1 Conceptual representation

The idea of extracting a conceptual vector from each batch of data was first presented in the Conceptual Clustering and Prediction (CCP) framework [92]. CCP makes two main assumptions on extracting conceptual vectors: (1) attributes are independent, (2) continuous features are drawn from the normal distribution. In this chapter, we release the first assumption as it is not valid in many real-world data. To overcome this limitation, we calculate the mean vector and covariance matrix of attributes for each class, instead of just extracting their mean and variance. In this way, we keep the dependencies among the attributes and consider them while checking for the similarity of two concepts. Considering the correlation between attributes will help us in a more accurate estimation of the data and also in the correct detection of those drifts which happen in feature dependencies.

6.1.2 Pool representation

The classifier pool is represented by a first-order Markov chain. The states contain a conceptual representation and their corresponding classifier. State i (S_i) is connected to state j (S_j), if concept j occurs exactly after concept i. The transition weight of S_i to S_j is the probability of the transition from concept i to concept j when we are at S_i, which is $P(S_j|S_i)$. Keeping these state adjacencies and transition weights will help us in classifying test data.

6.1.3 Similarity measure

Let $\boldsymbol{x}_i^t$ denote the i^{th} instance of batch t. Due to the central limit theorem, we can approximate the distribution of instances as a normal distribution from $\mathcal{N}(\boldsymbol{\mu}_c^t, \boldsymbol{\Sigma}_c^t)$ for each class c, if the number of instances is large enough. Making this assumption, we use the likelihood ratio as a testing criterion to check whether two batches (concepts) are significantly different. The likelihood ratio compares the within-covariance of the population ($\hat{\boldsymbol{\Sigma}}_\Omega$) with its between-covariance $\left(\sum N_t(\bar{\boldsymbol{x}}^t - \bar{\boldsymbol{x}})(\bar{\boldsymbol{x}}^t - \bar{\boldsymbol{x}})^T\right)$. $\bar{\boldsymbol{x}}^t$ is the mean of population t and is calculated by $\bar{\boldsymbol{x}}^t = \frac{1}{N_t}\sum_i \boldsymbol{x}_i^t$ and $\bar{\boldsymbol{x}} = \frac{1}{N}\sum_{t,i}\boldsymbol{x}_i^t$ is the total mean of all instances to be compared. N_t is the number of instances in the population t and N is the total population. Consequently, the likelihood ratio λ is calculated as [7]:

$$\lambda = \frac{\mid N\hat{\boldsymbol{\Sigma}}_\Omega \mid}{\mid N\hat{\boldsymbol{\Sigma}}_\omega \mid}, \text{ where} \tag{6.1}$$

$$N\hat{\boldsymbol{\Sigma}}_\Omega = \sum_{t,i}(\boldsymbol{x}_i^t - \bar{\boldsymbol{x}}^t)(\boldsymbol{x}_i^t - \bar{\boldsymbol{x}}^t)^T, \text{ and} \tag{6.2}$$

$$N\hat{\boldsymbol{\Sigma}}_\omega = \sum_{t,i}(\boldsymbol{x}_i^t - \bar{\boldsymbol{x}})(\boldsymbol{x}_i^t - \bar{\boldsymbol{x}})^T. \tag{6.3}$$

We can simplify equation (6.2):

$$\begin{aligned}
N\hat{\boldsymbol{\Sigma}}_\Omega &= \sum_{t,i}(\boldsymbol{x}_i^t - \bar{\boldsymbol{x}}^t)(\boldsymbol{x}_i^t - \bar{\boldsymbol{x}}^t)^T \\
&= \sum_{t,i}\left(\boldsymbol{x}_i^t\boldsymbol{x}_i^{tT} - \bar{\boldsymbol{x}}^t\boldsymbol{x}_i^{tT} - \boldsymbol{x}_i^t\bar{\boldsymbol{x}}^{tT} + \bar{\boldsymbol{x}}^t\bar{\boldsymbol{x}}^{tT}\right) \\
&= \sum_{t,i}\boldsymbol{x}_i^t\boldsymbol{x}_i^{tT} - \sum_t\left(\bar{\boldsymbol{x}}^t\sum_i\boldsymbol{x}_i^{tT} + \big(\sum_i\boldsymbol{x}_i^t\big)\bar{\boldsymbol{x}}^{tT} - \sum_i\bar{\boldsymbol{x}}^t\bar{\boldsymbol{x}}^{tT}\right) \\
&= \sum_{t,i}\boldsymbol{x}_i^t\boldsymbol{x}_i^{tT} - \sum_t\left(\bar{\boldsymbol{x}}^tN_t\bar{\boldsymbol{x}}^{tT} + N_t\bar{\boldsymbol{x}}^t\bar{\boldsymbol{x}}^{tT} - N_t\bar{\boldsymbol{x}}^t\bar{\boldsymbol{x}}^{tT}\right) \\
&= \sum_{t,i}\boldsymbol{x}_i^t\boldsymbol{x}_i^{tT} - \sum_t N_t\bar{\boldsymbol{x}}^t\bar{\boldsymbol{x}}^{tT}.
\end{aligned}$$

Also, equation (6.3) can be rewritten as:

$$\begin{aligned}
N\hat{\boldsymbol{\Sigma}}_\omega &= \sum_{t,i}(\boldsymbol{x}_i^t - \bar{\boldsymbol{x}})(\boldsymbol{x}_i^t - \bar{\boldsymbol{x}})^T \\
&= \sum_{t,i}\left(\boldsymbol{x}_i^t\boldsymbol{x}_i^{tT} - \bar{\boldsymbol{x}}\boldsymbol{x}_i^{tT} - \boldsymbol{x}_i^t\bar{\boldsymbol{x}}^T + \bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^T\right) \\
&= \sum_{t,i}\boldsymbol{x}_i^t\boldsymbol{x}_i^{tT} - \left(\bar{\boldsymbol{x}}\sum_{t,i}\boldsymbol{x}_i^{tT} + \big(\sum_{t,i}\boldsymbol{x}_i^t\big)\bar{\boldsymbol{x}}^T - \sum_{t,i}\bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^T\right) \\
&= \sum_{t,i}\boldsymbol{x}_i^t\boldsymbol{x}_i^{tT} - N\bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^T \\
&= \sum_{t,i}\boldsymbol{x}_i^t\boldsymbol{x}_i^{tT} - \sum_t N_t\bar{\boldsymbol{x}}^t\bar{\boldsymbol{x}}^{tT} + \sum_t N_t\bar{\boldsymbol{x}}^t\bar{\boldsymbol{x}}^{tT} - N\bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^T \\
&= N\hat{\boldsymbol{\Sigma}}_\Omega + \sum N_t(\bar{\boldsymbol{x}}^t - \bar{\boldsymbol{x}})(\bar{\boldsymbol{x}}^t - \bar{\boldsymbol{x}})^T.
\end{aligned}$$

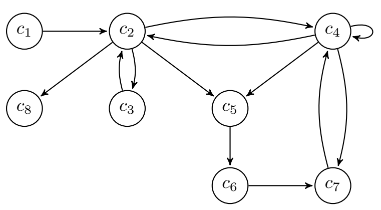
ALGORITHM 9: Merging algorithm

```
Input: a set of similar concepts, current graph, current data batch
1  initialize $MergedVertex$ to the first concept of similar concepts set
2  for each remaining vertex $v$ in similar concepts set do
3      for each vertex $u$ in which $v$ is in its neighborhood do
4          if $MergedVertex$ is in $u$'s neighborhood then
5              update and normalize corresponding $Transitionweight$
6              remove $v$ from $u$'s neighborhood
           else
7              add $MergedVertex$ instead of $v$ to neighbors of $u$
8              add $u$ to neighbor set of $MergedVertex$
9      for each $w$ in $v$ neighborhood do
10         if $w$ is in $MergedVertex$'s neighborhood then
11             update corresponding $Transisionweight$ from $MergedVertex$ to
                 $w$;
           else
12             add $w$ to $MergedVertex$ neighborhood with its corresponding
                 weight
13         remove $v$ from $w$ neighborhood
14 normalize transition weights from $MergedVertex$
15 update $MergedVertex$ with all similar concepts (due to corresponding
     Conceptual Vector and Learner)
16 update $MergedVertex$ with current batch
17 remove all vertices of similar concept set from pool
```

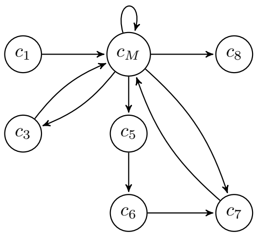
Now, by these simplifications in the within-covariance and between-covariance, it is straightforward to calculate λ from our conceptual representation vectors. When the null hypothesis is true $\left(\mu^{(1)} = \mu^{(2)}\right)$, λ from equation (6.1) is distributed as Wilk's lambda distribution: $U_{m,q-1,n}$, where $n = N - q$ and m is the number of attributes and q is the number of populations (here 2) [127]. Wilk's lambda distribution is a multivariate generalization of the F-distribution. It is the ratio of two independent Wishart distributed variables, the underlying distribution of the determinant of the covariance matrix. If the number of instances, n, is large enough, we may assume $-(n-1)\log\lambda$ is distributed as χ^2 distribution with $m(q-1)$ degrees of freedom (this estimation is valid when the number of observations is greater than 20 [7]). This test is done for each possible class value, and if there is a significant difference in one of them, we decide that the new batch contains a new concept.

6.1.4 Merge concepts

If the statistical similarity test detects two concepts are not significantly different, we take them to be similar with respect to current knowledge and the



concept	Neighbors	InNeighborsOf
1	{2}	{}
2	{3,4,5,8}	{1,3,4}
3	{2}	{2}
4	{2,4,5,7}	{2,4,7}
5	{6}	{2,4}
6	{7}	{5}
7	{4}	{4,6}
8	{}	{2}



concept	Neighbors	InNeighborsOf
1	{M}	{}
3	{M}	{M}
5	{6}	{M}
6	{7}	{5}
7	{M}	{M,6}
8	{}	{M}
M	{M,3,5,7,8}	{M,1,3,7}

Figure 6.1: Example illustrating the merging process in the graphical concept pool. The first graph shows the concept pool before merging and the second one after. c_2 and c_4 are merged to give c_M.

next step is to merge them. Merging consists of several steps (Algorithm 9). First, the two conceptual representations should be merged. The proposed conceptual representation model facilitates the merge. We can substitute the new conceptual representation by the linearity feature of mean and covariance. The new mean is calculated from $\frac{\boldsymbol{\mu}_i \times N_i + \boldsymbol{\mu}_j \times N_j}{N_i + N_j}$, and the new covariance is obtained from $\frac{(N_i-1)\times Cov(X^i,Y^i)+(N_j-1)\times Cov(X^j,Y^j)}{(N_i+N_j-2)}$. In case of interest, the linearity feature can facilitate weighing more on the most recent concept than the older ones by adding a decay factor.

The next step is to merge the classifiers. Any classifier that can be merged in the absence of training data can be used. Here, we use the Naïve Bayes Updateable classification algorithm. As Naïve Bayes keeps posterior statistics of feature occurrence for each class, we can easily merge them in the absence of data batches.

The last step in the merge procedure is to merge the transition links of these two concepts in the Markov graphical model. To illustrate this, we give an example (Figure 6.1). Assume the current snapshot of the concept graph is the first graph in Figure 6.1, and the statistical similarity test has indicated that c_2 and c_4 are similar to the newly arriving batch of data. Now, these two vertices should be merged according to our approach. Assume the merged concept is called c_M. For each concept in the pool, we keep two arrays: the array of *Neighbors*, which contains the outward links of the concept, and the array of *InNeighborsOf*, which consists of inward links to the concept. First, the links in *Neighbors* list of those concepts which have the transition link to c_2 and c_4 are replaced by c_M. Then the *Neighbors* list of the new concept is updated. In the updating process, we distinguish three different cases:

1) The concept is only in the neighborhood of one of the concepts to be merged (e.g., c_7 and c_8). This is an easy case, as we transfer this concept to the new *Neighbors* list, keeping its weight.

2) The concept is in the neighborhood of both concepts (e.g., c_5). Again, this concept is transferred to the *Neighbors* list of the new concept; however, its transition weights are summed up.

3) The other concept which should be merged with is in the *Neighbors* list (e.g., c_4 is in the list of c_2 and vice versa). This makes a loop case. Therefore, in the current example, the weight of the loop over c_M is the summation of weights of the c_2 to c_4 link, the c_4 loop, and the c_4 to c_2 link.

One should consider that to make the summation meaningful, and we sum up the number of times the transition is traversed, not the probability. In the previous example, assume the probability of the link between c_4 and c_5 is 0.1 and number of traversals is 100, and the probability of the link between c_2 and c_5 is 0.2, and the number of traversals is 20. It is clear that taking the average of the probability while ignoring the number of traversals would give the wrong results.

Relatively close to our merging mechanism, some researches keep track of cluster evolution in the context of spatio-temporal clustering [91, 167, 132, 133, 68]. These approaches trace changes in clusters and evaluate whether a cluster has been disappeared, or its members have been migrating to other clusters; or whether a new emerging cluster reflects a new group or it rather consists of existing clusters. MONIC [167] is one of the first frameworks, which models and tracks those cluster transitions. Its transition tracking mechanism is independent of any clustering algorithm and is based on the content of the underlying data stream. However, its need for post-processing clustering results at each time slot makes it infeasible for real-time problems. TRACDS [68] enhances MONIC by incrementally constructing a transition-count matrix whenever an instance is assigned to a specified cluster. The MEC [133] framework uses a transition detection algorithm and a tracking mechanism and keeps a taxonomy of various types of clusters' transitions. However, these methods are unsupervised and fundamentally different than our method. In the context of data stream classification, some methods combine micro-clustering and classification algorithms [3, 119]. In order to cover concept evolution, these methods benefit from a simple merging mechanism of micro-clusters.

6.1.5 Classifying a new batch of data

Let $CurrentState$ denote the current state at time t when data batch $X(t)$ is received to be classified. There could be two approaches for the classification of instances: using a single classifier or using a weighted majority vote (Algorithm 10). In the single classifier prediction method, we choose the

ALGORITHM 10: Classify new data batch

```
Input: $X(t)$: instances of batch $i$, $CurrentState$ of the $Pool$
1 if There is no concept in the neighborhood of $CurrentState$ then
2     Test $X(t)$ on $CurrentState$'s classifier
  else
3     switch classification method do
4         case single classifier do
5             Find the concept with the highest link's weight in the
                neighborhood of $CurrentState$
6             Test $X(t)$ on its corresponding classifier
7         case voted majority do
8             for All concepts in the neighborhood of $CurrentState$ do
9                 Get the weighted vote of their corresponding classifier based
                    on the link's weight
```

most probable concept in the neighborhood of the $CurrentState$ and use its corresponding classifier to predict the label of the new batch. In contrast, in the weighted majority vote method, we take the vote of all the classifiers of the concepts in the neighborhood of $CurrentState$ with respect to their transition weights. If there is no classifier in the neighborhood, we take $CurrentState$ as the potential concept for the following batch.

6.1.6 GraphPool algorithm

In this section, we explain Algorithm 11 and put together all the parts explained so far. In the beginning, the pool is empty, and the current state is null (line 1– 2). So, for the first batch of data, a new concept is added to the pool and the current state points to that (lines 5– 8). For the next batches of data, we first look at the current state's links and based on whether the classification method is single classifier or majority vote, we choose the most likely concept or a weighted vote of all concepts in the neighborhood to predict the label of the new batch (Algorithm 10). If there is no classifier in the neighborhood, we take the current state as the potential concept for the following batch.

After receiving labels of the new batch, the conceptual vector is calculated and compared to the concepts available in the pool (lines 3– 4):

1) If there is no similar concept to this batch in the pool, it should be added as a new concept to the pool and connected as a neighbor to the current state. Then the transition link weights of the current state are normalized (lines 9– 13).

2) If there is only one concept similar to the current batch (V_j), its conceptual representation and classifier are updated in a straightforward way; and the transition link between the current state and V_j is created, if it does

ALGORITHM 11: GraphPool algorithm

```
Input: $B(t) = (X(t), Y(t))$: labeled instances of batch $i$
1  Pool = $\phi$
2  CurrentState = $\phi$
3  make Conceptual Vector on $B(t)$
4  Compare the current Conceptual Vector with the concepts in the pool
5  if Pool is empty then
6      train a classifier on $B(t)$
7      add a new vertex $(j)$ to the Pool
8      update $CurrentState$ to $j$
   else
9      if number of similar concepts = 0 then
10         train a classifier on $B(t)$
11         add a new vertex $(j)$ to the Pool
12         add $j$ to the neighbors of $CurrentState$ and adjust their weights
13         update $CurrentState$ to $j$
14     else if number of similar concepts = 1 then
15         update Conceptual Vector and classifier of the similar concept (Vertex
            $j$)
16         if Vertex $j$ is in the neighborhood of $CurrentState$ then
17             update the weight of link between $CurrentState$ and $j$
           else
18             add $j$ vertex to $CurrentState$ neighborhood
19         adjust weights of link from $CurrentState$ to its neighbors
20         update $CurrentState$ to $j$
21     else if number of similar concepts > 1 then
22         execute the merging method (Algorithm 9)
23         update $CurrentState$ to $MergedVertex$
```

not exist yet; or only its weight is updated, if it already exists (lines 14– 20).

3) There is more than one concept similar to the current batch. As described in Section 6.1.4, we merge all these concepts into one concept, including the new batch (Algorithm 9). The graph transition update process is done as explained in the example of the previous section: For each concept to be merged, first substitute it for the merged concept in all other vertices. Then, update the neighborhood list of the merged concept.

6.1.7 Details of the GraphPool implementation

In the implementation of the GraphPool method, we faced some exceptional cases that led to adding some details to the implemented method:

- Remove noise in covariance matrix: In some real-world datasets, there were batches for which the magnitude of the covariance of the features was very small. To remove the noise in the corresponding batch

Table 6.1: Characteristics of tested datasets.

Dataset	#Instances	#Attributes	#Classes
Hyperplane	1,000,000	2	2
Waveform	1,000,000	21	3
Gaussian	6,000	2	4
Electricity	45,312	8	2
Sensor	847,923	6	3
Weather	10,414,909	10	2

and avoid the potential underflow in calculations, we have set a noise threshold and discarded covariance values below the threshold.

- Ignore unchanging features in the determinant calculation: There may be a case where one feature is fixed in a data batch and hence its variance is zero. This will make the determinant of the covariance matrix become zero. To avoid such cases, we ignore that feature only in the calculation of lambda in the statistical test.

- If $|N\hat{\mathbf{\Sigma}}_\Omega| = |N\hat{\mathbf{\Sigma}}_\omega| = 0$, we set the λ to 1. This case may happen as we remove small covariances as noise.

- Merge of two classifiers in the absence of instances: When it is time for the merging of concepts, we need to merge the classifiers in the absence of data. The Naïve Bayes classifier has a straightforward merging approach of distribution estimators, and that is the reason we have used it here.

6.2 Experimental results

In this section, we present the results of the proposed method (GraphPool) on synthetic and real-world datasets[1]. Experiments are done in a prequential way, which means each batch of data is first taken as test data and labeled by the current model; in the next time slice, the batch and its real labels are used to update the model. All the algorithms are implemented in the WEKA [70] or MOA [17] workbench and the average of accuracy, precision, recall and F-measure over all the batches are reported for each method in Table 6.2.

6.2.1 Datasets

We run the experiments on both synthetic and real-world datasets. Synthetic datasets are generated to cover different types of drifts. Three real-world datasets have been used in the experiments: Electricity, Sensor data,

[1]The GraphPool source code is available at `https://github.com/kramerlab/GraphPool`.

and US weather. Characteristics of the datasets are shown in Table 6.1. In the following we introduce each dataset:

- Moving hyperplane: One common synthetic dataset is based on a moving hyperplane [83]. A moving hyperplane in d-dimensional space is denoted by $\sum_{i=1}^{d} a_i x_i = a_0$. We label examples satisfying $\sum_{i=1}^{d} a_i x_i \geq a_0$ as positive, and the rest as negative. In order to have the possibility of data visualization, we generated hyperplanes in 2-dimensional space, consisting of one million uniformly distributed random instances. There is 5% uniformly distributed noise in the dataset. Figure 6.2 visualizes the sudden drifting dataset. It consists of 3 concept drifts that occur at points $250,000$, $500,000$ and $750,000$. The concept from T_1 to T_{250} is fixed; then at T_{251} a new concept occurs, which lasts until T_{500}. At T_{501} and T_{751} the first and the second concepts reoccur respectively. On the other hand, the second dataset (Figure 6.3) contains both gradual and sudden drifts. Gradual drift occurs after each instance. Figure 6.3 represents how the hyperplane moves over the batches of data; from the first instance to $125,000$ (T_{125}) there are slight changes in the concept after receiving each instance; then, there is a sudden drift to a previously seen concept at T_{126}. Again, slight gradual changes move the hyperplane until we receive the batch T_{250}. At T_{251}, the concept is exactly the inverse of the concept in the first batch. We repeat this pattern three more times to obtain one million instances.

- Waveform: Another synthetic dataset is a stream of three waveform types with 21 attributes. We have used the MOA framework [17] to generate this dataset. First, we generated $200,000$ instances using MOA; then, we permuted the labels in the following order: $c_1 \rightarrow c_2, c_2 \rightarrow c_3, c_3 \rightarrow c_1$ to generate the next $200,000$ drifted instances. For the next $200,000$ instances, another permutation was done on the labels of the original data with the order of $c_1 \rightarrow c_3, c_2 \rightarrow c_1, c_3 \rightarrow c_2$. Finally, the first $400,000$ instances were repeated to cover recurrent concepts.

- Gaussian[2]: This dataset was first proposed and used by Elwell and Polikar [47]. It has four classes, and each class is drawn from a Gaussian distribution and has gradual but independent drifts over time. The dataset includes class addition and removal as well; class c_4 appears at instance 2387 for the first time, and class c_1 appears at instance 3580 for the last time.

- Electricity [71]: The data was collected from the Australian New South Wales Electricity Market from 1996 to 1998 at 30-minute

[2]`http://users.rowan.edu/~polikar/research/NSE/`

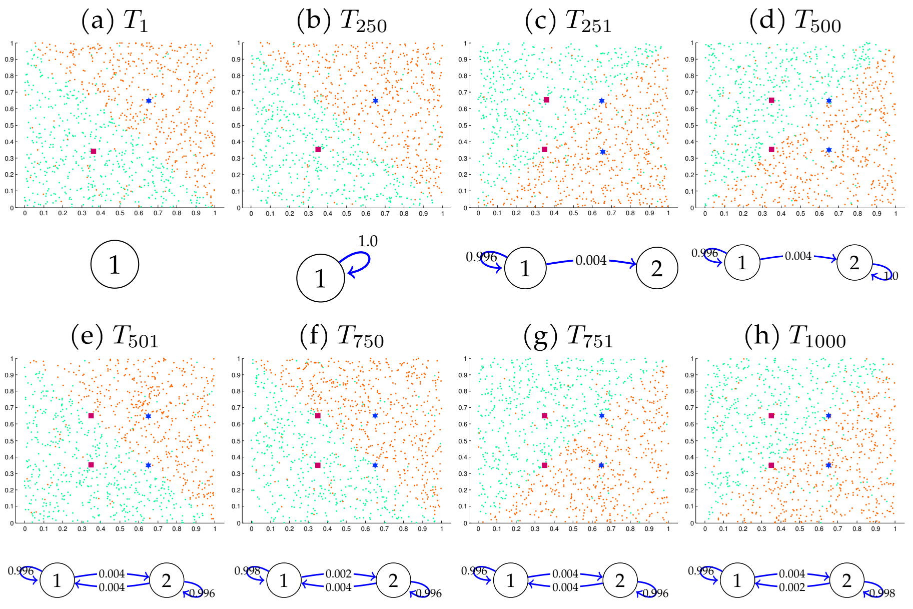


Figure 6.2: Snapshots of sudden drifting Hyperplane, illustrating concept mean vectors and the evolution of concept transition graphs. Red and green dots are data of the two classes and blue star and purple square points are mean vectors of concepts currently in the pool. Each time step represents 1000 instances.

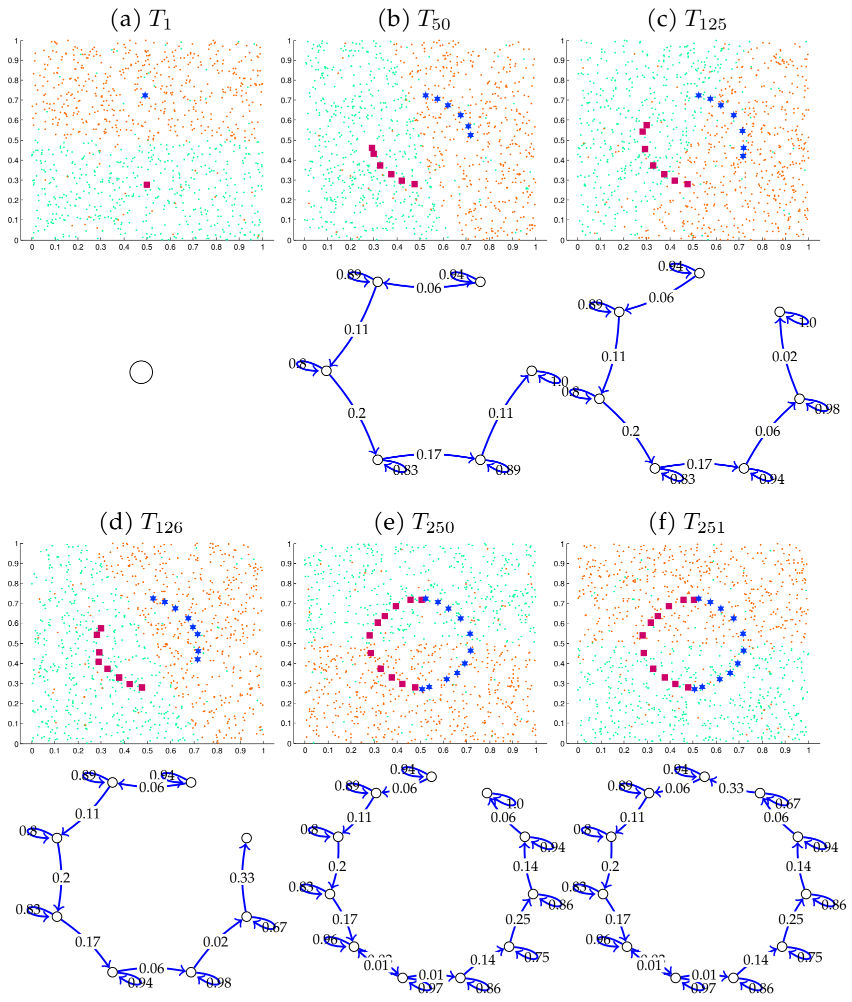


Figure 6.3: Evolution of classifier pool on Hyperplane dataset (with gradual and sudden drift). This snapshot contains gradual drift from the first batch (T_1) until the sudden drift at T_{251}. Data points are shown as red and green dots. Blue stars and purple squares indicate the mean vector of current concepts in the pool. Each time step represents 1000 instances.

intervals. The purpose is to predict if the price goes up or down regarding the last 24 hours. We have used the normalized version of dataset[3].

- Sensor: This dataset contains the temperature, humidity, light, sensor voltage and sensor location of 54 sensors in the Intel Berkeley Research Lab in March 2004[4]. There should be a report every 31 seconds; however, there are several missing epochs in the report. The purpose is to predict if the temperature increases until the next measurement (in less than 60 seconds) or stays the same or goes down. If the next epoch is missing, the class is set to equal. Note that the variations in the sensor voltage are highly correlated with the temperature. Also, the change in the light or temperature of a sensor during day and night causes some drifts.

- Weather: The U.S. National Oceanic Atmospheric Administration (NOAA) has reported weather measurements on a daily basis since the 1930s[5]. We have chosen all the stations in the U.S., which have available reports for more than 40 years. We follow the procedure used by Elwell and Polikar [47] to prepare the dataset. Seven measurements were selected based on their availability, eliminating those with a missing rate above 10%. Missing values were replaced by the mean of features. The features of the dataset are latitude, longitude, and elevation of the station, and seven measurements (e.g., temperature, pressure, wind speed, etc.). The target is a binary indicator of rain: 28.53% positive and 71.47% negative in this dataset.

6.2.2 Methods of comparison and parameter setting

We compare the performance of the proposed method to the most well-known data stream classification approaches in the literature. In all experiments, the base classifier of all methods is Naïve Bayes, and in our method, the updateable version was used. Another parameter is the batch size: On real-world datasets and Gaussian, the batch size is set to 100 instances, and on the other synthetic datasets, it is set to 1000. GraphPool has a p-value parameter, which is set to 0.01 for synthetic datasets and to make the algorithm more confident about the dissimilarities and reduce potential false alarms, it is set to 0.001 for real-world datasets. Also, the epsilon for noise removal is 0.001 in all experiments. In the following, we explain the chosen

[3]`http://sourceforge.net/projects/moa-datastream/files/Datasets/Classification/`

[4]Raw data was extracted from `http://db.csail.mit.edu/labdata/labdata.html`

[5]Raw data was extracted from `ftp://ftp.ncdc.noaa.gov/pub/data/gsod/`

parameters of each method[6]:

- Recurring Concept Drift framework (RCD)[7] [66]: The buffer size is set to the window size. The maximum size of the ensemble is set to 100. EDDM [10] is used as the drift detection method, as previous experiments have shown that it produces better results [66]. The multivariate non-parametric test is KNN with $K = 7$, and the p-value is 0.01, which are the default values in MOA. We tested if on real-world datasets decreasing the p-value to 0.001 would help the algorithm. The results on the tested datasets were slightly worse (except for Sensor, on which the results were slightly better). So, we report the results for 0.01.

- Pool and Accuracy based Stream Classification (PASC): We choose the CCP batch assignment, as the experiments show it has better accuracy in general on different types of datasets and is the fastest method. To classify the new batch, we use the so-called active classifier method that is to use the current single classifier. The maximum number of classifiers in all experiments is set to 100. The method has a θ parameter in the batch assignment, which is used in measuring the similarity of conceptual vectors. Finding the best θ for each dataset is challenging, as it can take any positive number; and this choice has a substantial effect on the accuracy of the method. For example, if on the Hyperplane dataset we choose 2.5 instead of 0.1, the accuracy degrades by about 20%. In our experiments, we tested several values and got the best results with $\theta = 0.1$ on Hyperplane (both sudden drifting dataset and the gradual-sudden drifting one) and $\theta = 2.5$ for the rest of the datasets.

- Learn++.NSE[7] [47]: We use the error-based forgetting mechanism. The maximum ensemble size is set to 100 in all experiments (reducing the ensemble size to 10 on synthetic datasets did not help in the performance). There are two more parameters in weighting, for which we use the values suggested in the paper: sigmoid slope is 0.5, and the sigmoid crossing point is set to 10.

- OzaBagAdwin[8] [18]: This algorithm is not specifically designed for recurring concepts, but it has been used widely in data stream classi-

[6]We tried to compare our method to the method presented by Yang *et al.* [201], as it has been proposed to handle recurrent concepts and has a close, yet different, approach from GraphPool. Unfortunately, we could not reach the corresponding author; and there were some unclear parts in the explanation of the method that prevents reimplementation. We have compared the concept similarity algorithm proposed by Yang *et al.* to our statistical similarity test in the following subsections.

[7]We use the implementation provided at `https://sites.google.com/site/moaextensions/` for RCD, Learn++.NSE and DWM.

[8]We used the code provided in the MOA framework.

fication. The only parameter we have to set is the ensemble size. In the first experiments, we set the parameter to 100 for all datasets, but experiments showed that on synthetic datasets it has much higher accuracy when the ensemble size is reduced to 10.

- Dynamic Weighted Majority (DWM)[7] [98]: We set the punishment factor to 0.1; the minimum fraction weight of each model to 0.01; the maximum size of the ensemble is set to 100, and the period between classifier creation/removal is set to the window size. The batch-wise accuracy of DWM fluctuates a lot even on simple synthetic datasets. Decreasing the ensemble size did not have any visible effect. When we increased the period between the creation/removal of classifiers, the fluctuations were reduced. However, it is a trade-off with performance. Finally, we decided to report better performance, which was when the period was set to the window size.

6.2.3 Proof of concept

We start the evaluation of the GraphPool algorithm with experiments on the moving hyperplane data. The first experiment tests the effectiveness of the statistical similarity measure. Figure 6.2 shows the steps of expanding the classifier pool over time on the sudden drifting dataset. The red and green dots in the plots show the data at the corresponding time slice. The means of conceptual vectors are indicated by blue and purple points on the data representation. As it can be seen, at T_1 the first concept is added to the pool with no link. Up to batch T_{250}, the similarity measure finds concepts similar to the available concept; therefore, only a link to the current concept is created. At T_{251}, the similarity test detects a sudden drift. The new concept is added to the pool and, the weights of links are also updated. Up to T_{500}, there is no drift and a self-loop is added to the second concept. At T_{501} and T_{751}, these concepts reoccur suddenly, and the algorithm handles them properly and updates the graphical pool links accordingly.

In Figure 6.3, the experiment is repeated on the Hyperplane dataset with both gradual and sudden drift. This experiment tests the effectiveness of the merging method as well as the statistical similarity test. Again, the first snapshot is taken at T_1 with the only concept in the pool. The gradual drift moves the hyperplane slowly up to T_{125}, and the similarity test detects 7 different concepts until then. By the recurrence of one previous concept at T_{126} (almost the same as the concept at T_{50}), a new concept is added temporarily, however, vanishes and is merged in the next steps (look at the backward link between concept 6 and 5 in T_{250}). The gradual movement of the hyperplane continues, and the graphical model of the pool is updated until T_{250}, when it reaches the inverse concept from the beginning. At T_{251}, a sudden drift causes a return to the first concept, and we see from the graph

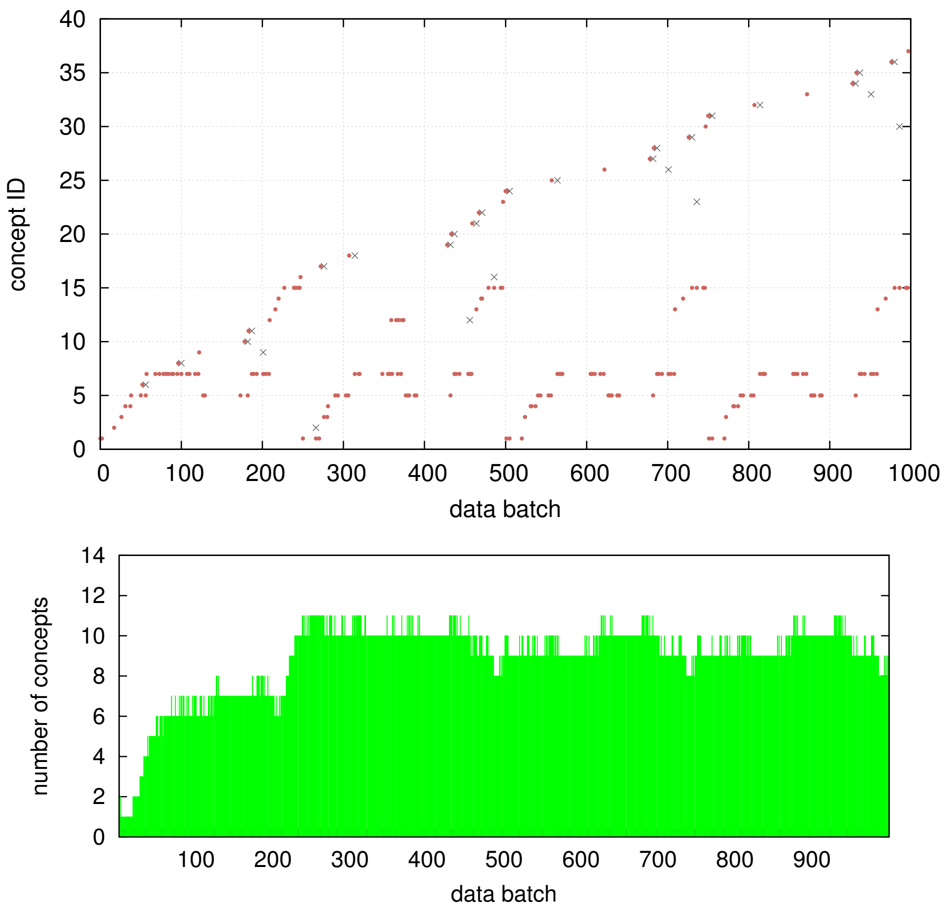


Figure 6.4: The upper plot represents the concepts' life cycles on the HyperPlane data with gradual and sudden drifts. The X-axis indicates data batches of size 1000, and the Y-axis is the unique concept ID. Red dots show the creation or reappearance or the merge point of another concept with the corresponding concept. The black cross denotes the disappearance of a concept by merging with another concept. The lower plot shows the histogram of pool sizes at each data batch.

of the pool that the algorithm finds the correct concept and links the last one to the first concept. The top plot in Figure 6.4 shows the life cycle of each concept. The Y-axis indicates the unique ID of a concept and the first red dot for each value in Y-axis shows the times the concept was created first. The other red dots show the times another concept was merged with this concept or the concept has reappeared. The black cross shows the time a concept vanishes, if ever. Apart from the short-term concepts, we can see that a pattern in the concept usage is repeated for every 250 batches. These short-term concepts can be viewed as a short-term memory to recognize a concept more precisely. The second plot in Figure 6.4 shows the number of concepts in the pool at each time slice. This shows that the number of short-term concepts does not grow arbitrarily.

On the next experiment, we evaluate the effectiveness of using the covariance representation versus variance (Figure 6.5). To test the impact of using covariance, we generated a two-class random multivariate normal distribution with three features. To have control over the covariance matrix values, we used the vine method [109] to generate a random correlation matrix in a specified interval. By generating random feature variances in a definite interval, we can convert the random correlation matrix to a covariance one. To include concept drifts, we rotate the covariance matrix by a random angle and shift the mean vector by a small vector. In the following experiment, we have generated 10 concepts and for each $10,000$ instances. In total, we have generated $1,000,000$ instances from these concepts. The experiment is repeated for three different correlation intervals: low correlation ($[0, 0.2]$), medium correlation ($[0.4, 0.6]$) and high correlation ($[0.8, 1]$). Figure 6.5 illustrates the life cycle of concepts that appear in the pool with the batch size of 1000. As we can see, the stronger the correlation between features are, the more short-term concepts appear in the pool if we ignore the covariance modeling and only consider the variance of features. This is evidence for the benefit of covariance in better-approximating data.

6.2.4 Performance analysis

In this section, we compare GraphPool to the other methods in the literature in terms of classification performance as well as time and resource complexity.

Classification results analysis

Figure 6.6 shows the accuracy of all the comparison methods on datasets. On synthetic datasets, plots illustrate batch-wise accuracy, but due to many drifts in real-world datasets, the batch-wise accuracy fluctuates a lot. Thus, we have plotted the accumulative accuracy for real-world datasets. We have also excluded DWM from the batch-wise accuracy plots, because of too

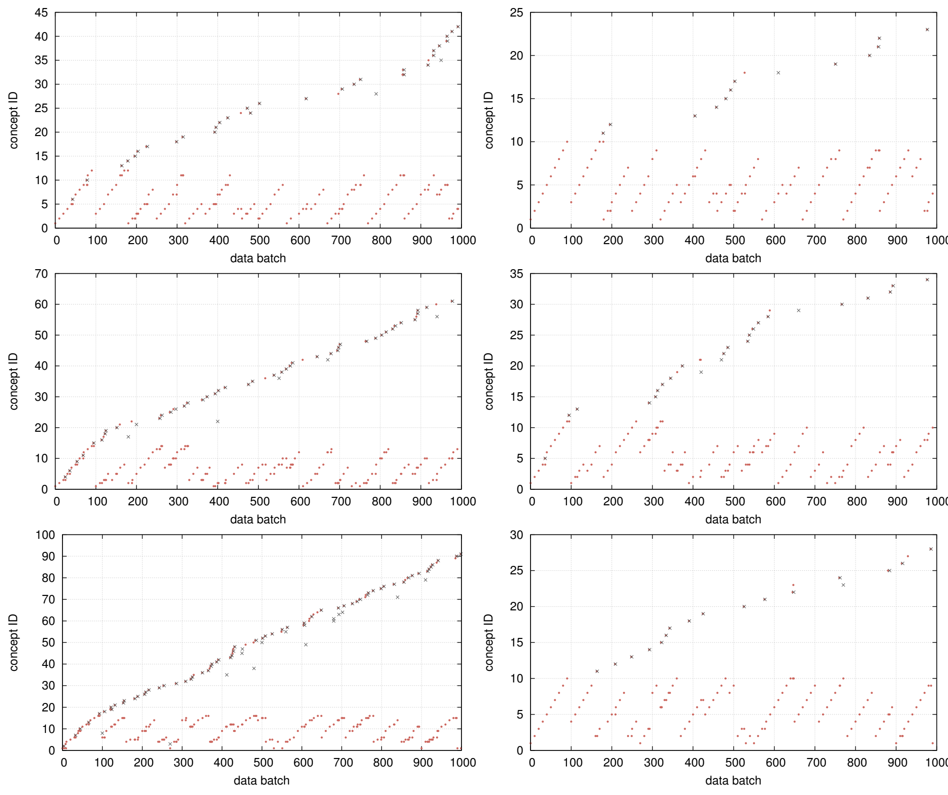


Figure 6.5: Concepts' life cycles of GraphPool variations: Left column illustrates variance representation life cycles and right column belongs to the covariance representation on the same dataset. Correlation between features differs in each row from $[0, 0.2]$, $[0.4, 0.6]$, and $[0.8, 1]$ (top to bottom), respectively. The batch size is 1000.

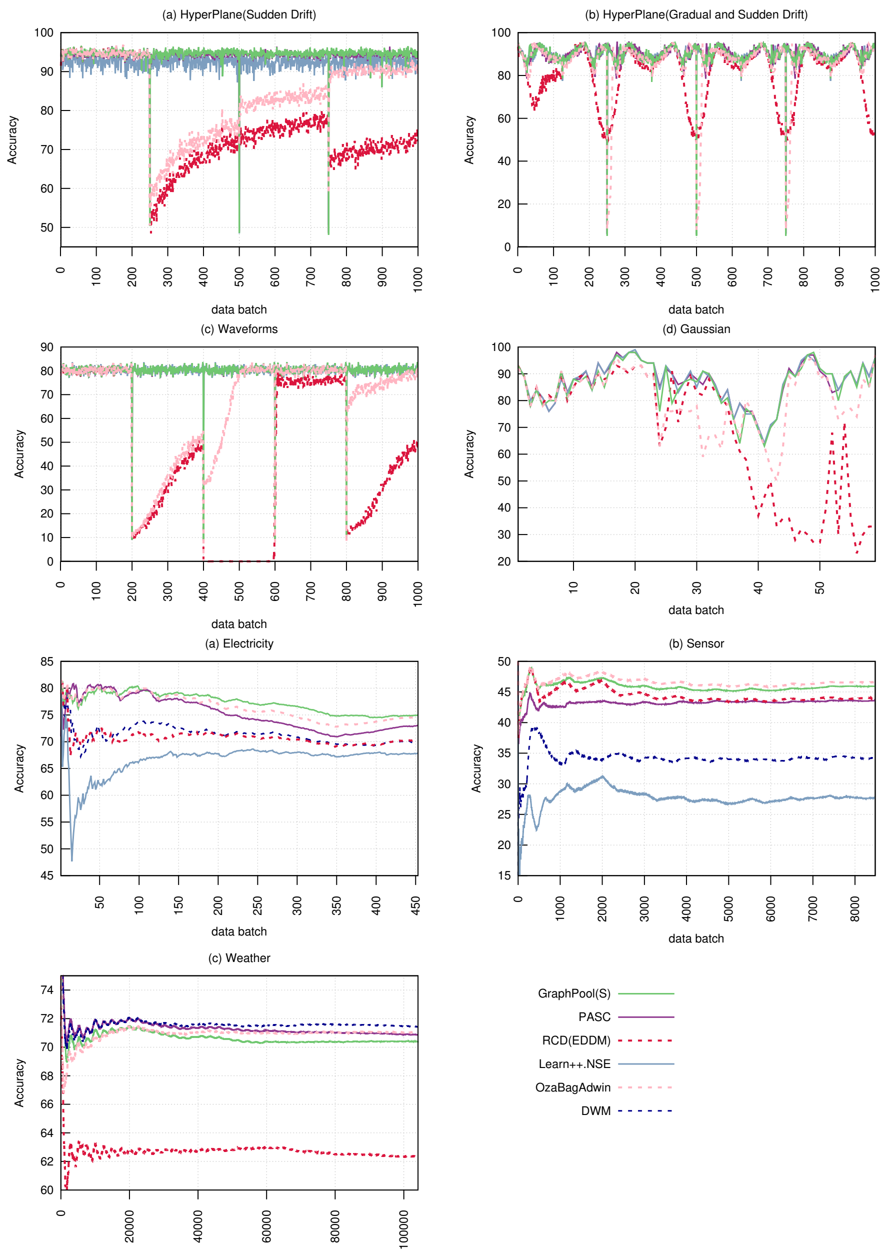


Figure 6.6: Accuracy of different algorithms on synthetic and real-world datasets. Synthetic dataset plots show batch-wise accuracy, while Electricity, Sensor and Weather plots illustrate accumulative accuracy over seen batches. Each batch size is set to 100, except for HyperPlane and Waveform, where the batch size is set to 1000.

many fluctuations which made the plots unreadable, but the results are reported in Table 6.2. As we can see, GraphPool has the best or nearly the best accuracy on all tested datasets, synthetic and real-world. On synthetic datasets, one can clearly notice that GraphPool recovers very quickly after each sudden drift. This is not the case for RCD and OzaBagAdwin. Methods with a fixed-size ensemble of classifiers, e.g., OzaBagAdwin, need more time to forget the previous concepts and converge to the new one.

Although Learn++.NSE performs quite similar to GraphPool on synthetic datasets, on real-world datasets its performance is much worse, and it does not even finish the experiment successfully on the Weather dataset[9]. Our method performs better than PASC on many of the datasets, however, on some real-world datasets, it performs slightly worse in terms of accuracy. Looking at the other performance measures, one can see even if the accuracy is slightly worse, the recall or F-measure is still better (e.g., on Weather in Table 6.2) and the method is still comparable. We need to recall that PASC faces the challenge of setting a domain-specific parameter, which has a substantial effect on the performance, but this issue is resolved in GraphPool. Also, PASC does not extract the concepts' connections, and finding repetitive patterns in drifts is not possible.

Table 6.2 reports the average accuracy, precision, recall, F-measure, and the running time of all methods. The average of precision and recall is calculated over all batches of data and for all the classes, and the average F-measure is computed as $\frac{2P_{avg}R_{avg}}{P_{avg}+R_{avg}}$.

We should remark that the Electricity dataset has been used as a popular benchmark in testing many of adaptive classifiers, but a recent study shows that there is a strong temporal dependency in this dataset [19]: If we use a naïve baseline that ignores all the features and predicts the next class label only based on the current label, the accuracy on Electricity dataset will reach to 85.3% (Table 6.3). However, we can see that on synthetic datasets where there is no temporal dependency, the accuracy of the so-called no-change classifier is the same as a random classifier. None of the methods of comparison in our previous experiments as well as our proposed method consider temporal dependencies, that is, they all assume instances are independent. If one is interested to include temporal dependencies, a simple wrapper method (SWT) has been proposed in the literature [19] and it still is an open problem.

Time and resource complexity

The last column in the table shows the pool size in GraphPool after processing all the batches. For a better visualization of time complexity, we have plotted a heat map, which represents the log ratio of each method's time,

[9]The experiment was finished for only 10% of the data after 72 hours.

Table 6.2: Performance Comparison of GraphPool to other methods over synthetic and real datasets. Accuracy (Acc), Precision (P), Recall (R) are averaged over all the batches of data. F-measure is calculated based on average P and R. The pool size is the final number of concepts in the pool after processing the whole data.

Dataset	Methods	Acc.	P	R	F_1	Time(s)	Pool size
Hyperplane Sudden drift	GraphPool	**94.52**	**94.51**	**94.52**	**94.51**	**2.93**	2
	RCD	76.37	76.36	76.37	76.36	54.82	
	OzaBagAdwin	84.44	84.45	84.44	84.45	15.67	
	L++.NSE	92.33	92.38	92.33	92.35	757.49	
	DWM	91.47	90.15	91.47	90.81	64.30	
	PASC	94.36	94.36	94.36	94.36	3.72	
Hyperplane Gradual & Sudden drift	GraphPool	**90.06**	**90.08**	**90.07**	**90.07**	**2.91**	9
	RCD	80.02	80.03	80.02	80.01	61.06	
	OzaBagAdwin	86.18	86.19	86.18	86.17	139.30	
	L++.NSE	90.03	90.04	90.03	90.02	740.10	
	DWM	89.29	83.81	86.25	85.01	63.76	
	PASC	89.83	89.83	89.82	89.83	3.69	
Waveform	GraphPool	**80.11**	**83.45**	**80.16**	**81.77**	**13.69**	3
	RCD	43.22	44.54	43.23	43.87	221.46	
	OzaBagAdwin	66.62	68.55	66.64	67.58	99.13	
	L++.NSE	80.06	83.33	80.10	81.68	6469.55	
	DWM	69.68	67.36	69.72	68.52	789.21	
	PASC	80.00	83.39	80.05	81.69	25.85	
Gaussian	GraphPool	85.93	69.27	68.87	69.07	0.17	17
	RCD	66.96	55.11	54.54	54.82	0.48	
	OzaBagAdwin	80.51	65.02	64.16	64.59	1.29	
	L++.NSE	**86.98**	**69.94**	**69.71**	**69.82**	0.75	
	DWM	76.81	61.50	61.32	61.41	0.17	
	PASC	86.86	69.88	69.66	69.77	**0.15**	
Electricity	GraphPool	**75.23**	**77.15**	72.35	**74.67**	0.64	8
	RCD	70.42	73.84	71.22	72.51	5.48	
	OzaBagAdwin	74.63	75.00	71.11	73.00	12.25	
	L++.NSE	67.83	72.63	**72.78**	72.70	54.08	
	DWM	69.94	67.31	68.91	68.10	5.35	
	PASC	73.02	72.95	68.53	70.67	**0.56**	
Sensor	GraphPool	45.93	35.01	**36.79**	35.88	35.40	268
	RCD	43.90	28.77	34.33	31.31	25.22	
	OzaBagAdwin	**46.54**	31.24	35.61	32.28	247.07	
	L++.NSE	27.64	22.26	22.46	22.36	12573.0	
	DWM	34.29	21.47	35.01	26.61	159.88	
	PASC	43.58	**36.07**	36.69	**36.38**	**6.47**	
Weather	GraphPool	70.38	65.14	**66.63**	**65.88**	**103.87**	53
	RCD	62.32	60.28	61.60	60.93	595.86	
	OzaBagAdwin	71.04	**65.49**	65.21	65.34	3429.68	
	L++.NSE	-	-	-	-	-	
	DWM	**71.42**	56.76	61.82	59.18	2362.72	
	PASC	70.85	65.09	66.26	65.67	165.94	

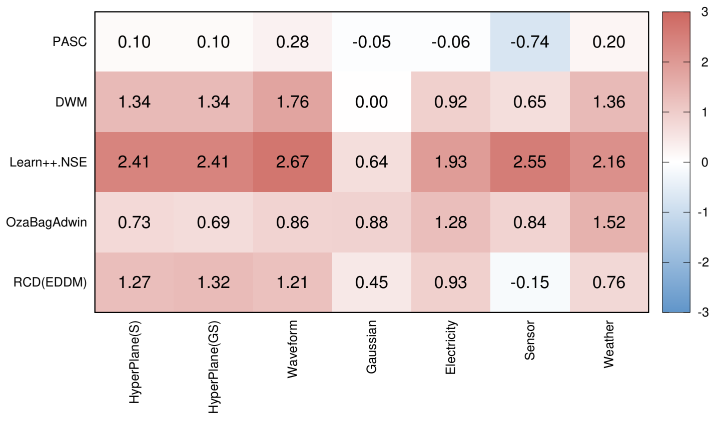


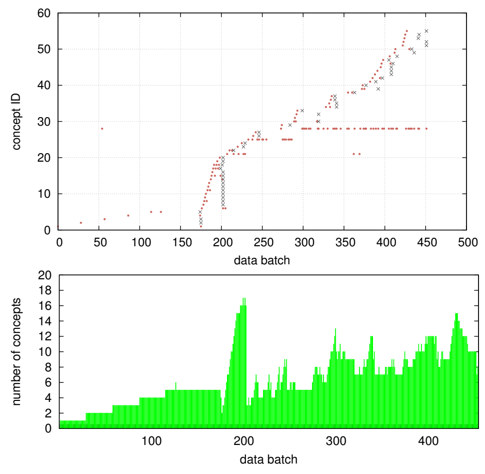
Figure 6.7: Runtime comparison of different algorithms to GraphPool in terms of log ratio.

Table 6.3: Accuracy of no-change naive baseline classifier.

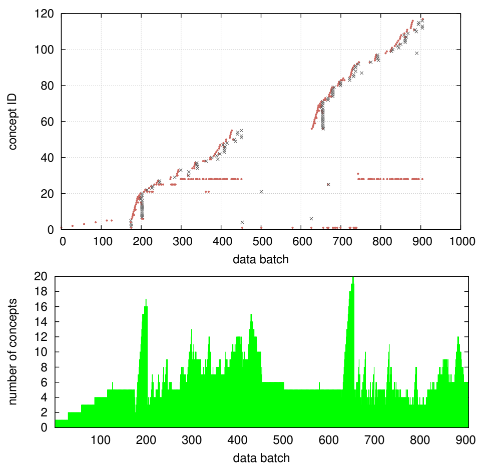
	Hyperplane(S)	Hyperplane(GS)	Waveform	Gaussian	Electricity	Sensor	Weather
Accuracy	49.9	49.9	33.3	38.9	85.3	39.5	67.9

i.e. $\log_{10} \frac{t_{method}}{t_{GraphPool}}$ (Figure 6.7). This heat map illustrates the difference in the order of time needed to finish each method in comparison to GraphPool for each dataset. We observe that except PASC, all the methods need much more time than GraphPool. That means the cost of removing useless concepts in methods like Learn++.NSE and DWM is high, whereas the merging of concepts in GraphPool is not that costly. PASC performs slightly slower on most of the datasets. However, on Sensor, where our method detects many concepts, it performs faster because of the limit on the number of classifiers.

To study the pool's expansion in more detail, we have plotted the concepts' life cycles and the pool size for Electricity in the same way as for Hyperplane (Figure 6.8). The left plots illustrate the results for one repeat of the dataset and the right plots belong to two repeats of the same data. We can see that the pattern for the short-term concepts is almost the same on the repeat of data, except that the first building steps (on the first 100 batches) are not needed when repeating (Figure 6.8(b) batch 500 to 600). Nonetheless, the pool size never goes beyond 20 at any time slice. We expect by receiving enough instances from each concept, the learners and the conceptual vectors converge. Thus, if no drifts occur, the number of concepts in the pool should stay the same, and there will be no short-term concepts anymore.



(a)



(b)

Figure 6.8: (a) Electricity data, (b) two repeats of Electricity data. The upper plots represent the concepts' life cycles. The X-axis indicates data batches of size 100, and the Y-axis is the unique concept ID. Red dots show the creation point or when the concept has reappeared or when another concept is merged with the concept. A black cross denotes the disappearance of a concept by merging with another concept. The lower plots show the histogram of the pool size at each data batch.

6.2.5 Comparisons of different variations of the GraphPool approach

GraphPool consists of different components: the conceptual representation, the statistical similarity measure, and the graphical model of the pool. In this section, we go through more experiments on each part and test some possible alternatives and their impact on the performance of the framework.

Label prediction: single classifier vs. majority vote

As discussed above, GraphPool classifies test data using one state or all the states in the neighborhood of the current state. In the first comparison, we test the effect of labeling an instance by the weighted majority vote instead of a single classifier. Table 6.4 indicates that using the weighted majority vote in most cases degrades the performance. On some datasets (e.g., Waveform and Weather), the self-loop to a concept has much larger weight, such that the other concepts will not have any effect in the decisions. Only on Electricity, the weighted majority improves performance.

Table 6.4: Comparison of GraphPool variations on different datasets: single classifier (SC) vs. majority vote (MV).

	Acc.		P		R		F_1	
Dataset	SC	MV	SC	MV	SC	MV	SC	MV
Hyperplane(S)	**94.52**	94.48	**94.51**	94.49	**94.52**	94.49	**94.51**	94.49
Hyperplane(GS)	**90.06**	90.00	**90.08**	90.01	**90.07**	90.00	**90.07**	90.01
Waveform	80.11	80.11	83.45	83.45	80.16	80.16	81.77	81.77
Gaussian	85.93	85.93	**69.27**	69.26	**68.87**	68.86	**69.07**	69.06
Electricity	75.23	**75.25**	77.15	**77.25**	72.35	**72.37**	74.67	**74.73**
Sensor	**45.93**	45.92	35.01	**35.02**	36.79	36.79	35.87	**35.88**
Weather	70.38	70.38	65.14	65.14	66.63	66.63	65.88	65.88

Table 6.5: Comparison of GraphPool variations on different datasets: statistical likelihood test (SL) vs. classifier similarity (CS).

	Acc.		P		R		F_1		Time (s)		Pool size	
Dataset	SL	CS	SL	CS	SL	CS	SL	CS	SL	CS	SL	CS
Hyperplane(S)	94.52	**94.59**	94.51	**94.59**	94.52	**94.60**	94.51	**94.59**	**2.93**	3.73	2	2
Hyperplane(GS)	**90.06**	88.44	**90.08**	88.45	**90.07**	88.43	**90.07**	88.44	**2.91**	6.69	9	5
Waveform	**80.11**	80.01	**83.45**	83.40	**80.16**	80.06	**81.77**	81.69	**13.69**	48.88	3	3
Gaussian	**85.93**	85.22	**69.27**	68.74	**68.87**	68.36	**69.07**	68.54	**0.17**	0.32	17	16
Electricity	**75.23**	74.13	**77.15**	76.49	**72.35**	72.17	**74.67**	74.26	**0.64**	1.47	8	5
Sensor	**45.93**	42.59	35.01	**35.90**	**36.79**	36.54	35.88	**36.22**	**35.40**	987.5	268	387
Weather	**70.38**	70.19	**65.14**	64.52	**66.63**	65.30	**65.88**	64.90	**103.9**	2132.3	53	36

Conceptual equivalence: statistical likelihood test vs. classifier similarity

Yang *et al.* used another approach to measure the concept similarity [201]. Instead of applying a test on data summaries, they use a score on how two classifiers share the same "opinion". Thus, for every batch, a new classifier is trained and compared to all the classifiers in the pool. If they (the new classifier and the one in the pool) classify an instance in the same class, whether correct or incorrect, the score is increased by 1; otherwise it is decreased by 1. The final score is in the $[-1, 1]$ interval since we compute the total score on the new batch and divide it by the batch size. Table 6.5 compares this approach to GraphPool's approach. A similarity threshold needs to be set for the classifier similarity approach. We have set it to 0.8 on synthetic datasets. On real-world datasets, there is more noise; therefore, the threshold is set to 0.6. Experimental results show that on most of the datasets the statistical test outperforms the classifiers' similarity. Furthermore, the statistical test takes much less time than checking all the classifiers for each batch of data.

6.2.6 Sensitivity analysis

Besides the batch size and the base classifier, which are the general parameters of any learning framework, our method has another parameter to be set: the confidence level of the similarity test. In this section, we test the effect of different p-values as well as the batch size on the performance and pool size of GraphPool.

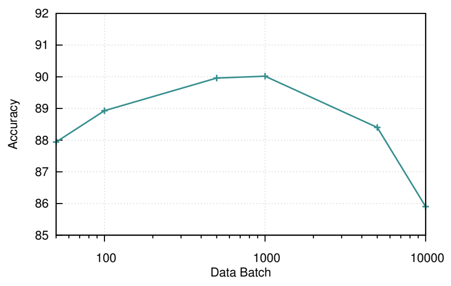
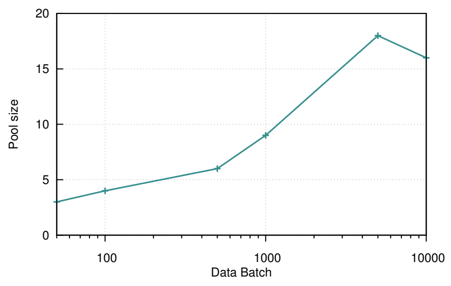


Figure 6.9: Sensitivity analysis on Hyperplane with gradual and sudden drifts: effect of batch size on the average accuracy and pool size.

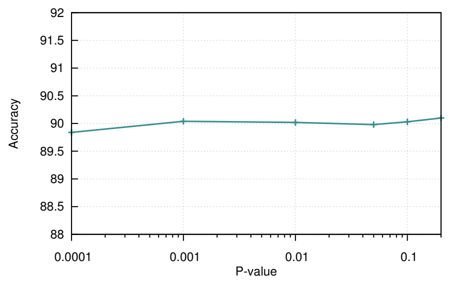
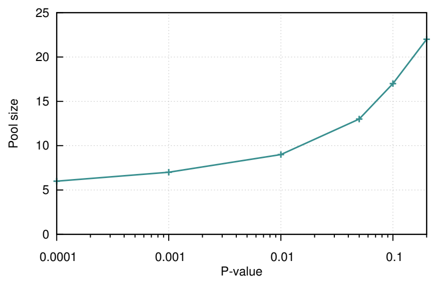


Figure 6.10: Sensitivity analysis on Hyperplane with gradual and sudden drifts: effect of p-value on the accuracy and pool size.

Change of batch size

Figure 6.9 shows the effect of different batch sizes on the average accuracy and the pool size of Hyperplane with gradual and sudden drifts. We have repeated the experiment for batches of size 50, 100, 500, 1000, 5000 and 10,000. The X-axis is on a logarithmic scale. The accuracy plot indicates that by increasing the batch size up to 1000, the accuracy improves. However, larger batches will not help. This was predictable, as the fewer instances we provide to a classifier, the weaker it performs. On the other hand, enlarging the batch size may violate the stationarity assumption of data batches and harm performance. This is also shown on the pool size plot: the larger the batch is, the more concepts are added to the pool.

Change of confidence level in statistical test

To test the effect of the confidence level, we have repeated the experiment for different p-values: 0.0001, 0.001, 0.01, 0.05, 0.1 and 0.2. Figure 6.10 shows the accuracy and the pool size for each of the values on the Hyperplane dataset with gradual and sudden drifts. The X-axis is again on the logarithmic scale. We observe that the change in the p-value does not have a remarkable effect on the accuracy, however, the number of concepts in the pool increases for larger p-values. This is the result of accepting new concepts even when the test is not certain, which results in a larger pool of classifiers.

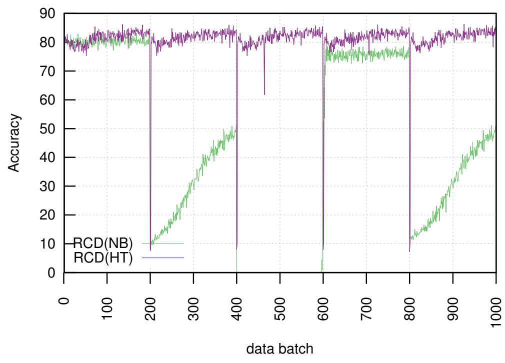


Figure 6.11: Batch-wise accuracy comparison of Hoeffding tree(HT) and Naïve Bayes(NB) in the RCD framework on Waveform. X-axis indicates data batches of size 1000.

6.2.7 Future directions

Merging classifiers might be an easy task for generative classifiers where the underlying distributions are kept (e.g., Naïve Bayes), however, it is more complicated for discriminative classifiers that learn a direct map from input space to the class labels (e.g., Hoeffding tree). In this chapter, we used a simple generative classifier, Naïve Bayes, in our experiments. While comparing GraphPool to the literature in Section 6.2.4, we observed that RCD does not recover from sudden drifts too well. We tried to change different parameters and noticed that if the base learner is exchanged by a stronger classifier, e.g., a Hoeffding tree in this case, the performance of the method improves a lot. An extreme case can be seen in Figure 6.11, where a comparison of the batch-wise accuracy of RCD variations, one with Hoeffding tree and the other one with Naïve Bayes as base classifiers, is shown. Although this is an extreme case and the performance change is not always like that, this can open a new direction of research to work on merging methods for discriminative classifiers, like Hoeffding trees, as well as other generative classifiers. An alternative approach to the merging of classifiers is to maintain an ensemble of trees for each concept. However, this again requires some forgetting mechanism whenever the number of classifiers in the ensemble reaches the maximum limit.

The framework proposed in this chapter only contains merging concepts, however, there may be some cases where an existing concept starts to become heterogeneous gradually or even all of a sudden. In these cases, having a mechanism to check whether a concept should be split and how the splitting should be done would be beneficial. Therefore, a possible future direction could be developing methods for splitting concepts.

6.3 Conclusions

In this chapter, we introduced a new classification framework called GraphPool, which deals with recurrent concepts using graphical models. Recurring concepts add even more complexity to the problem of classifying concept drifting data streams. While existing approaches dealing with recurring concepts focus on only learning a pool of classifiers, GraphPool aims to solve their limitation by keeping the transition probabilities among the concepts in a graphical model. This graphical modeling gives the opportunity to model the behavior of an environment, useful, for instance, in recommender systems or forecasting systems. In addition, to enhance the generalization of concepts and classifiers, GraphPool benefits from a merging procedure. The merging procedure helps to combine concepts even in the absence of the original data. Concepts which are statistically similar to each other and were yet distinguished as different concepts due to the lack of enough data are merged together. GraphPool also expands the idea of conceptual representations by considering the correlations in feature space. Experimental evaluations show that this extension improves performance, particularly on real-world datasets.

In comprehensive experiments, GraphPool has shown to perform well in terms of accuracy and running times compared to other approaches in the literature. However, there is still room for improvement. One direction concerns the merging of classifiers. In this chapter, we used Naïve Bayes because of its ability to easily merge two learners in the absence of their training data. One promising research direction would be to work on the merging of stronger classifiers online, for instance, of Hoeffding trees.

Chapter 7

Modeling Multi-Label Recurrence of Data Streams

Most of the existing data stream algorithms assume a single label as the target variable. However, large-scale multi-label classification has become an essential problem with the emergence of real-world applications such as image annotation [187], query recommendation [4], and credit assignment for social bookmarking [144]. For example, in processing massive social media corpora, each news article may belong to several topics/categories. These topics may change, evolve, or even reoccur over time. Our study on German media shows how highlighted topics evolve or reoccur based on the upcoming political events [5]. Classification of multi-label data can be viewed as a generalization of multi-class classification where labels do not exclude each other, and they may have unknown dependencies among each other as well as with the features, which may change over time. Making accurate predictions for such non-stationary multi-label streaming data with the consideration of dependencies among labels and potential drifts is a challenging task. The few existing studies mostly cope with drifts implicitly, and all learn models on the original label space, which requires a lot of time and memory. In this chapter, we propose a graph-based framework that maintains a pool of multi-label concepts with transitions among them and the corresponding multi-label classifiers. As a base classifier, we benefit from our previously developed fast linear label space dimension reduction method, RACE, that transforms the labels into a random encoded space and trains models in the reduced space.

Discovering recurrent concepts in streaming data is a challenge that has been faced only quite recently [66]. Although keeping a pool of classifiers is a common way of dealing with recurrent concepts in single-label data streams, no such study exists for multi-label streams. In this chapter, our goal is to propose, for the first time, a multi-label framework for coping with recurrent drifts in multi-label streams, and at the same time, taking advantage of hidden label correlations to improve multi-label classification performance. We follow and extend the idea of modeling recurrent concepts via graphical models to the multi-label setting. GraphPool, as discussed in Chapter 6, is a successful framework for accurately modeling single-label recurrent concepts by keeping the transitions between the concepts and employing a concept merging mechanism when necessary. Experiments confirm that keeping the transitions between the concepts helps to quickly recover from drifts with periodic behavior. Since each concept is initially based on a small batch of instances, merging the existing concepts of the pool is another effective feature of GraphPool in combining several similar concepts into one more general concept or building a larger concept from its smaller subconcepts. However, to generalize the framework for multi-label streams, we need to address several challenges:

1. Many of the well-established multi-label classifiers scan the training data an arbitrary number of times, and hence, are not efficient for large-scale problems. Processing multi-label streams needs efficient methods in terms of time and space complexity. To solve the issue, we use a fast label space dimension reduction method, RACE, as a multi-label classifier. RACE applies a random compression encoding matrix to the original label set to transform the original label set into a much smaller random space. It then trains single-label classifiers on the compressed space. The least squares solution gives a decoding matrix to map the compressed labels to the original space.

2. GraphPool extracts concept representatives directly from training data. It makes a simplifying assumption on data batches and estimates instances of each class value with a multivariate Gaussian distribution. It then keeps the mean and covariance matrices of the feature space as concept representatives. However, in the multi-label stream setting, this information will be rather sparse due to the large label space. To resolve the issue, instead of extracting concept representatives from data, we represent a multi-label concept with RACE's decoding matrix. Consequently, a comparison of the concept representatives is not possible with a multivariate likelihood test (as it was proposed in the original version of GraphPool) anymore. Therefore, we use a matrix distance measure (e.g., the average Euclidean distance of matrix elements) and apply a user-defined threshold to check if two multi-label concepts are similar.

3. Comparison of large decoding matrices may result in various false positive alarms for drifts, due to the very large space and high data sparsity in a small batch of data. In our multi-label GraphPool framework, we propose an extra module to monitor the performance of the current concept and run the previous step only when there is a notable

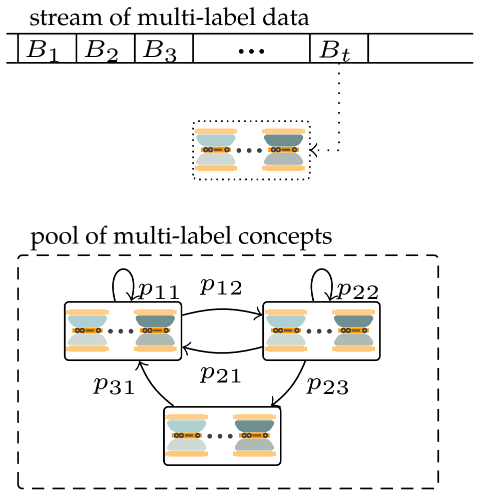


Figure 7.1: A sample snapshot of the proposed framework. Each concept is represented by an ensemble of RACE classifiers

drop in performance. This way, we benefit from a two-stage drift detection module and add more robustness to the signals.

Figure 7.1 represents a sample snapshot of our multi-label GraphPool framework. For a new batch of multi-label data B_t, we check its similarity to the existing concepts in the pool. Therefore, we train a RACE classifier on the data and use its decoding matrix as a concept representative. To have a valid comparison, we initialize the random encoding matrices to the same value for every batch. Furthermore, to reduce the effect of randomness in the encoding matrix of RACE, we train an ensemble of RACE classifiers with different initialization matrices.

7.1 Multi-label GraphPool: A multi-label recurrence framework

Inspired by the idea of modeling recurrent concepts via graphical models in single-label streaming data, we propose a novel framework that models recurrent concepts in a multi-label scenario. Graphical modeling of concepts has shown promising results in quickly detecting recurring concepts and patterns in single-label streams. However, detecting recurrent concepts in the context of multi-label stream processing is much more challenging, and methods like GraphPool, in their current form, are not applicable to the multi-label setting. The first issue is to choose an efficient multi-label updateable classifier as a base classifier for stationary data batches. Very few existing multi-label stream frameworks work in the original label space, and therefore, lack efficiency when the label space is large. To handle this issue, we benefit from RACE as the base multi-label learner. We use a variation of

ALGORITHM 12: Adaptive variation of RACE algorithm

```
Input: $B(t) = (X(t), L(t))$ is the $t^{th}$ batch of data,
       $l$: the original label space size,
       $k$: the reduced label space size
Output: trained model and decoding matrix $\beta_{k \times l}$
1  Generate an orthonormal random matrix $A_0$ of size $l \times k$
   // Initialization step
2  $H_0 = L(0)A_0$
3  $H_0^{r,c} = \begin{cases} 1 & \text{if } H_0^{r,c} \geq 0 \\ 0 & \text{otherwise} \end{cases}$
4  Train a Binary Relevance updateable model on $(X(0), H_0)$
5  Initialize label decoders: $K_0 = ({H_0}^T H_0)^{-1}$ , $\beta_0 = K_0 H_0^T L(0)$
   // Sequential step
6  while more batches of data do
7      $A_{t+1} = {\beta_t}^T$
8      Get pseudo labels by $H_{t+1} = L(t+1)A_{t+1}$
9      $H_{t+1}^{r,c} = \begin{cases} 1 & \text{if } H_{t+1}^{r,c} \geq 0 \\ 0 & \text{otherwise} \end{cases}$
10     Update Binary Relevance model with $(B(t+1), H_{t+1})$
11     Update $K_{t+1}$ and $\beta_{t+1}$ using equation (4.9)
12     $t := t + 1$
```

RACE that employs an adaptive random compression encoding matrix to compress labels into a much smaller random space and then trains single-label classifiers on the compressed space (Algorithm 12). Using a random encoder will accelerate the method considerably compared to the methods which optimize the encoding matrix [193]; and will help in the efficiency of the multi-label classifier especially in the streaming context. A least squares solution maps the encoded labels to the original space and is updated incrementally. Although choosing a random encoding matrix is an effective approach, it adds variance to the resulting models. To reduce such effect of the randomness, we train an ensemble of RACE classifiers for each batch and concept in the model.

Representation of concepts and drift detection are the other challenges in processing multi-label streams. To the best of our knowledge, there is no study on multi-label concept representation. GraphPool extracts representatives directly from data by interpolating instances of each class with a multivariate Gaussian distribution, however, such concept representation in a multi-label scenario adds the assumption of label independence. It may also result in very sparse information when the number of labels increases. Different from GraphPool, we represent multi-label concepts with the decoding matrix of RACE classifiers. This way, we can represent a concept more compactly and still keep the latent information among the labels in the representatives. We then compare the concept representative matrices by the element-wise absolute distance of representatives and apply a pre-defined threshold to find potential drifts. This approach is more efficient than comparing the performance of all existing classifiers in the pool for each batch of data, as proposed in the literature [201]. Element-wise comparison of potentially huge decoding matrices may cause false alarms in detecting drifts as the possible space is massive and the batch sizes are chosen to be small to contain only stationary instances. Hence, we augment our drift detection module by also monitoring the performance of the current state: For each new batch of data, we first train an ensemble of RACE classifiers temporarily. If the performance of the current concept of the pool degrades drastically, we compare the concept representative of the new batch to the other concepts in the pool. If no similarity is found, a new multi-label concept is added to the pool. In case more than one concept is found as similar to the new one, we merge all of them to generalize the multi-label concept. This may happen since the concepts are built on a very reduced number of instances. In the following subsections, we explain the details of our novel multi-label recurrence framework.

The idea of extracting a representative of a concept was first proposed in the Conceptual Clustering and Prediction (CCP) framework [92] by estimating the posterior distribution of each feature given the single-label target class with a normal distribution in every batch of data. However, extracting such information from multi-label batches of data does not give a good estimation, as the number of labels is large, and the data in each label or label combination is very sparse in every batch. Hence, instead of extracting a representative from data, we use the decoding matrix of the RACE classifier as a representative of a multi-label concept. To have a valid comparison among different RACE decoders, we initialize the same encoding matrix A_0 for every new batch. In this way, we can compare the $k \times l$ decoding matrices element-wise by any similarity measure (e.g., absolute difference value). We represent all the concepts in a pool of concepts by a first-order Markov chain. The states contain a concept representative and their corresponding ensemble of RACE classifiers. State S_i is connected to state S_j, if concept j occurs exactly after concept i. The transition weight is the probability of the transition from concept i to concept j when we are at S_i. Keeping a pool of concepts with these state adjacencies and transition weights will help us in a better classification of multi-label recurrent patterns.

The Multi-label GraphPool algorithm (Algorithm 13) starts with an empty pool and a null current state. After receiving the first batch of data, we add the trained ensemble of RACE classifiers as a new multi-label concept to the pool and use it as the current concept for future predictions (line 3– 5). For the next batches of data, we first look at the current state's links and choose the most likely concept in the neighborhood to predict the label of the new batch. When the labels are received, we first observe the

ALGORITHM 13: Multi-label GraphPool algorithm

```
Input: $B(t) = (X(t), L(t))$: instances of $t^{th}$ batch of data,
       $A_{0,i}$: a set of random initialization of encoding matrices where
       $i \in \{1, \dots, s\}$,
       $s$ is the ensemble size
1  Train an ensemble of $s$ RACE classifiers on $B(t)$ with $A_{0,i}$
2  $CR_{t,i} = \beta_{t,i}$
3  if Pool is empty then
4      Add a new vertex $j$ to the Pool
5      Update $CurrentState$ to $j$
   else
6      if accuracy of $CurrentState < \theta$ then
7          Find similar concepts in the pool to the $CR_t$ by taking the majority
             vote over $CR_{t,i}$ (equation (7.1))
8          if number of similar concepts = 0 then
9              Add a new vertex $j$ to the Pool
10             Add $j$ to the neighbors of $CurrentState$ and adjust their weights
11             Update $CurrentState$ to $j$
12         else if number of similar concepts = 1 then
13             Update the existing concept $j$ with $B(t)$
14             if Vertex $j$ is in the neighborhood of $CurrentState$ then
15                 Update the weight of link between $CurrentState$ and $j$
               else
16                 Add $j$ vertex to $CurrentState$ neighborhood
17             Adjust weights of link from $CurrentState$ to its neighbors
18             Update $CurrentState$ to $j$
19         else if number of similar concepts > 1 then
20             Execute the concept merge algorithm (Algorithm 14)
21             Update $CurrentState$ to $C_M$
       else
22         Update $CurrentState$ with $B(t)$
```

level of performance change. If there is a notable drop in the prediction of the new batch, a drift may happen, and the second stage of our multi-label drift detector is activated. For that, we compare the decoding matrix of the trained ensemble of RACE classifiers on the new batch (with the same initialization points as in previous batches) to the existing concepts in the pool (line 6– 7). One straightforward approach may detect a drift if at least one element-wise absolute difference is greater than an acceptable threshold. However, such a method is highly sensitive to small changes and the value of the distance threshold. In our method, we calculate the average distance of a pseudo label, and if it is greater than the threshold, a drift is triggered. Then, we get a majority vote over the ensemble of RACE classifiers:

$$\sum_{i=1}^{s} [\![\sum_{x=1}^{k} [\![\frac{1}{l} \sum_{y=1}^{l} \mid \beta_{t,i}^{(x,y)} - \beta'^{(x,y)}_{j,i} \mid > \theta]\!] > 0]\!] \geq \frac{s}{2}, \tag{7.1}$$

where $[\![.]\!]$ is the indicator function, and θ, l, k, and s are the distance threshold, the label space size, the reduced label space size, and the ensemble size, respectively. Experiments have shown the performance improvement and consistent behavior when RACE is updated by a new data batch more than one iteration. As training and updating RACE is not costly, in our work, we also show each data batch more than once to the learner. Comparing the new concept to the old ones: (1) If no concept in the pool is similar to the new concept, the new concept is added to the pool. The state transition to the current state is created, and the transition link weights of the current state are normalized (line 8– 11). (2) If one concept is similar to the new one, the classifiers and the corresponding concept representative of the existing concept, V_j, are updated. The transition link between the current state and V_j either does not exist yet and we create a new link, or it exists and we only need to update its weight (line 12– 18). (3) More than one concept is similar to the new one (line 19– 21). Following Algorithm 14, we merge all these concepts, including the new batch, into one concept and update the current state to the merged concept.

Merge concepts

If the similarity of the two concepts is greater than a predefined threshold, we apply a merging mechanism. Concept merging consists of several steps (Algorithm 14). First, we merge the transition links of the similar concepts in the Markov model. We update the links in the *neighbor* list of those concepts which have the transition link to the concepts to be merged (lines 4– 9). Then, the *neighbors* list of the new concept is updated (lines 10– 14). Subsequently, we normalize the transition weights of all the concepts involved in this step.

In the next step, we merge the classifiers. As the initial encoding matrix of the ensemble of RACE classifiers for each concept is the same, we will merge the RACE classifiers with the same initialization correspondingly. For that, we first update the base classifiers of RACE. Any classifier that can be merged in the absence of training data can be used. Here, we use a simple generative classifier, the Naïve Bayes Updateable classifier. By keeping posterior statistics of feature occurrence for each label, we can easily update them in the absence of training data. We do not update the decoding matrices, as the similarity measure has detected them to be similar.

ALGORITHM 14: Concept merge algorithm

```
Input: $C_s$: a set of similar concepts in the current $Pool$,
       $B(t)$: current data batch
1  Initialize $C_M$ to the first concept in $C_s$
2  $C_s = C_s - \{C_M\}$
3  for each vertex $v \in C_s$ do
4      for each vertex $u$ in which $v$ is in its neighborhood do
5          if $C_M$ is in $u$'s neighborhood then
6              Update and normalize corresponding $Transition\ weight$
7              Remove $v$ from $u$'s neighborhood
           else
8              Add $C_M$ instead of $v$ to neighbors of $u$
9              Add $u$ to neighbor set of $C_M$
10     for each $w$ in $v$'s neighborhood do
11         if $w$ is in $C_M$'s neighborhood then
12             Update corresponding $Transition\ weight$ from $C_M$ to $w$;
           else
13             Add $w$ to $C_M$ neighborhood with its corresponding weight
14         Remove $v$ from $w$ neighborhood
15 Normalize transition weights from $C_M$
16 Update $C_M$'s learners with all elements of $C_s$ and $B(t)$
17 Remove all vertices of similar concept set from pool
```

Classifying a new batch of data

Let a new batch of data at time t, $X(t)$, be received for classification and $CurrentState$ denote the current state of the pool. To classify the batch, if no concept is in the neighborhood of the $CurrentState$, we take the $CurrentState$ itself as a potential concept for the following batch. Otherwise, if the current concept has connections to the other concepts, we benefit from our previous knowledge and choose the most likely concept for the prediction of the new batch. Each concept is associated with an ensemble of RACE classifiers; hence, we obtain the majority vote of every single RACE prediction on each label.

7.2 Experimental evaluation

Multi-label GraphPool is the first framework for detecting and tracking multi-label recurring concepts. As multi-label stream classification is in its early stages, there are no well-established public datasets to evaluate various multi-label concept drifts. In the following, we first explain a new recurrent multi-label data generator and then report the experimental setting and results on both synthetic and real-world datasets.

ALGORITHM 15: Synthetic multi-label drifting data generator

```
Input: $n_l$: total number of labels,
       $n_f$: number of boolean features,
       $n_i$: number of instances in each concept,
       $A$: set of Dirichlet parameter for each concept
1  for $\alpha \in A$ do
2      $\Phi \sim Dir(\boldsymbol{\alpha}_{n_l \times n_f})$
3  for every concept with $\alpha \in A$ do
4      $global_l = \emptyset$, $global_{l,c} = \emptyset$
5      for $i \in \{1, \dots, n_i\}$ do
           // generate random labels
6          $local_l = \emptyset$, $local_{l,c} = \emptyset$
7          $\boldsymbol{l}_i = [\mathbf{0}]_{1 \times n_l}$
8          for $l \in \{1, \dots, max_l\}$ do
9              $table$ = Sample from $p_l \sim PY(d, \gamma, local_{l,c})$
10             if $table$ is new then
11                 $table_g$ = Sample from $p_g \sim PY(d, \alpha, global_{l,c})$
12                 Update $global_l$ and $global_{l,c}$ with respect to $table_g$
13                 Update $local_l$ and $local_{l,c}$ with respect to $global_l$
14                 $l_{i,t} = 1$
               else
15                 Choose a label from $local_l$ that is assigned to $table$ and update
                     its entry in $local_{l,c}$
16                 $l_{i,t} = 1$
           // generate random features
17         for $w \in \{1, \dots, max_f\}$ do
18             $l_w$ = Choose a label from $\boldsymbol{l}_i$
19             $f_w$ = Sample from distribution $\Phi_\alpha[l_w]$
20             Add $< f_w, l_w >$ to instance $i$ of concept $\alpha$
```

7.2.1 Multi-label concept drifting data generator

Inspired by one standard application of multi-label streaming data, text categorization, we develop a synthetic multi-label document generator (Algorithm 15). We assume that each generated document consists of at most max_w words and belongs to at most max_l categories as its labels. Documents of each concept are drawn from a Dirichlet distribution with randomly chosen parameters (line 1– 2). For each multi-label concept, we choose the global variables (line 4), and for each new sample, we create the local distribution, $local_l$, over labels with the global distribution as a prior (line 6). We sample a label from a Pitman-Yor process with respect to the local distribution of labels, $local_l$ (line 9 and 15– 16); moreover, we assume a global Pitman-Yor distribution of labels over words, $PY(d, \alpha, G_t)$, to draw a label when local sampling chooses a new table (line 10– 14). Then, we sample the max_w words of the document from the Dirichlet distribution of the

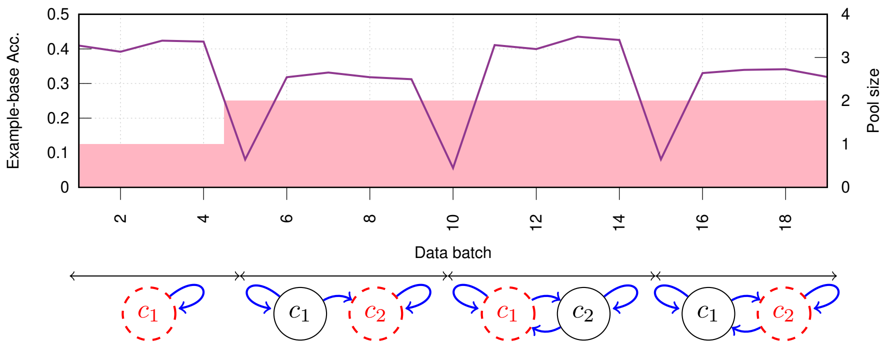


Figure 7.2: Evolution of multi-label classifier pool on the synthetic dataset. The purple line illustrates the batch-wise example-base accuracy, and pink bars show the pool size at each time stamp. Red dashed nodes indicate the current state of the algorithm.

corresponding concept and add both randomly chosen features and labels to the current instance (line 17– 20). We can generate abrupt and recurrent multi-label streams with the proposed generator[1].

7.2.2 Experimental setting

We generate two synthetic datasets with the proposed multi-label recurrent concept generator. *Synthetic 1* consists of two concepts with an α equal to 0.2 and 0.8 and $d = 0.75$. We generate 500 documents from each concept over 100 possible labels and 100 possible words, and resample another 500 documents from each of them respectively to model the recurrence of concepts. We set $max_w = 20$ and $max_l = 20$ for each document. The second synthetic dataset (*Synthetic* 2) is generated from three concepts with $\alpha = \{0.2, 0.8, 0.5\}$ and the concept order of $\{c_1, c_2, c_3, c_2, c_1, c_3, c_1, c_3, c_2\}$, each 1000 documents. The other parameters are set as before. In addition to synthetic datasets, we evaluate our framework on two widely used text benchmarks in multi-label evaluations: Enron (with 1001 features, 53 labels, and 1702 samples) and rcv1v2-subset1 (with 47236 features, 101 labels, and 6000 samples) [179]. Experiments are run in a prequential order. The window size is set to 100, the ensemble size of RACE is 5, each batch is shown to RACE three consecutive times, the performance drop threshold is 0.8, and the dissimilarity threshold of decoding matrices is 0.05. Because of the random selection of the encoding matrix in RACE, the experiments are repeated 10 times and average values are reported.

[1]The source code is available at `https://github.com/kramerlab/Multi-LabelGraphPool`.

Table 7.1: Performance comparison of Multi-label GraphPool, RACE and OBR on different datasets.

Dataset	Method	Ex.-based Acc.	Hamming loss	Macro-avg. F-measure	Micro-avg. F-measure	Pool size
Synthetic 1	Multi-label GraphPool	**0.32**	**0.07**	0.14	0.44	2
	RACE (adaptive classifier)	0.30	**0.07**	0.14	0.44	
	OBR	0.31	0.08	**0.15**	**0.46**	
Synthetic 2	Multi-label GraphPool	**0.31**	**0.07**	**0.16**	**0.44**	3
	RACE (adaptive classifier)	0.25	0.08	0.15	0.37	
	OBR	0.28	0.09	**0.16**	0.42	
enron	Multi-label GraphPool	**0.29**	**0.09**	**0.23**	**0.36**	2
	RACE (adaptive classifier)	0.26	**0.09**	0.21	0.35	
	OBR	0.23	0.18	0.14	0.29	
rcv1v2	Multi-label GraphPool	0.11	**0.17**	0.09	0.10	2
	RACE (adaptive classifier)	0.08	0.18	0.04	0.07	
	OBR	**0.13**	0.43	**0.13**	**0.12**	

7.2.3 Experimental results

We start our experiments with a proof-of-concept experiment on Synthetic 1. Figure 7.2 illustrates how the pool evolves by receiving more batches of multi-label data. At each time step, the $CurrentState$ is represented by a red dashed node. We observe that monitoring the performance of the $CurrentState$ is an effective approach when the data is stationary. Upon occurrence of a sudden drift, however, the performance drops, and this time our effective concept representative comparison method finds the similar/dissimilar multi-label concepts in the pool. If we only monitor the performance, the model is not capable of finding recurrent concepts. If we only check the similarity of concept representatives, we may see extra concepts in the pool as false positives, which are later caught by our merge mechanism. In this experiment, we monitor the example-based accuracy. However, the behavior of other multi-label evaluation measures is almost the same.

In the next step, we extend our experiments to other datasets. We compare our proposed multi-label GraphPool framework to its baseline, RACE, which was developed for stationary multi-label problems and ignores any concept drift/recurrence, and the online version of the well-known Binary Relevance classifier (OBR). Previous studies show that although OBR makes simplifying assumptions on data labels, it still performs better than many other multi-label streaming baselines (e.g., Online Ensemble of Classifier Chains (OECC) or the majority label classifier) and can be used as a strong baseline for comparisons. Table 7.1 presents a performance comparison of these algorithms for different datasets. As stated before, RACE's objective function is to optimize the Hamming loss. Our experiments show that RACE and its successor, multi-label GraphPool, outperform OBR in this regard. On larger datasets with more concepts and drifts (i.e., Synthetic 2 and rcv1v2), Multi-label GraphPool shows its advantage over RACE in terms of all measures (i.e., Hamming loss, Example-based accuracy, and Micro/Macro F-measure). On the other hand, multi-label GraphPool shows its clear effectiveness when there are many drifts and recurrences in the data (i.e., Synthetic 2). RACE and multi-label GraphPool compress the label space by a logarithmic factor, and hence, are less computationally complex than methods dealing with the original label space (e.g., OBR), especially when the label set is large.

7.3 Conclusions

This chapter studied the problem of multi-label recurrence for the first time, and proposed a novel graph-based framework with a two stage drift detection method to handle the challenging problem of multi-label recurring concepts. In order to test our framework under controlled conditions, we proposed a new concept drifting multi-label data generator based on the Pitman-Yor process. Our experimental results show the success of the proposed framework in detecting multi-label drifts and concept recurrences on both synthetic and real-world benchmarks. In the future, we will elaborate on the impact of different types of multi-label drifts by expanding our data generator to a multi-label stream generator with different levels of gradual drifts and broaden our experiments to various domains. Parallelization of our proposed framework is straightforward, hence, we can easily extend our experiments with streams of larger size on a distributed cluster. So far, RACE and multi-label GraphPool compress the label space independent of the feature space. However, feature-aware label space reduction methods can embed more information of the data and possibly become more successful in detecting multi-label drifts. Therefore, we plan to extend our framework to become feature-aware. In our current framework, we assumed all the labels are available for the training data. However, annotating such a large label set is a tedious task. Hence, one could think of ways of handling missing labels in multi-label GraphPool as another possible research direction.

Chapter 8

Conclusion

More than two decades of studying different real-world data streaming problems have yielded a variety of learning solutions. Although there have been extensive studies on single-target class data, not many studies have focused on multi-label stream mining, perhaps due to the recent emergence of their applications. This thesis studied the problem of recurrent concepts in supervised learning for both single-label and multi-label streams. We proposed solutions and frameworks with domain-independent parameters that have the possibility of extracting patterns in periodic environments. In the remainder of this chapter, we review our findings from previous chapters.

8.1 Deep learning for text classification

A large portion of today's data sources is text. The first step to understanding complex documents is modeling sentences and their semantic content. The core of a sentence model is a feature function that captures features for different text units and performs compositions over variable-length sequences. Training a well-performing classifier for a specific task on a single sentence that has a limited amount of contextual data is a challenging task. The recent success of deep neural networks in other domains such as image and speech processing brought enough motivation for their application in natural language processing tasks. In contrast to the traditional bag-of-words model, deep neural networks can preserve the order of words and syntactic structures. Moreover, their significant advantage lies in the removed requirements for feature engineering. In Chapter 3, we investigated an in-depth study of two well-known deep neural network architectures for different text classification problems. Recursive neural tensor networks are a generalization of classic sequence modeling neural networks to tree structures. They integrate semantic content of a sentence (i.e., its parse tree information) to the recursive network architecture with a bottom-up composition strategy. Convolutional neural networks models are another alternative to model sentences. They treat each sentence individually as a bag of n-grams and apply one-dimensional convolution kernels sequentially on word vectors using sliding windows to extract local features. Although recursive neural tensor networks have shown to work well in many cases, they still suffer from the need of intensive manual labeling to overcome the vanishing gradient problem. To avoid that, we proposed two methods to automatically label internal phrases: a rule-based method, which can only be used for sentiment analysis and a convolutional neural network based method for general purposes. Our evaluation results indicated that the CNN based labeling tends to assign a positive or negative polarity to the phrases while the rule-based method classifies many of the phrases as neutral. Our next experiments on an extensive set of standard benchmark datasets by employing automatic labeling demonstrated that the proposed convolutional neural network model outperforms recursive neural tensor network. However, if the manual labeling is available for recursive neural tensor networks, they outperform convolutional neural networks.

8.2 Multi-label stream classification

Chapter 4 studied the problem of multi-label stream classification in stationary environments. Multi-label classification can be viewed as a generalization of multi-class classification where labels do not exclude each other and may have unknown dependencies among each other as well as with the features. Most of the previous multi-label classification methods require a lot of time and memory, which make them infeasible for data stream setting. In Chapter 4, we took the sparsity of label sets and their unknown interdependencies into consideration and proposed *RACE*, an efficient multi-label learning method that reduces the label space by applying random projection. Instead of using iterative numeric solutions like deep neural networks to find a mapping between the original label space and the compressed one, *RACE* finds the mapping function analytically based on the least squares solution. This way, besides providing a fast approach for compression, it does not have many control parameters to be set. Extensive experiments showed its advantage over existing methods in terms of time complexity and accuracy, F-measure, and the Hamming loss.

8.3 Modeling of recurrent single-label streams

From Chapter 5 to 7, we focused on the problem of recurrent concepts. Instead of incrementally updating a classifier, existing approaches, which handle recurrent concepts, maintain a pool of concepts/classifiers and use that pool for future classifications to reduce the error. Chapter 5 focused on single-label streams. It proposed the *PASC* framework by extending *CCP*, one of the successful existing frameworks for recurrent concepts. *CCP* uses domain-specific parameters whose correct choice affects its performance considerably. The pool size is fixed, and after reaching the limit, the most similar classifier is updated regardless of its level of similarity. This can distract a classifier from learning only one concept. In *PASC*, we present a merging procedure that finds the most similar concepts in the pool and merges them. It also improves the similarity measure in such a way that it is less domain-specific. Experiments showed the effectiveness of the proposed modifications, especially on larger datasets with more classes. Studies on the effect of different prediction strategies showed better results in favor of weighted classifier predictions.

Chapter 6 continued the problem of recurrent concepts for single-label streams, but this time by proposing a new first-order Markov chain framework to model the connection of concepts in the pool, called *GraphPool*. *GraphPool*, like its successors *PASC* and *CCP*, makes a simplifying assumption on data batches and estimates instances of each class value with a Gaussian distribution but this time a multivariate Gaussian distribution. To remove the domain dependency of the similarity measure, *GraphPool* applies a likelihood statistical test. It also benefits from an effective merging strategy to manage the pool size. The merging procedure helps to combine concepts even in the absence of the original data. Comprehensive experiments showed that *GraphPool* performs well in terms of accuracy and running times compared to other approaches in the literature. Experiments on synthetic data indicated its perfect ability of modeling concepts and transitions.

8.4 Modeling of recurrent multi-label streams

Although keeping a pool of classifiers is a common way of dealing with recurrent concepts in single-label data streams, no such study exists for multi-label streams. Hence, Chapter 7 extended the *GraphPool* framework to multi-label stream setting. For that, we integrated our efficient multi-label stream classifier, *RACE*, into the successful *GraphPool* framework. However, this combination was by no means trivial. We needed to come up with new ways of concept representation and concept comparisons for multi-label data. *GraphPool* extracts concept representatives directly from training data, however, this information will be rather sparse in multi-label streams due to the large label space. Therefore, *Multi-label GraphPool* uses RACE's decoding matrix as a concept representative and a matrix distance measure (e.g., the average Euclidean distance of matrix elements) to find the similarity of concepts. To reduce false positive alarms for multi-label drifts, resulting from the very large label space, *Multi-label GraphPool* benefits from a separate performance monitoring module as a second stage of the drift detection module. Experiments on multi-label text data represented promising results of the proposed framework in detecting multi-label drifts and concept recurrences.

8.5 Outlook

This thesis proposed methods to merge classifiers for simple generative classifiers. This can be an easy task as estimations of the underlying distributions are kept in generative models. However, there are well-established discriminative classifiers, which perform well on streaming data (e.g., Hoeffding trees), and merging them is not straightforward. Previous studies approached the problem for decision trees by either using simple cases of only one variable [12], or converting decision trees to a set of rules [69, 8], or approximating data with histograms [67]. These solutions are not optimal yet. They either suffer from high computational complexity or inadequate performance. Hence, one future research direction could be proposing new efficient ways to combine discriminative classifiers, especially for Hoeffding trees.

GraphPool and *Multi-label GraphPool* frameworks only consider merging concepts, however, there may be some cases where an existing concept starts to become heterogeneous gradually or even all of a sudden. In these cases, having a mechanism to check whether a concept is diverging and how to split it is beneficial. Therefore, another possible future research can focus on developing methods for splitting concepts in the absence of their training data.

So far, *RACE* and *multi-label GraphPool* compress the label space independent of the feature space. Although feature-aware methods are more computationally complex, feature-aware variations of other label space reduction methods have improved the performance of their predecessors [33, 113, 111]. Extending *RACE* to become feature-aware may lead to embedding more information of the data and possibly helps in better detection of multi-label drifts. Moreover, in our current framework, we assumed that all the labels are available for the training data. Nevertheless, annotating such a large label set is a tedious task. Hence, one could think of ways of handling missing labels in *RACE* and *multi-label GraphPool* as another possible research direction.

Finally, this thesis focused on extracting concept representations and classifiers for every single batch of data. However, these summaries are not always the best summaries of data. For some batches, keeping outliers or density estimations would be more adequate. Inspired by recent advances in deep reinforcement learning in AI problems [118, 139], one can view this problem as that of intelligent resource allocation from experience.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Iris Adä and Michael Berthold. "EVE: a framework for event detection". In: *Evolving systems* 4.1 (2013), pp. 61–70.

[2] Charu Aggarwal. *Data classification: Algorithms and applications*. CRC Press, 2014.

[3] Charu Aggarwal, Jiawei Han, Jianyong Wang, and Philip Yu. "On demand classification of data streams". In: *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2004, pp. 503–508.

[4] Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. "Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages". In: *Proceedings of the 22nd international conference on World Wide Web (WWW)*. 2013, pp. 13–24.

[5] Zahra Ahmadi, Sophie Burkhardt, and Stefan Kramer. "Online Topic Modeling: Keeping Track of News Topics for Social Good". In: *2nd Workshop on Data Science for Social Good in European Conference on Principles of Data Mining and Knowledge Discovery (SoGood)*. 2017, pp. 1–4.

[6] Nir Ailon and Bernard Chazelle. "Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform". In: *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*. 2006, pp. 557–563.

[7] Theodore Wilbur Anderson. *An introduction to multivariate statistical analysis*. Wiley New York, 2003, pp. 411–454.

[8] Artur Andrzejak, Felix Langner, and Silvestre Zabala. "Interpretable models from distributed data via merging of decision trees". In: *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*. 2013, pp. 1–9.

[9] Abad Miguel Ángel, Gomes João Bartolo, and Menasalvas Ernestina. "Predicting recurring concepts on data-streams by means of a meta-model and a fuzzy similarity function". In: *Expert Systems with Applications* 46 (2016), pp. 87–105.

[10] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, Ricard Gavalda, and Rafael Morales-Bueno. "Early drift detection method". In: *Proceedings of the 4th International Workshop on Knowledge Discovery from Data Streams*. Vol. 6. 2006, pp. 77–86.

[11] Krishnakumar Balasubramanian and Guy Lebanon. "The landmark selection method for multiple output prediction". In: *Proceedings of the 29th International Conference on Machine Learning (ICML)*. 2012, pp. 283–290.

[12] Yael Ben-Haim and Elad Tom-Tov. "A streaming parallel decision tree algorithm". In: *Journal of Machine Learning Research* 11.Feb (2010), pp. 849–872.

[13] Yoshua Bengio and Paolo Frasconi. "Input-output HMMs for sequence processing". In: *IEEE Transactions on Neural Networks* 7.5 (1996), pp. 1231–1249.

[14] Wei Bi and James Tin-Yau Kwok. "Efficient multi-label classification with many labels". In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*. Vol. 28. 2013, pp. 405–413.

[15] Albert Bifet and Ricard Gavalda. "Kalman filters and adaptive windows for learning in data streams". In: *Proceedings of the International Conference on Discovery Science (DS)*. 2006, pp. 29–40.

[16] Albert Bifet and Ricard Gavalda. "Learning from time-Changing data with adaptive windowing". In: *Proceedings of the SIAM International Conference on Data Mining (SDM)*. Vol. 7. 2007, pp. 443–448.

[17] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. "Moa: Massive online analysis". In: *The Journal of Machine Learning Research* 11 (2010), pp. 1601–1604.

[18] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldá. "New ensemble methods for evolving data streams". In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2009, pp. 139–148.

[19] Albert Bifet, Jesse Read, Indre Zliobaite, Bernhard Pfahringer, and Geoff Holmes. "Pitfalls in benchmarking data stream classification and how to avoid them". In: *Proceedings of Joint European Conference on Machine Learning and Knowledge Discovery in Databases (PKDD)*. 2013, pp. 465–479.

[20] Hanen Borchani, Pedro Larrañaga, João Gama, and Concha Bielza. "Mining multi-dimensional concept-drifting data streams using Bayesian network classifiers". In: *Intelligent Data Analysis* 20.2 (2016), pp. 257–280.

[21] Hanen Borchani, Ana Martínez, Andrés Masegosa, Helge Langseth, Thomas Nielsen, Antonio Salmerón, Antonio Fernández, Anders Madsen, and Ramón Sáez. "Modeling concept drift: A probabilistic graphical model based approach". In: *Proceedings of the International Symposium on Intelligent Data Analysis*. 2015, pp. 72–83.

[22] Abdelhamid Bouchachia and Charlie Vanaret. "GT2FC: An online growing interval type-2 self-learning fuzzy classifier". In: *IEEE Transactions on Fuzzy Systems* 22.4 (2014), pp. 999–1018.

[23] Hervé Bourlard and Yves Kamp. "Auto-association by multilayer perceptrons and singular value decomposition". In: *Biological cybernetics* 59.4 (1988), pp. 291–294.

[24] Matthew Boutell, Jiebo Luo, Xipeng Shen, and Christopher Brown. "Learning multi-label scene classification". In: *Pattern recognition* 37.9 (2004), pp. 1757–1771.

[25] Samuel Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher Manning, and Christopher Potts. "A fast unified model for parsing and sentence understanding". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. 2016.

[26] Lei Cao and Jianhua Xu. "A label compression coding approach through maximizing dependence between features and labels for multi-label classification". In: *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. 2015, pp. 1–8.

[27] Gail Carpenter, Stephen Grossberg, and John Reynolds. "ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network". In: *Neural networks* 4.5 (1991), pp. 565–588.

[28] Vitor Carvalho and William Cohen. "Single-pass online learning: Performance, voting schemes and online feature selection". In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2006, pp. 548–553.

[29] Gladys Castillo. "Adaptive learning algorithms for Bayesian network classifiers". PhD thesis. Aveiro University, 2006.

[30] Allen Chan and Alex Freitas. "A new ant colony algorithm for multi-label classification with applications in bioinfomatics". In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. 2006, pp. 27–34.

[31] Danqi Chen and Christopher Manning. "A fast and accurate dependency parser using neural networks". In: *Proceedings of Empirical Methods in Natural Language Processing*. 2014, pp. 740–750.

[32] Shang-Fu Chen, Yi-Chen Chen, Chih-Kuan Yeh, and Yu-Chiang Frank Wang. "Order-free RNN with visual attention for multi-label classification". In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*. 2018, pp. 6714–6721.

[33] Yao-Nan Chen and Hsuan-Tien Lin. "Feature-aware label space dimension reduction for multi-label classification". In: *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS)*. 2012, pp. 1538–1546.

[34] Weiwei Cheng and Eyke Hüllermeier. "Combining instance-based learning and logistic regression for multilabel classification". In: *Machine Learning* 76.2-3 (2009), pp. 211–225.

[35] Moustapha Cisse, Nicolas Usunier, Thierry Artieres, and Patrick Gallinari. "Robust bloom filters for large multilabel classification tasks". In: *Proceedings of the 27th Advances in Neural Information Processing Systems (NIPS)*. 2013, pp. 1851–1859.

[36] Amanda Clare and Ross King. "Knowledge discovery in multi-label phenotype data". In: *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*. 2001, pp. 42–53.

[37] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. "Very deep convolutional networks for text classification". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*. 2017, pp. 1107–1116.

[38] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina Balcan, and Le Song. "Scalable kernel methods via doubly stochastic gradients". In: *Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS)*. 2014, pp. 3041–3049.

[39] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. "A sparse johnson: Lindenstrauss transform". In: *Proceedings of the 42nd ACM Symposium on Theory of Computing*. 2010, pp. 341–350.

[40] Tamraparni Dasu, Shankar Krishnan, Suresh Venkatasubramanian, and Ke Yi. "An information-theoretic approach to detecting changes in multi-dimensional data streams". In: *Proceedings of the Symposium on the Interface of Statistics, Computing Science, and Applications*. 2006.

[41] Krzysztof Dembczynski, Weiwei Cheng, and Eyke Hüllermeier. "Bayes optimal multilabel classification via probabilistic classifier chains". In: *Proceedings of the 27th International Conference on Machine Learning (ICML)*. Vol. 10. 2010, pp. 279–286.

[42] Krzysztof Dembczyński, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier. "On label dependence and loss minimization in multi-label classification". In: *Machine Learning* 88.1-2 (2012), pp. 5–45.

[43] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. "Predicting parameters in deep learning". In: *Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS)*. 2013, pp. 2148–2156.

[44] Thomas Dietterich. "Machine learning for sequential data: A review". In: *Proceedings of the Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. 2002, pp. 15–30.

[45] Pedro Domingos and Geoff Hulten. "Mining high-speed data streams". In: *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2000, pp. 71–80.

[46] André Elisseeff and Jason Weston. "A kernel method for multi-labelled classification". In: *Proceedings of the 17th Advances in Neural Information Processing Systems (NIPS)*. 2002, pp. 681–687.

[47] Ryan Elwell and Robi Polikar. "Incremental learning of concept drift in nonstationary environments". In: *IEEE Transactions on Neural Networks* 22.10 (2011), pp. 1517–1531.

[48] Behrouz Farhang-Boroujeny. *Adaptive filters: Theory and applications*. John Wiley & Sons, 2013.

[49] Yoav Freund and Robert Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[50] Yoav Freund and Robert Schapire. "Game theory, on-line prediction and boosting". In: *Proceedings of the 9th Annual Conference on Computational Learning Theory (COLT)*. 1996, pp. 325–332.

[51] Johannes Fürnkranz, Eyke Hüllermeier, Eneldo Loza Mencía, and Klaus Brinker. "Multilabel classification via calibrated label ranking". In: *Machine learning* 73.2 (2008), pp. 133–153.

[52] João Gama. *Knowledge discovery from data streams*. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. CRC Press, 2010.

[53] João Gama, Ricardo Fernandes, and Ricardo Rocha. "Decision trees for mining data streams". In: *Intelligent Data Analysis* 10.1 (2006), pp. 23–45.

[54] João Gama and Petr Kosina. "Recurrent concepts in data streams classification". In: *Knowledge and Information Systems* 40.3 (2014), pp. 489–507.

[55] João Gama and Petr Kosina. "Tracking recurring concepts with meta-learners". In: *Proceedings of the Portuguese Conference on Artificial Intelligence*. 2009, pp. 423–434.

[56] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. "Learning with drift detection". In: *Proceedings of Advances in Artificial Intelligence (SBIA)*. 2004, pp. 286–295.

[57] João Gama, Pedro Medas, and Ricardo Rocha. "Forest trees for online data". In: *Proceedings of the ACM Symposium on Applied Computing (SAC)*. ACM. 2004, pp. 632–636.

[58] João Gama, Indre Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. "A survey on concept drift adaptation". In: *ACM Computing Surveys* 46.4 (2014), pp. 1–44.

[59] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. "Mining data streams under block evolution". In: *ACM SIGKDD Explorations Newsletter* 3.2 (2002), pp. 1–10.

[60] Jing Gao, Wei Fan, and Jiawei Han. "On appropriate assumptions to mine data streams: Analysis and practice". In: *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM)*. 2007, pp. 143–152.

[61] Eva Gibaja and Sebastián Ventura. "A tutorial on multilabel learning". In: *ACM Computing Surveys (CSUR)* 47.3 (2015), pp. 1–39.

[62] Igor Goldenberg and Geoffrey Webb. "Survey of distance measures for quantifying concept drift and shift in numeric data". In: *Knowledge and Information Systems* (2018), pp. 1–25.

[63] Christoph Goller and Andreas Kuchler. "Learning task-dependent distributed representations by backpropagation through structure". In: *Proceedings of the IEEE International Conference on Neural Networks*. Vol. 1. 1996, pp. 347–352.

[64] João Bártolo Gomes, Mohamed Medhat Gaber, Pedro Sousa, and Ernestina Menasalvas. "Mining recurring concepts in a dynamic feature space". In: *IEEE Transactions on Neural Networks and Learning Systems* 25.1 (2013), pp. 95–110.

[65] João Bártolo Gomes, Ernestina Menasalvas, and Pedro Sousa. "Learning recurring concepts from data streams with a context-aware ensemble". In: *Proceedings of the ACM Symposium on Applied Computing (SAC)*. 2011, pp. 994–999.

[66] Paulo Gonçalves Jr. and Roberto Barros. "RCD: A recurring concept drift framework". In: *Pattern Recognition Letters* 34.9 (2013), pp. 1018–1025.

[67] Ram B Gurung, Tony Lindgren, and Henrik Boström. "Learning decision trees from histogram data using multiple subsets of bins". In: *Proceedings of the 29th International Flairs Conference*. 2016.

[68] Michael Hahsler and Margaret Dunham. "Temporal structure learning for clustering massive data streams in real-time". In: *Proceedings of the SIAM International Conference on Data Mining (SDM)*. 2011, pp. 664–675.

[69] Lawrence O Hall, Nitesh Chawla, Kevin W Bowyer, et al. "Combining decision trees learned in parallel". In: *Working Notes of the KDD-97 Workshop on Distributed Data Mining*. 1998, pp. 10–15.

[70] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian Witten. "The WEKA data mining software: An update". In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18.

[71] Michael Harries. *SPLICE-2 comparative evaluation: Electricity pricing*. Tech. rep. University of New South Wales, 1999.

[72] Michael Bonnell Harries, Claude Sammut, and Kim Horn. "Extracting hidden context". In: *Machine learning* 32.2 (1998), pp. 101–126.

[73] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[74] Robert Hecht-Nielsen. "Context vectors: General purpose approximate meaning representations self-organized from raw data". In: *Proceedings of the World Congress on Computational Intelligence, Neural networks*. 1994, pp. 43–56.

[75] Karl Moritz Hermann and Phil Blunsom. "The role of syntax in vector space models of compositional semantics". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*. 2013, pp. 894–904.

[76] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. "Deep neural networks for acoustic modeling in speech recognition". In: *IEEE Signal processing magazine* 29 (2012), pp. 82–97.

[77] Geoffrey Hinton and Ruslan Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *science* 313.5786 (2006), pp. 504–507.

[78] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[79] Ryan Hoens, Robi Polikar, and Nitesh Chawla. "Learning from streaming data with concept drift and imbalance: An overview". In: *Progress in Artificial Intelligence* 1.1 (2012), pp. 89–101.

[80] Daniel Hsu, Sham Kakade, John Langford, and Tong Zhang. "Multi-label prediction via compressed sensing". In: *Proceedings of the 23th Annual Conference on Neural Information Processing Systems (NIPS)*. Vol. 22. 2009, pp. 772–780.

[81] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme learning machine: A new learning scheme of feedforward neural networks". In: *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*. Vol. 2. 2004, pp. 985–990.

[82] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. "Label ranking by learning pairwise preferences". In: *Artificial Intelligence* 172.16-17 (2008), pp. 1897–1916.

[83] Geoff Hulten, Laurie Spencer, and Pedro Domingos. "Mining time-changing data streams". In: *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2001, pp. 97–106.

[84] Piotr Indyk and Rajeev Motwani. "Approximate nearest neighbors: Towards removing the curse of dimensionality". In: *Proceedings of the 30th annual ACM Symposium on Theory of Computing*. 1998, pp. 604–613.

[85] Ozan Irsoy and Claire Cardie. "Deep recursive neural networks for compositionality in language". In: *Proceedings of Advances in Neural Information Processing Systems*. 2014, pp. 2096–2104.

[86] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. "Deep unordered composition rivals syntactic methods for text classification". In: *Proceedings of 53rd Annual Meeting of the Association for Computational Linguistics*. 2015, pp. 1681–1691.

[87] Ghazal Jaber, Antoine Cornuéjols, and Philippe Tarroux. "Online learning: Searching for the best forgetting strategy under concept drift". In: *Proceedings of the International Conference on Neural Information Processing (ICONIP)*. 2013, pp. 400–408.

[88] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. "What is the best multi-stage architecture for object recognition?" In: *Proceedings of the 12th IEEE International Conference on Computer Vision (ICCV)*. 2009, pp. 2146–2153.

[89] William Johnson and Joram Lindenstrauss. "Extensions of Lipschitz mappings into a Hilbert space". In: *Contemporary mathematics* 26.189-206 (1984), p. 1.

[90] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. "A convolutional neural network for modelling sentences". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. 2014, pp. 655–665.

[91] Panos Kalnis, Nikos Mamoulis, and Spiridon Bakiras. "On discovering moving clusters in spatio-temporal data". In: *Proceedings of International Symposium on Spatial and Temporal Databases*. 2005, pp. 364–381.

[92] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. "Tracking recurring contexts using ensemble classifiers: An application to email filtering". In: *Knowledge and Information Systems* 22.3 (2010), pp. 371–391.

[93] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. "Detecting change in data streams". In: *Proceedings of the 30th International Conference on Very Large Databases (VLDB)*. 2004, pp. 180–191.

[94] Yoon Kim. "Convolutional neural networks for sentence classification". In: *Proceedings of Empirical Methods in Natural Language Processing*. 2014, pp. 1746–1751.

[95] Dan Klein and Christopher Manning. "Accurate unlexicalized parsing". In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*. 2003, pp. 423–430.

[96] Ralf Klinkenberg. "Learning drifting concepts: Example selection vs. example weighting". In: *Intelligent Data Analysis* 8.3 (2004), pp. 281–300.

[97] Ralf Klinkenberg and Thorsten Joachims. "Detecting concept drift with support vector machines". In: *Proceedings of the 17th International Conference on Machine Learning (ICML)*. 2000, pp. 487–494.

[98] Jeremy Zico Kolter and Marcus Maloof. "Dynamic weighted majority: An ensemble method for drifting concepts". In: *The Journal of Machine Learning Research* 8 (2007), pp. 2755–2790.

[99] Jeremy Zico Kolter and Marcus Maloof. "Using additive expert ensembles to cope with concept drift". In: *Proceedings of the 22nd International Conference on Machine Learning (ICML)*. 2005, pp. 449–456.

[100] Lingpeng Kong, Nathan Schneider, Swabha Swayamdipta, Archna Bhatia, Chris Dyer, and Noah Smith. "A dependency parser for tweets". In: *Proceedings of Empirical Methods in Natural Language Processing*. 2014, pp. 1001–1012.

[101] Georg Krempl, Indre Zliobaite, Dariusz Brzeziński, Eyke Hüllermeier, Mark Last, Vincent Lemaire, Tino Noack, Ammar Shaker, Sonja Sievi, Myra Spiliopoulou, and Jerzy Stefanowski. "Open challenges for data stream mining research". In: *ACM SIGKDD Explorations Newsletter* 16.1 (2014), pp. 1–10.

[102] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "Imagenet classification with deep convolutional neural networks". In: *Proceedings of the 26th Advances in neural information processing systems (NIPS)*. 2012, pp. 1097–1105.

[103] Ludmila Kuncheva. "Classifier ensembles for changing environments". In: *Proceedings of the International Workshop on Multiple Classifier Systems*. 2004, pp. 1–15.

[104] Ludmila Kuncheva and Indre Zliobaite. "On the window size for classification in changing environments". In: *Intelligent Data Analysis* 13.6 (2009), pp. 861–872.

[105] Esther Landhuis. "Scientific literature: information overload". In: *Nature* 535.7612 (2016), pp. 457–458.

[106] Mihai Lazarescu. "A multi-resolution learning approach to tracking concept drift and recurrent concepts". In: *Proceedings of the 5th International Workshop on Pattern Recognition in Information Systems (PRIS)*. 2005, pp. 52–61.

[107] Quoc Le, Tamás Sarlós, and Alexander Smola. "Fastfood-approximating kernel expansions in loglinear time". In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*. 2013, pp. 244–252.

[108] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[109] Daniel Lewandowski, Dorota Kurowicka, and Harry Joe. "Generating random correlation matrices based on vines and extended onion method". In: *Journal of multivariate analysis* 100.9 (2009), pp. 1989–2001.

[110] Jiwei Li, Minh-Thang Luong, Dan Jurafsky, and Eudard Hovy. "When are tree structures necessary for deep learning of representations?" In: *Proceedings of Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. 2015, pp. 2304–2314.

[111] Xin Li and Yuhong Guo. "Multi-label classification with feature-aware non-linear label space transformation". In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*. 2015, pp. 3635–3642.

[112] Nan-Ying Liang, Guang-Bin Huang, Paramasivan Saratchandran, and Narasimhan Sundararajan. “A fast and accurate online sequential learning algorithm for feedforward networks”. In: *IEEE Transactions on Neural Networks* 17.6 (2006), pp. 1411–1423.

[113] Zijia Lin, Guiguang Ding, Mingqing Hu, and Jianmin Wang. “Multi-label classification via feature-aware implicit label space encoding”. In: *Proceedings of the 31st International Conference on Machine Learning (ICML)*. 2014, pp. 325–333.

[114] Nick Littlestone. “Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm”. In: *Machine Learning* 2.4 (1988), pp. 285–318.

[115] Nick Littlestone and Manfred Warmuth. “The weighted majority algorithm”. In: *Information and computation* 108.2 (1994), pp. 212–261.

[116] Mingbo Ma, Liang Huang, Bing Xiang, and Bowen Zhou. “Dependency-based convolutional neural networks for sentence embedding”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*. 2015, pp. 174–179.

[117] Stéphane Mallat and Zhifeng Zhang. “Matching pursuits with time-frequency dictionaries”. In: *IEEE Transactions on signal processing* 41.12 (1993), pp. 3397–3415.

[118] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. “Resource management with deep reinforcement learning”. In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM. 2016, pp. 50–56.

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

[120] Andrew McCallum. “Multi-label text classification with a mixture model trained by EM”. In: *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI) Workshop on Text Learning*. 1999, pp. 1–7.

[121] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. “Recurrent neural network based language model”. In: *Proceedings of the 11th Annual Conference of the International Speech Communication Association*. 2010, pp. 1045–1048.

[122] Tomas Mikolov, Ilya Sutskever, Chen Chen, Gregory Corrado, and Jeffrey Dean. "Distributed representations of words and phrases and their compositionality". In: *Proceedings of Advances in Neural Information Processing Systems*. 2013, pp. 3111–3119.

[123] Leandro Minku, Allan White, and Xin Yao. "The impact of diversity on online ensemble learning in the presence of concept drift". In: *IEEE Transactions on knowledge and Data Engineering* 22.5 (2010), pp. 730–742.

[124] Leandro Minku and Xin Yao. "DDD: A new ensemble approach for dealing with concept drift". In: *IEEE Transactions on Knowledge and Data Engineering* 24.4 (2012), pp. 619–633.

[125] Hossein Morshedlou and Ahmad Abdollahzade Barforoush. "A new history based method to handle the recurring concept shifts in data streams". In: *World Academy of Science, Engineering and Technology* 58 (2009), pp. 917–922.

[126] Lili Mou, Hao Peng, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. "Discriminative neural sentence modeling by tree-based convolution". In: *Proceedings of Empirical Methods in Natural Language Processing*. 2015, pp. 2315–2325.

[127] Robb Muirhead. *Aspects of multivariate statistical theory*. Vol. 197. John Wiley & Sons, 2009.

[128] Jinseok Nam, Eneldo Loza Mencía, Hyunwoo Kim, and Johannes Fürnkranz. "Maximizing subset accuracy with recurrent neural networks in multi-label classification". In: *Proceedings of the 31st Advances in Neural Information Processing Systems (NIPS)*. 2017, pp. 5413–5423.

[129] Jelani Nelson and Huy Nguyên. "OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings". In: *Proceedings of the IEEE 54th Annual Symposium on Foundations of Computer Science*. 2013, pp. 117–126.

[130] Kyosuke Nishida. "Learning and detecting concept drift". In: *Graduate School of Information Science and Technology, Hokkaido University* (2008).

[131] Kyosuke Nishida and Koichiro Yamauchi. "Detecting concept drift using statistical testing". In: *Proceedings of the International Conference on Discovery Science*. 2007, pp. 264–269.

[132] Irene Ntoutsi, Myra Spiliopoulou, and Yannis Theodoridis. "Tracing cluster transitions for different cluster types". In: *Control and Cybernetics* 38.1 (2009), pp. 239–259.

[133] Márcia Oliveira and João Gama. "MEC - Monitoring clusters' transitions". In: *Proceedings of the 5th Starting AI Researchers' Symposium (STAIRS)*. 2010, pp. 212–224.

[134] Nikunj Oza and Stuart Russell. "Experimental comparisons of online and batch versions of bagging and boosting". In: *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2001, pp. 359–364.

[135] Amir Padovitz, Seng Wai Loke, and Arkady Zaslavsky. "Towards a theory of context spaces". In: *Proceedings of the 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops*. 2004, pp. 38–42.

[136] Ewan Page. "Continuous inspection schemes". In: *Biometrika* 41.1/2 (1954), pp. 100–115.

[137] Bo Pang and Lillian Lee. "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales". In: *Proceedings of the 43rd annual meeting on Association for Computational Linguistics*. 2005, pp. 115–124.

[138] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation". In: *Proceedings of Empirical Methods in Natural Language Processing*. 2014, pp. 1532–1543.

[139] Julien Perolat, Joel Z Leibo, Vinicius Zambaldi, Charles Beattie, Karl Tuyls, and Thore Graepel. "A multi-agent reinforcement learning model of common-pool resource appropriation". In: *Proceedings of Advances in Neural Information Processing Systems*. 2017, pp. 3643–3652.

[140] Jordan Pollack. "Recursive distributed representations". In: *Artificial Intelligence* 46.1 (1990), pp. 77–105.

[141] Wei Qu, Yang Zhang, Junping Zhu, and Qiang Qiu. "Mining multilabel concept-drifting data streams using dynamic classifier ensemble". In: *Proceedings of the 1st Asian Conference on Machine Learning (ACML)*. 2009, pp. 308–321.

[142] Ali Rahimi and Benjamin Recht. "Random features for large-scale kernel machines". In: *Proceedings of the 21st Annual Conference on Neural Information Processing Systems (NIPS)*. 2007, pp. 1177–1184.

[143] Rafal Rak, Lukasz Kurgan, and Marek Reformat. "A tree-projection-based algorithm for multi-label recurrent-item associative-classification rule generation". In: *Data & Knowledge Engineering* 64.1 (2008), pp. 171–197.

[144] Daniel Ramage, David Hall, Ramesh Nallapati, and Christopher Manning. "Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP): Volume 1*. 2009, pp. 248–256.

[145] Sasthakumar Ramamurthy and Raj Bhatnagar. "Tracking recurrent concept drift in streaming data using ensemble classifiers". In: *Proceedings of the 6th International Conference on Machine Learning and Applications (ICMLA)*. 2007, pp. 404–409.

[146] Marc'Aurelio Ranzato, Lan Boureau, and Yann LeCun. "Sparse feature learning for deep belief networks". In: *Proceedings of the 21th Advances in Neural Information Processing Systems (NIPS)*. 2007, pp. 1185–1192.

[147] Jesse Read, Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. "Scalable and efficient multi-label classification for evolving data streams". In: *Machine Learning* 88.1-2 (2012), pp. 243–272.

[148] Jesse Read and Fernando Pérez-Cruz. "Deep learning for multi-label classification". In: *CoRR* abs/1502.05988 (2015).

[149] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. "Classifier chains for multi-label classification". In: *Machine learning* 85.3 (2011), pp. 333–359.

[150] Jesse Read, Bernhard Pfahringer, and Geoffrey Holmes. "Multi-label classification using ensembles of pruned sets". In: *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM)*. 2008, pp. 995–1000.

[151] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. "Contractive auto-encoders: Explicit invariance during feature extraction". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning (ICML)*. 2011, pp. 833–840.

[152] David Rumelhart, Geoffrey Hinton, and Ronald Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.

[153] Sripirakas Sakthithasan, Russel Pears, Albert Bifet, and Bernhard Pfahringer. "Use of ensembles of Fourier spectra in capturing recurrent concepts in data streams". In: *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. 2015, pp. 1–8.

[154] Marcos Salganicoff. "Density-adaptive learning and forgetting". In: *Proceedings of the 10th International Conference on Machine Learning (ICML)*. Vol. 3. 1993, pp. 276–283.

[155] Andrew Saxe, Pang Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Ng. "On random weights and unsupervised feature learning". In: *Proceedings of the 28th international conference on machine learning (ICML)*. 2011, pp. 1089–1096.

[156] Robert Schapire and Yoram Singer. "BoosTexter: A boosting-based system for text categorization". In: *Machine learning* 39.2-3 (2000), pp. 135–168.

[157] Jeffrey Schlimmer and Richard Granger. "Incremental learning from noisy data". In: *Machine Learning* 1.3 (1986), pp. 317–354.

[158] Wouter Schmidt, Martin Kraaijveld, and Robert Duin. "Feedforward neural networks with random weights". In: *Proceedings of 11th IAPR International Conference on Pattern Recognition Methodology and Systems*. 1992, pp. 1–4.

[159] Martin Scholz and Ralf Klinkenberg. "An ensemble classifier for drifting concepts". In: *Proceedings of the 2nd International Workshop on Knowledge Discovery in Data Streams*. Vol. 6. 11. 2005, pp. 53–64.

[160] Junming Shao, Zahra Ahmadi, and Stefan Kramer. "Prototype-based learning on concept-drifting data streams". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2014, pp. 412–421.

[161] Zhongwei Shi, Yimin Wen, Chao Feng, and Hai Zhao. "Drift detection for multi-label data streams based on label grouping and entropy". In: *Proceedings of the IEEE International Conference on Data Mining Workshop (ICDMW)*. IEEE. 2014, pp. 724–731.

[162] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *CoRR* abs/1409.1556 (2014). URL: `http://arxiv.org/abs/1409.1556`.

[163] Richard Socher, Brody Huval, Christopher Manning, and Andrew Ng. "Semantic compositionality through recursive matrix-vector spaces". In: *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. 2012, pp. 1201–1211.

[164] Richard Socher, Cliff Lin, Chris Manning, and Andrew Ng. "Parsing natural scenes and natural language with recursive neural networks". In: *Proceedings of the 28th international conference on machine learning (ICML)*. 2011, pp. 129–136.

[165] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng, and Christopher Potts Potts. "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Proceedings of Empirical Methods in Natural Language Processing*. 2013, pp. 1631–1642.

[166] Ge Song and Yunming Ye. "A new ensemble method for multi-label data stream classification in non-stationary environment". In: *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. 2014, pp. 1776–1783.

[167] Myra Spiliopoulou, Irene Ntoutsi, Yannis Theodoridis, and Rene Schult. "Monic: Modeling and monitoring cluster transitions". In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2006, pp. 706–711.

[168] 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 22nd International Joint Conference on Artificial Intelligence (IJCAI)*. 2011, pp. 1583–1588.

[169] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[170] Kenneth Stanley. *Learning concept drift with a committee of decision trees*. Tech. rep. Department of Computer Sciences, University of Texas at Austin, USA, 2003.

[171] Nick Street and YongSeog Kim. "A streaming ensemble algorithm (SEA) for large-scale classification". In: *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2001, pp. 377–382.

[172] Farbound Tai and Hsuan-Tien Lin. "Multilabel classification with principal label space transformation". In: *Neural Computation* 24.9 (2012), pp. 2508–2542.

[173] Kai Sheng Tai, Richard Socher, and Christopher Manning. "Improved semantic representations from tree-structured long short-term memory networks". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*. 2015, pp. 1556–1566.

[174] Lena Tenenboim-Chekina, Lior Rokach, and Bracha Shapira. "Identification of label dependencies for multi-label classification". In: *Working Notes of the 2nd International Workshop on Learning from Multi-Label Data*. 2010, pp. 53–60.

[175] Lloyd Trefethen and David Bau III. *Numerical linear algebra*. Vol. 50. Society for Industrial and Applied Mathematics (SIAM), 1997.

[176] Grigorios Tsoumakas, Anastasios Dimou, Eleftherios Spyromitros, Vasileios Mezaris, Ioannis Kompatsiaris, and Ioannis Vlahavas. "Correlation-based pruning of stacked binary relevance models for multi-label learning". In: *Proceedings of the 1st international workshop on learning from multi-label data*. 2009, pp. 101–116.

[177] Grigorios Tsoumakas and Ioannis Katakis. "Multi-label classification: An overview". In: 3.3 (2007), pp. 1–13.

[178] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. *Data mining and knowledge discovery handbook, chapter Mining Multi-label Data*. 2010.

[179] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. "Mining multi-label data". In: (2009), pp. 667–685.

[180] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. "Random k-labelsets for multilabel classification". In: *IEEE Transactions on Knowledge and Data Engineering* 23.7 (2011), pp. 1079–1089.

[181] Grigorios Tsoumakas, Eleftherios Spyromitros-Xioufis, Jozef Vilcek, and Ioannis Vlahavas. "Mulan: A Java library for multi-Label learning". In: *Journal of Machine Learning Research* 12 (2011), pp. 2411–2414.

[182] Grigorios Tsoumakas and Ioannis Vlahavas. "Random k-labelsets: An ensemble method for multilabel classification". In: *Proceedings of 18th European Conference on Machine Learning* (*ECML*). 2007, pp. 406–417.

[183] Alexey Tsymbal. "The problem of concept drift: Definitions and related work". In: *Computer Science Department, Trinity College Dublin* 106 (2004).

[184] Naonori Ueda and Kazumi Saito. "Parametric mixture models for multi-labeled text". In: *Proceedings of the 18th Advances in Neural Information Processing Systems* (*NIPS*). 2003, pp. 737–744.

[185] Peter Vorburger and Abraham Bernstein. "Entropy-based concept shift detection". In: *Proceedings of the IEEE 6th International Conference on Data Mining* (*ICDM*). 2006, pp. 1113–1118.

[186] Abraham Wald and Jacob Wolfowitz. "Optimum character of the sequential probability ratio test". In: *The Annals of Mathematical Statistics* 19.3 (1948), pp. 326–339.

[187] Changhu Wang, Shuicheng Yan, Lei Zhang, and Hong-Jiang Zhang. "Multi-label sparse coding for automatic image annotation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*). 2009, pp. 1643–1650.

[188] Haixun Wang, Wei Fan, Philip Yu, and Jiawei Han. "Mining concept-drifting data streams using ensemble classifiers". In: *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2003, pp. 226–235.

[189] Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. "CNN-RNN: A unified framework for multi-label image classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2285–2294.

[190] Lulu Wang, Hong Shen, and Hui Tian. "Weighted ensemble classification of multi-label data streams". In: *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*. 2017, pp. 551–562.

[191] Peng Wang, Peng Zhang, and Li Guo. "Mining multi-label data streams using ensemble-based active learning". In: *Proceedings of the SIAM international conference on data mining (SDM)*. 2012, pp. 1131–1140.

[192] Geoffrey Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. "Characterizing concept drift". In: *Data Mining and Knowledge Discovery* 30.4 (2016), pp. 964–994.

[193] Jörg Wicker, Bernhard Pfahringer, and Stefan Kramer. "Multi-label classification using boolean matrix decomposition". In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC)*. 2012, pp. 179–186.

[194] Jörg Wicker, Andrey Tyukin, and Stefan Kramer. "A nonlinear label compression and transformation method for multi-label classification using autoencoders". In: *Proceedings of the 20th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD)*. 2016, pp. 328–340.

[195] Gerhard Widmer and Miroslav Kubat. "Learning in the presence of concept drift and hidden contexts". In: *Machine learning* 23.1 (1996), pp. 69–101.

[196] Gerhard Widmer and Miroslav Kubat. "Special issue on context sensitivity and concept drift-introduction". In: *Machine Learning* 32.2 (1998).

[197] Bernard Widrow, Aaron Greenblatt, Youngsik Kim, and Dookun Park. "The no-prop algorithm: A new learning algorithm for multilayer neural networks". In: *Neural Networks* 37 (2013), pp. 182–188.

[198] Ronald Williams and David Zipser. "Gradient-based learning algorithms for recurrent". In: *Backpropagation: Theory, architectures, and applications* 433 (1995).

[199] Clay Woolam, Mohammad Masud, and Latifur Khan. "Lacking labels in the stream: Classifying evolving stream data with few labels". In: *Proceedings of the International Symposium on Methodologies for Intelligent Systems*. 2009, pp. 552–562.

[200] Yao-Yuan Yang, Yi-An Lin, Hong-Min Chu, and Hsuan-Tien Lin. "Deep learning with a rethinking structure for multi-label classification". In: *CoRR* abs/1802.01697 (2018). URL: `http://arxiv.org/abs/1802.01697`.

[201] Ying Yang, Xindong Wu, and Xingquan Zhu. "Mining in anticipation for concept change: Proactive-reactive prediction in data streams". In: *Data mining and knowledge discovery* 13.3 (2006), pp. 261–289.

[202] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. "Deep fried convnets". In: *Proceedings of the IEEE International Conference on Computer Vision (CCV)*. 2015, pp. 1476–1483.

[203] Chih-Kuan Yeh, Wei-Chieh Wu, Wei-Jen Ko, and Yu-Chiang Frank Wang. "Learning deep latent space for multi-label classification". In: *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*. 2017, pp. 2838–2844.

[204] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. "Comparative study of CNN and RNN for natural language processing". In: *CoRR* abs/1702.01923 (2017). URL: `http://arxiv.org/abs/1702.01923`.

[205] Min-Ling Zhang. "ML-RBF: RBF neural networks for multi-label learning". In: *Neural Processing Letters* 29.2 (2009), pp. 61–74.

[206] Min-Ling Zhang and Zhi-Hua Zhou. "A k-nearest neighbor based algorithm for multi-label classification". In: *Proceedings of the IEEE International Conference on Granular Computing*. Vol. 2. 2005, pp. 718–721.

[207] Min-Ling Zhang and Zhi-Hua Zhou. "Multilabel neural networks with applications to functional genomics and text categorization". In: *IEEE transactions on Knowledge and Data Engineering* 18.10 (2006), pp. 1338–1351.

[208] Rui Zhang, Honglak Lee, and Dragomir Radev. "Dependency sensitive convolutional neural networks for modeling sentences and documents". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2016, pp. 1512–1521.

[209] Ye Zhang and Byron Wallace. "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification". In: *CoRR* abs/1510.03820 (2015).

[210] Yi Zhang and Jeff Schneider. "Maximum margin output coding". In: *Proceedings of the 29th International Conference on Machine Learning* (*ICML*). 2012, pp. 1575–1582.

[211] Yi Zhang and Jeff Schneider. "Multi-label output codes using canonical correlation analysis". In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics* (*AISTATS*). 2011, pp. 873–882.

[212] Tianyi Zhou, Dacheng Tao, and Xindong Wu. "Compressed labeling on distilled labelsets for multi-label learning". In: *Machine Learning* 88.1-2 (2012), pp. 69–126.

[213] Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. "Long short-term memory over recursive structures". In: *Proceedings of the 32nd International Conference on Machine Learning*. 2015, pp. 1604–1612.

[214] Indre Zliobaite. "Adaptive training set formation". PhD thesis. Vilniaus universitetas, 2010.

[215] Indre Zliobaite, Mykola Pechenizkiy, and João Gama. "An overview of concept drift applications". In: *Big Data Analysis: New Algorithms for a New Society*. Springer, 2016, pp. 91–114.

CV has been removed from the online version.