NOVEL CLASS DETECTION AND CROSS-LINGUAL DUPLICATE DETECTION OVER ONLINE DATA STREAM

by

Ahmad Mohammad Mustafa

To The Almighty and My Family

NOVEL CLASS DETECTION AND CROSS-LINGUAL DUPLICATE DETECTION OVER ONLINE DATA STREAM

by

AHMAD MOHAMMAD MUSTAFA, BS, MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

May 2018

ACKNOWLEDGMENTS

My sincerest gratitude first goes to Professor Latifur Khan, my supervisor, for giving me an opportunity to pursue the doctoral program. This dissertation would not have been possible without his support and guidance. I am also thankful to my committee members, Dr. Farokh Bastani, Dr. Haim Schweitzer, and Dr. Alvaro Cárdenas, for their suggestions on the content of this work. I would also like to thank the graduate students in Dr. Khan's group for their help and support.

I am grateful to my parents, Mr. Mohammad Mustafa and Mrs. Nuha Smadi, my sisters, and brothers for their encouragement and sacrifices during my graduate studies. I would also like to express my special thanks to my friends who eased my way through the hurdles in everyday life. I also wish to thank every single one of my teachers for pushing me to do my best. Finally and most importantly, I thank God for His grace and mercy, and for the amazing people He has placed in my life.

This material is based in part upon work supported by the National Science Foundation, the National Security Agency, the Air Force Office of Scientific Research, the United States Department of Energy, and IBM Research.

February 2018

NOVEL CLASS DETECTION AND CROSS-LINGUAL DUPLICATE DETECTION OVER ONLINE DATA STREAM

Ahmad Mohammad Mustafa,
The University of Texas at Dallas, 2018

Supervising Professor: Dr. Latifur Khan, Chair

Data streams are continuous flows of data points. They are very common now-a-days in several domains such as e-commerce, education, health, security, and social networks. Examples of data streams are network traffic, social media blogs, sensor data, call center records, and credit card transactions. Their sheer volume and throughput speed pose a great challenge for the data mining community to extract useful knowledge from such streams. Data stream classification refers to the task of predicting class labels of data instances using classification models trained with the past labeled data. Data stream classification has been a major research thrust for the past several years because of increasing demand in many business and security applications. Data streams induce several unique properties when compared with traditional datasets, such as infinite length, concept-drift, and concept-evolution. Concept-drift occurs in data streams when the underlying concept of data changes over time, making previously trained models obsolete. Concept-evolution refers to the emergence of a new or novel class. Existing classification techniques that address the concept-evolution and concept-drift require labeled data to detect stream changes. Labeled data is scarce and expensive. This dissertation addresses the aforementioned challenges in a number of ways. We address infinite length, concept-drift, and concept-evolution by proposing an ensemble-based classification model that exploits unsupervised deep embeddings and non-parametric change point detection. We show empirically on both synthetic and several benchmark data streams that the proposed techniques outperform state-of-the-art techniques.

The detection of duplicate news reports - those which cover the same event - plays an important role in condensing information about an event into an easily digestible format. Several methods of duplicate report detection have been developed. However, these existing methods either result in poor accuracy or are not effective when reports are in different languages. We propose a novel method to measure the similarity between news reports written in different languages in an unsupervised learning approach. Experimental results on publicly available datasets of multi-lingual reports show that our approach efficiently and significantly reduces duplicate detection errors compared to state-of-the-art techniques.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Data streams are continuous flows of data. Typical examples of data streams include network traffic, sensor data, and call center records, among others. The sheer volume and throughput speed pose a great challenge for the data mining community to extract useful knowledge from such streams. Data streams demonstrate several unique properties when compared with traditional data set, such as: infinite length, concept-drift, and concept-evolution. Concept-drift occurs in data streams when the underlying concept of data changes over time (Aggarwal and Yu, 2010; Zhang et al., 2011; Kolter and Maloof, 2007; Street and Kim, 2001; Wang et al., 2003, 2006, 2007). In concept-evolution, a new class may emerge over stream (Masud et al., 2011). Neither multi-step methodologies and techniques nor multi-scan algorithms suitable for typical knowledge discovery and data mining can be readily applied to data streams due to well-known limitations such as unbounded memory to handle infinite length, online data processing to handle concept drift, and the need for one-pass techniques (i.e., forgotten raw data).

Data stream classification has been a major research thrust for the past several years because of increasing demand in many business and security applications, such as credit card transaction monitoring, online blog or micro-blog (e.g., twitter messages) categorization, and evolving malicious code detection. Traditional batch classification techniques are not applicable to the aforementioned domains because of the evolving nature of the data. In particular for malware detection, signature-based techniques are widely used and dynamic signature updating is very common. Antivirus software can adapt to new malware threats by updating its signature database as soon as a single instance of the malware has been identified and analyzed by experts. In contrast, polymorphic malware can pose some significant challenges to signature updating because the malware modifies itself during propagation yielding many variants over time. A malware detector may fail to identify such malware due to the usage of outdated signatures. One way to address this problem is to update the signatures, which achieves superior adaptability over current polymorphic malware. While this advantage has kept antivirus products mostly ahead in the virus-antivirus co-evolution race (Hamlen et al., 2009) up to the present time, a malware detector needs to be adaptive to cope with the changes in the wild.

Stream classification falls into the following categories: single model, ensemble classification, and hybrid. Single model classification techniques maintain and incrementally update the single classification model and effectively respond to concept drift (Yang et al., 2005). In ensemble based techniques, a number of classification models are maintained, and over time some outdated classification models are replaced by new models (Masud et al., 2011). Hybrid methods combine the strength of the above two (Brzezinski and Stefanowski, 2014). In current state of the art ensemble techniques, data stream is divided into a number of chunks so that each chunk can be accommodated in memory and processed online (Parker et al., 2012; Masud et al., 2011). Each chunk is used to train one classification model as soon as all the instances in the chunk are labeled. Concept-drift is handled by maintaining an ensemble of M such classification models. An unlabeled instance is classified by taking a majority vote among the classifiers in the ensemble. The ensemble is continuously updated so that it represents the most recent concept in the stream. The update is performed as follows. As soon as a new model is trained, one of the existing models in the ensemble is replaced by it, if necessary. The removed model is chosen by evaluating the error rate of each of the existing models in the ensemble on the latest labeled chunk, and discarding the one with the highest error rate.

Current state-of-the-art techniques suffer from a high number of misclassifications, such as missing novel class instances (false negatives) or incorrectly identifying existing classes as novel (false positives).

Concept-drift refers to the change of data feature values over time. To prevent significant performance degradation due to concept-drift, the classification model needs continual updating and maintenance via retraining. However, given that model updating is time consuming and requires labeled data, which is usually scarce, determining appropriate retraining frequencies becomes critical. A naïve approach is to retrain the classifier periodically (Masud et al., 2011). However, this strategy results in unnecessary updates and missed drifts, which can degrade performance. Such limitations can be avoided by retraining only when the distribution of the data changes (Haque et al., 2016).

The state-of-the-art stream classification techniques divide data streams in equal sizes (i.e., fixed size) (Masud et al., 2011; Aggarwal and Yu, 2010). These approaches fail to capture concept-drift and concept-evolution immediately. Moreover, if the data chunk is too small, such techniques can engender models of poor quality (few training data points) and/or additional computational overhead to update the ensemble. Conversely, for large chunk sizes, these approaches must wait much longer to build the next classifier. As a result, the ensemble is updated less frequently than desired, meaning the ensemble remains outdated for a longer period of time. This ultimately causes increased error rates.

1.1 Unsupervised deep embedding for novel class detection

To overcome the aforementioned disadvantages, we propose our approach to monitor data streams and determine the chunk size dynamically by promptly tracking distributional changes. *Change Point Detection* (CPD) techniques can be applied to observe and detect distributional changes. CPD can also be applied to partition the data stream. Most of the CPD algorithms assume that distributions of data before and after the change are known (Basseville and Nikiforov, 1993; Chen and Gupta, 2012). However, in practice, this assumption may not always hold. If the distributions of data before and after are unknown entirely, these approaches are not applicable. For this specific case, methods have been developed for detecting change points nonparametrically over single-dimensional data (Ferger, 1991; Baron, 2000). If data is multidimensional, these approaches detect changes after reducing the number dimensions (e.g., using PCA in (Qahtan et al., 2015)). In contrast, our proposed CPD approach can be scaled over multidimensional data without reducing dimensionality. Moreover, since it is a nonparametric approach, we do not assume a particular data distribution—it can detect change points regardless of the distribution of the data. In addition, current state-of-the-art CPD techniques (Haque et al., 2016; Bifet et al., 2010) require the availability of class labels for some or all data points. Our proposed CPD technique is unsupervised and does not require class labels.

In addition to CPD, with regard to novel class detection, both clustering and outlier detection are essential parts. Studies have shown that using deep learning techniques in classification, clustering and outlier detection have significantly outperformed conventional methods (Vincent et al., 2010; Xie et al., 2015; Sakurada and Yairi, 2014). Motivated by such successes, we investigate the application of deep learning methods to novel class detection. In particular, we employ autoencoders for feature learning. Several papers have proposed outlier detection methods using autoencoders (Sakurada and Yairi, 2014; Mazhelis, 2006). Prior work has also shown that autoencoders can obtain stable and effective clustering (Song et al., 2013; Xie et al., 2015). The aforementioned approaches have proved the effectiveness of autoencoders in both clustering and outlier detection. However, previous approaches have not used deep learning techniques in the context of novel class detection over stream data.

Denoising Autoencoders (DAEs) have been used in several studies for their ability to extract abstract features that can outperform the original input feature representation when used in classification or other tasks (Weninger et al., 2014; Vincent et al., 2010). A DAE is a type of neural network with hidden layers aiming to reconstruct the input vector from a corrupted version with minimum error. It is an unsupervised learning method. When designed with multiple hidden layers, DAEs can learn *deep* abstract features.

Our proposed work has several contributions to data stream classification.

- To the best of our knowledge, this is the first study to combine deep learning with *novel class detection in stream data*.

- we introduce a nonparametric multidimensional change point detection to detect concept-drift.
- our proposed approach enriches traditional classification models with a novel class detection mechanism and unsupervised deep learning method, combining the strength of these techniques.
- we apply our technique on both synthetic and real-world data and obtain much better results than state-of-the-art stream classification algorithms.

1.2 Binary Class Stream Data Classification for Multidimensional Data using Change point detection

In this work, we propose a novel Stream Classification using Change Detection (SC^2D) method. We use decision tree to model changes across multiple dimensions and classify instances. The dimensions contribute to the prediction based on their normalized information gain using decision tree. So, using this approach we do not need to manually set a threshold. Decision tree partitions and selects the dimensions based on their information gain. This proposed approach is then used to determine chunk boundary over data streams dynamically which leads to better models to classify instances of evolving data streams. Since it uses dynamic chunk size based on change in the distribution of the data, it can capture drift in the concept or detect arrival of a new class immediately. We focus on classifying instances of data streams having two classes e.g., benign and malicious. Our adaptive stream classification technique is beneficial to various communities, including those working in cyber security to analyze massive amounts of stream data e.g., differentiate between benign and malicious events.

We focus on classifying data streams having instances from two classes. We consider a scenario where instances from each class come in a sequence and class of the sequences change alternatively. We determine the class label of a chunk by calculating the change point and taking into account previous sequence's class label. It reduces false alarm rates and overall classification error. We test the performance of our approach on several benchmark datasets. We compare the experimental results with both supervised and unsupervised techniques. For supervised techniques we include various state of the art approaches implemented in MOA (Bifet et al., 2010) framework. On the other hand, for unsupervised techniques, we use variants of multi dimensional change point detection approaches.

Primary contributions are as follows.

- To the best of our knowledge, this is the first effort that develops an adaptive classification technique exploiting multi-dimensional non parametric change point detection to address concept-drift/concept-evolution problems in a timely manner.
- Our approach exploits dynamic sized chunks. In other words, chunk size is determined over stream on the fly based on multi-dimensional non-parametric change point detection. Once a change point is detected, the algorithm continues to find subsequent change points. Two subsequent change points create a chunk with variable length.

1.3 Supervised Duplicate Detection

Textual data streams, such as emails, blogs, or news reports, are continuous flows of data. Streams tend to demonstrate several unique properties when compared to traditional datasets, but they also come with their own set of challenges. Traditional batch classification techniques are often rendered useless in the face of sheer volume and throughput speed, as well as concept drift. Concept drift refers to the change of data feature values over time (Aggarwal and Yu, 2010; Zhang et al., 2011; Kolter and Maloof, 2007; Street and Kim, 2001; Wang et al., 2003, 2006, 2007). As a result of concept drift, a supervised classification model needs continual updating (via retraining on recent data) in order to avoid potential performance degradation over time (Masud et al., 2008).

News reports are another type of textual data stream. Because of the nature of political conflicts and diversity of sources, multiple reports often address the same event, but with different details or dates of publication. These so-called *duplicate* reports by the political scientists are most often found when different news agencies cover the same event, but they can also be presented as updated versions of old reports from the same agency. The detection of duplicate news reports in a textual stream environment is an important problem: being able to condense duplicate reports would allow analysts or machines to merge information extracted from different documents for a given event, thus enabling the extracted information to be interpreted correctly.

Our goal in is to examine the task of duplicate reports detection in a textual stream environment. When detecting duplicate reports, a newly published report can be compared to a set of stored reports and is considered a duplicate if there is a match. At first glance, one may address this task using an unsupervised approach. One advantage of this approach is that it does not rely on labeled data, and can be applied simply by calculating how similar the features of a pair of reports are. A pair is considered duplicate only if this similarity is greater than a prearranged threshold set by the domain expert. However, the task is not as simple as it seems. To see the reason, consider Figures 1.1 and 1.2, which show a pair of duplicate reports and a pair of non-duplicate reports respectively. Using unigrams as features, the cosine similarity of the duplicate reports is 0.49, which is lower than that of the non-duplicate reports (0.82).

REPORT$_1$: The Kiev police have reported that nine persons were killed during the clashes in Kiev on 18 February, the Interfax-Ukraine news agency reported at 1703 gm…	REPORT$_2$: 11 people died during clashes in the Ukrainian capital, Kiev, on 18 February, Interfax-Ukraine reported on Tuesday 18 February…

Figure 1.1: Example of duplicate news reports.

REPORT$_1$: Many persons are feared killed in Maiduguri, the Borno State capital, Monday, after two explosions went off at the Maiduguri main market, where two suicide bombers last week detonated deadly explosives, killing themselves and several other people...	REPORT$_2$: A double bombing Thursday in the central Nigeria city of Jos killed at least 31 people near the site where a similar attack in May killed 118 people...

Figure 1.2: Example of non-duplicate news reports.

Given the weaknesses of the unsupervised approach, one may consider a supervised approach. In the supervised approach, the pair is measured against a trained model, which is created using the features of a training pair set. The pairs are then labeled as either duplicate or non-duplicate (discrete) by human coders. Though it is technically functional, the supervised approach is neither cost-effective nor adaptive to concept drift.

There are four main challenges to focus upon:

i. The amount of reports required to be stored directly affects the number of comparisons.

ii. The features used to differentiate between duplicate and non-duplicate (discrete) reports should not lead to high dimensionality. Otherwise, they would result in poor accuracy.

iii. Any static model is fundamentally maladaptive to concept drift.

iv. Human-labeled data is expensive. Any improved supervised approach should minimize the amount of human-labeled data sets needed to train (or retrain) the model.

Therefore, we use change point detection as an active learning technique. Active learning (Zliobaite et al., 2014) focuses on reducing the number of human-labeled data for learning classification models.

In a more refined, novel approach, the reports are stored in a sliding window, a type of queue data structure. A sliding window of size d stores reports of the past d days. To represent the similarity between the target and past reports, the named entities (NEs : including location and person names, organizations, and dates) within the reports are compared. Then, a similarity tuple is generated. The tuple consists of the concatenation of the similarity values between NEs. It is then fed into a logistic regression classifier, which labels it as duplicate or non-duplicate. All of this labelling is typically done by human coders, but to reduce costs within this approach, humans will only label the data points for which the classifier confidence value changes. Such changes are detected when the parameters of confidence distribution shift.

We have the following contributions:

- We propose a supervised duplicate detection method to find news reports discussing the same events that is adaptive to concept drift.
- We apply unsupervised change detection method to reduce the human effort spent in labeling the data needed to update the model.
- We show that our supervised approach outperforms other baseline approaches.
- Empirical results show that our supervised approach can minimize the number of labeled data needed for model updating by 85 % without significantly reducing the accuracy.

1.4 Unsupervised Duplicate Detection

News reports are the main type of textual data streams. Due to the nebulous nature of information distribution during a tragedy, reports covering the same event often differ in their details or dates of publication. These so-called *duplicate* reports by the political scientists are most often found when different news agencies cover the same event, but they can also be presented as updated versions of old reports from the same agency. Condensing these duplicates into one organized source allows analysts and machines to extract and interpret information far more efficiently.

Our goal is to examine the task of duplicate reports detection across languages. Two reports are duplicate if they cover the same event. Our definition of duplicate reports includes reports that are not identical such as in case of developing news where the news emerges in parts. For example, a report about an earthquake in region X is followed by a second report stating that the affected region is Y. Later, a third report shows a total of 27 casualties, etc. Here, the first news report is reporting on the same event as the second and third ones but the newer reports have more details and, thus, the overlap with regard to news story is not high. The size of reports may vary. For example, breaking news might be short and may consist of one or two sentences. Comparison with long news articles is challenging.

Typically, in order to detect duplicate reports, a newly published report is compared to a set of stored reports. If the target report matches any of the stored ones, it is declared duplicate. We address the problem by proposing a novel approach to measuring the cross-lingual similarity between news reports.

At first glance, one may address this task using an unsupervised approach. One advantage of this approach is that it does not rely on labeled data, and can be applied simply by calculating how similar the features of a pair of reports are (Mahajan et al., 2014; Alsulami et al., 2012). A pair is considered duplicate only if this similarity is greater than a prearranged threshold set by the domain expert. However, the task is not as simple as it seems. Different news sources use different languages to report events. Detecting duplicates across languages is a challenging task. When comparing between two documents written in two different languages, typical unsupervised similarity measurements like cosine similarity between TfIdf vectors are not effective. That is because the two documents use two disjoint sets of vocabulary. So, they do not share the same dimensional space even though they may report the same event. Take for example the reports shown in Figure 1.3. These two articles are addressing the same event in two different languages using different vocabulary. Although it may seem easy to match the English and Spanish articles due to the use of similar words, the problem becomes more difficult for articles using different characters sets such as Arabic.

One may suggest using machine translation to translate the articles to one language (e.g., English) and compute similarity between them using known methods (e.g., Cosine

REPORT$_1$: Thousands of people in Madrid have held a rally in protest at plans by the Catalan regional government to hold a referendum on independence. Campaigners for Spanish unity filled Plaza de Cibeles calling for the Catalan regional president, Carles Puigdemont, to be sent to jail. Similar demonstrations were held in other Spanish cities but the one in Madrid was the largest since the referendum was called earlier this year…	REPORT$_2$: Con banderas españolas y proclamas contra el referéndum de Cataluña, unas 350 personas se han manifestado en la plaza Sant Jaume de Barcelona a favor de la unidad de España. Durante la manifestación delante de la Generalitat se han producido momentos de tensión cuando algunas personas gritaron consignas independentistas. La concentración fue convocada simultáneamente frente a todos los ayuntamientos del país por la Fundación de Defensa de la Nación Española…

Figure 1.3: English and Spanish duplicate reports taken from EuroNews

Similarity) (Gottschalk and Demidova, 2017; Duh et al., 2013; Alzahrani et al., 2010). However, machine translation is a challenging task and still an active research area. So, with this approach duplicate detection inherits problems from machine translation which may lead to poor results. Moreover, machine translation may not allow us to correctly detect small amounts of overlapping news in the case of developing news.

Another solution is to calculate Jaccard similarity (i.e., overlap) between named entities like locations, persons, and organizations that are mentioned in the documents (Uyar, 2009). This approach may not work in certain scenarios where reports mention different details about an event (e.g., different witnesses and locations) or if the matching between entities is not possible across different languages.

An alternative solution is to use bag of words to characterize each language separately. Then transform the features of languages into one common dimensional space. Several mathematical and statistical methods can be used to obtain a common latent space (Leban et al., 2014).

Supervised approaches are not cost-effective. They require training (labeled) data in order to build a model which is then used to distinguish between duplicate and non-duplicate reports.

Word Embedding models like word2vec (Mikolov et al., 2013) provide numerical vectors to represent words and phrases. Multilingual Word Embedding models (Ammar et al., 2016) are trained to minimize the distance between words in a language (A) and their translations in a target language (B). *Semantic concepts/topics* are addressed in articles using words and phrases. When using Multilingual embeddings, these concepts are reduced to vectors represented in a unified dimensional space regardless of the used language. So, similarity between two documents becomes meaningful.

By exploiting the distribution of words in reports, we can measure the similarity between a pair of reports written in different languages. We propose our purity-based distance score, a score that measures the ratio of concepts mentioned in one report but not in the other with respect to the total number of mentioned concepts. We use density-based clustering to group the two documents' words in clusters. Each cluster can be seen as a concept. Its members (words) are mentioned in one or both documents. We call a cluster *pure* if all its members come strictly from one document. Otherwise, it is impure. If the clusters of a pair are mostly pure, this means that the reports are addressing different concepts. This indicates that the pair are not duplicates. On the other hand, having a large number of impure clusters indicates addressing common concepts and hence being duplicates.

We tackle the following main challenges:

i. Labeled data is scarce so our approach should be unsupervised

ii. The features used to measure the distance between reports should be language-agnostic features and should work regardless of the languages character set (e.g., English vs. Arabic).

We have the following contributions:

- We propose an unsupervised method to measure cross-lingual similarity between text documents.
- We applied our method to detect news reports discussing the same events across different languages.
- We show that our approach outperforms other approaches.

1.5 Outline of the dissertation

This dissertation discusses various challenges in data stream mining and duplicate detection, and proposes unique and efficient solutions to address those challenges. The rest of the dissertation is organized as follows.

Chapter 2 discusses related works in data stream classification and duplicate detection. First, we discuss works related to novel class detection and how our approach is different from other approaches. Second, we discuss the state-of-the-art data stream classification techniques that address the concept-drift problem, and the advantages of change point detection over those techniques. Third, we discuss different approaches to detect duplicate news reports in online stream fashion. Finally, we discuss work related to unsupervised duplicate detection.

Chapter 3 discusses our approach to novel class detection using unsupervised deep embedding and CPD. The first section provides a brief background about autoencoders and unsupervised deep embedding. The second section explains the details of our approach. The rest of the chapter shows the evaluation and discussion of our approach.

Chapter 4 studies the classification of binary class multidimensional data streams using CPD. We start by describing the problem. Then we explain our approach. Finally, we study the performance of our approach in comparison with the current state-of-the-art.

In Chapter 5 we describe our supervised method to detect duplicate news reports in data stream.

Chapter 6 discusses unsupervised duplicate detection and presents our method to measure distance between text reports.

We conclude in Chapter 7 by summarizing our proposed approaches and by giving pointers to future work.

CHAPTER 2

LITERATURE SURVEY

In this chapter we survey a wide range of existing approaches for stream classification and distinguish our work from the prior works.

2.1 Unsupervised deep embedding for novel class detection

Our novel class detection technique differs from traditional one-class novelty detection techniques (Markou and Singh, 2003) that can only distinguish between normal and anomalous data. Traditional novelty detection techniques assume that there is only one normal class, and any instance that does not belong to the normal class is an anomaly/novel class instance. Therefore, they are unable to distinguish among different types of anomalies. Our approach overcomes this limitation by employing a multi-class framework for the novelty detection problem, which can distinguish between different classes of normal and anomalous behavior, and discover new emerging classes. In addition, traditional novelty detection techniques identify data points as outliers that deviate from the normal class. Conversely, our approach discovers whether a group of such outliers constitutes a new class by displaying strong cohesion. Therefore, our approach synergizes with a multi-class classification model and a novel class detection model.

Unlike other novel class detection methods (cf., (Masud et al., 2011; Al-Khateeb et al., 2016)), our approach does not divide data steams into fixed-size chunks, but instead uses a sliding window. This allows our approach to capture concept-drift immediately. Many approaches that use a sliding window (e.g., (Bifet and Gavaldà, 2009)) detect changes in classification error, and hence require true labels of classified instances (i.e., supervised). To minimize dependency on true labeling, ECHO (Haque et al., 2016) estimates the classifier's confidence values in an unsupervised way without requiring true labels. Assuming that confidence values can capture concept-drift, ECHO monitors the change in these values. However, the user must have prior knowledge about the distribution family of the confidence values. Our change detection method monitors the distributional changes in the original data stream, so it requires no prior knowledge about the distribution, and does not require true labels.

Prior work (Chandra et al., 2016) has proposed a method to classify multi-stream data points. The method applies supervised and unsupervised change point detection techniques. The supervised technique is used for labeled streams by monitoring changes in the labels. On the other hand, the unsupervised one is used for unlabeled streams by monitoring changes in the distribution of classifier confidence values (as in ECHO). Novel class detection is not addressed in this prior work.

None of the previous approaches have considered deep learning in novel class detection over data stream. However, several papers have proposed feature learning with autoencoders to perform different tasks in various domains. In image processing, higher level representations learned by stacked DAEs help boost the performance of SVMs (Vincent et al., 2010). Adaptive DAEs for unsupervised domains have also been proposed (Deng et al., 2014). Recursive autoencoders have been leveraged to generate vector space representations for variable-sized phrases (Li et al., 2013). Prior work (Sakurada and Yairi, 2014; Mazhelis, 2006) has proposed outlier detection methods using autoencoders (but not for novel class detection). Combining autoencoders with clustering has been shown to outperform typical clustering methods (Song et al., 2013; Xie et al., 2015). Marginalized DAEs (Chen et al., 2014) marginalize out corruption during training with fewer training epochs.

In our approach, we propose to use DAEs rather than other types of autoencoders because of their ability to generate robust features for clustering and novel class detection in data streams, where outliers, noise, and drifts are challenges. DAEs work well in such environments (Xing et al., 2016).

2.2 Binary Class Stream Data Classification for Multidimensional Data using Change point detection

Our approach exploits change point detection technique along with data mining techniques to classify data instances from data streams. In this section, first we intend to look into some related works on change point detection techniques. We will also look into some related data stream classification techniques.

There are many change point detection (CPD) algorithms available in the literature. However, most of the work is based upon certain assumptions. For example, CPD method proposed by Hinkley (Hinkley, 1970) assumes that distribution families before and after the change point belong to known parametric families, e.g., normal, binomial. On the other hand, CPD algorithm proposed by Bhattacharyya et. al. (Bhattacharyya and Johnson, 1968) assumes that the variables after the change are stochastically larger than those before. In other words, distribution families before and after the change point differ only in their levels, e.g., mean or median. M. Baron proposes an online non-parametric change point detection approach in (Baron, 2000) which is free from the above assumptions. This approach uses histogram density estimation procedures to estimate distribution families before and after the change. These assumptions are then used in a CUSUM-type process to estimate the change point with a stopping time.

These above mentioned approaches work on uni-dimensional data only. However, most of the real life datasets are multi-dimensional. In the literature, several change detection methods exist to deal with multidimensional data also. Song et. al. (Song et al., 2007) propose a log-likelihood change detection method, called density test, which makes use of EM algorithm and kernel density estimator to infer the distribution. However, it is efficient for low dimensional data. Kuncheva et. al. (Kuncheva and Faithfull, 2014) apply principal component analysis (PCA) to multidimensional data and keep the components with the smaller variance. Then the approach uses semi-parametric log-likelihood detector (SPLL) (Kuncheva, 2013) to detect the distribution change. However, SPLL loses theoretical precision in order to achieve computational simplicity as the assumptions are rarely met. This makes it difficult to set up a threshold.

Extracting useful information efficiently from data stream has become a research focus since data streams are becoming more and more common in today's connected digital world. Masud et. al. propose an efficient stream mining approach ECSMiner (Masud et al., 2011), where the authors proposed techniques to find novel classes over stream. It uses hyper-spheres to capture the decision boundary for classes as the stream is processed. They divide the stream into equal sized chunks (fixed length chunk). In other words, they have not considered dynamic length chunk size to address concept drift issues. SluiceBox (Parker et al., 2012) is a method for data stream mining which builds a hierarchy of ensemble classifiers. They divide stream data into equal chunk size (fixed one) and do classification.

Bifet et. al. propose MOA: Massive Online Analysis (Bifet et al., 2010), which is a framework for stream classification and clustering. It implements a wide range of classification and clustering methods for data streams. Bifet et. al. (Bifet et al., 2013) present a framework to implement multiple change point detection methods using MOA. ADWIN (Bifet and Gavaldà, 2009) is a change detection method that uses only one parameter (confidence bound). (HAT-ADWIN) is an adaptive learning method that uses ADWIN to detect changes and update the model. SC^2P differs from ADWIN in the following ways: First, SC^2P exploits entirely nonparametric change point detection; ADWIN uses CUSUM and EWMA as non-parametric methods for detecting specific changes in the location parameter - drifts. However, the drift detection method uses Binomial distribution so it is parametric. Second, on one hand, SC^2P does not use threshold and strives to detect change point immediately or with minimum delay. On the other hand, ADWIN may encounter delay to reach the threshold or confidence. Experimental results also show that SC^2P outperforms HAT-ADWIN.

Our approach uses Change Point Detection (CPD) to classify instances in data streams. Our approach incorporates both statistical CPD algorithm and classical data mining methods.

It does not need to divide the stream into equal chunks to address infinite length which may lead to poor performance. The proposed approach detects change point on multi-dimensional data without reducing dimensionality or using subspace. Moreover, as our approach uses decision tree algorithm, each dimension contributes to the final label prediction according to its information gain. Experimental results show that this approach can efficiently be used to detect security threats, e.g., classify malicious or spam instances.

2.3 Supervised Duplicate Detection

Duplicate (or near-Duplicate) Detection aims to detect news articles that report the same event. Duplicate Detection is a very important part of event meta-data extraction. Several techniques are applied during the detection process. Authors in (Broder, 2000; Alonso et al., 2013; Uyar, 2009) use shingling to detect duplicates. Manku et. al. (Manku et al., 2007) present a fingerprinting approach. Xiao et. al. in (Xiao et al., 2011) proposed a similarity measure that can be used to compare between a pair of documents. Steorts et. al. in (Steorts and Fienberg, 2014; Steorts, 2015) proposed a Bayesian unsupervised approach to detect duplicate records (e.g., patient records). However, it is not designed to detect duplicates in large unstructured text like news reports. Our approach is significantly different from other approaches. Unlike other approaches, we treat the problem as a stream of documents. Therefore, we deal with several stream mining issues like incremental (online) learning which keeps the model updated hence solving any concept drift that may occur during the streaming. None of the other approaches have considered stream environment nor concept drift. Moreover, we use both linguistic and non-linguist features to deal with unstructured text. In addition, our approach significantly minimizes the human-labeling effort.

2.4 Unsupervised Duplicate Detection

Event data analysis is being carried out over decades. Azar, et al. (Azar, 1980) in their COPDAB project have used political event data to study foreign policy. Baum, et al. (Baum and Zhukov, 2015) have studied political event data related to armed conflict.

Very few frameworks exist to event extraction from news articles. Schrodt and Van Brackle (Schrodt and Van Brackle, 2013) show a complete picture of generating events from raw news text. They describe the life cycle of event generating, key pre-processing steps and also real-time coding with existing NLP tools. Schrodt and Van Brackle(Schrodt, Beieler, and Idris, Schrodt et al.) also show a framework (EL:DIABLO) that consists of both CoreNLP, and PETRARCH. It shows the comparison between TABARI and PETRARCH.

Duplicate Detection aims to detect news articles that report the same event. Duplicate Detection is very important part of event meta-data extraction. Several techniques are applied during the detection process (Alonso et al., 2013; Gibson et al., 2008). Authors in (Broder, 2000; Alonso et al., 2013; Uyar, 2009) use shingling to detect duplicates. Manku et. al. (Manku et al., 2007) presented a fingerprinting approach. Xiao et. al. in (Xiao et al., 2011) proposed a similarity measure that can be used to compare between a pair of documents, focusing on the speed aspect of join operation on database records. Steorts et. al. in (Steorts and Fienberg, 2014; Steorts, 2015) proposed a Bayesian unsupervised approach to detect duplicate records (e.g., patient records). However, it is not designed to detect duplicates in large unstructured text like news reports.

All aforementioned approaches do not tackle duplicate detection across multiple languages. Several multilingual duplicate detection methods identifies duplicate websites or wikipedia pages (Rinser et al., 2013; PR and MS, 2015; Pamulaparty et al., 2013) using webpage meta-data. Webpage meta-data could not be used in the problem we are addressing. We aim to determine whether two text articles are duplicate regardless of whether these articles are webpages or not. So, we can not use webpage meta-data in our analysis.

Plagiarism detection shares similar characteristics with duplicate detection. Alzahrani et. al. (Alzahrani et al., 2010) developed a web-based system to detect cross-lingual plagiarism. The system reduces query document by summarization. The Summary is translated to English. Then similar web resources are detected. Duh et. al. (Duh et al., 2013) proposes an approach to identify new passages in source document with respect to target document and suggest positions to place their translations. Gottschalk et. al. (Gottschalk and Demidova, 2017) developed methods to link text passages written in different languages and containing overlapping information. The authors used Named entities and text translation to English as features to estimate the similarity between documents. These approaches use text translation as part of the process of obtaining a common comparison space. However, since text translation is a challenging task, it may lead to high error rate. Our approach uses multilingual word embedding instead of relaying on machine translation tools. Ferrero et. al. (Ferrero et al., 2017) proposed methods for cross-lingual plagiarism detection using word embeddings. These methods require training using decision tree or weights optimization, hence they are supervised methods. Our approach is unsupervised.

Our approach determines duplicate reports by calculating purity-based distance. We cluster multilingual word embeddings to capture the distribution of semantics in documents. The closest work to our approach is Event registry (Rupnik et al., 2016; Leban et al., 2014). However, unlike our problem, Event registry detects duplicate groups of reports. Each group consists of reports having the same language (hence, a group is monolingual). Similarity score is calculated between different groups of different languages. Event registry have proposed multiple methods to transform documents features from source language to target. These methods include using CCA and SVD. Our goal is to detect duplicates between individual reports. We adopted multilingual word embeddings technique (Smith et al., 2017) that transforms embeddings of source language to target language using SVD. Moreover, we applied CCA as one of the baseline approaches.

CHAPTER 3

UNSUPERVISED DEEP EMBEDDING FOR NOVEL CLASS DETECTION

In this chapter we discuss our novel class detection approach (Mustafa et al., 2017) which monitors distributional changes in deep autoencoder features using CPD to detect concept-drift.

3.1 Background: Denoising autoencoder (DAE)

An autoencoder consists of two functions: encoder f and decoder g. The encoding function $f(x) = \sigma(Wx + b)$ encodes input $x \in \mathbb{R}^d$ to a hidden representation $z \in \mathbb{R}^p$. Function $\sigma(s) = (1 + exp(s))^{-1}$ is the sigmoid, W is the weight matrix, and b is the bias. Multiple hidden layers can be added to incorporate deeper features. Let d be the number of dimensions of the input vector x, and let $p < d$ be the number of dimensions of the deepest hidden layer. The decoding function $g(z) = \sigma(W^T z + b)$ decodes z to x'. Autoencoders aim to minimize the reconstruction error. We use cross entropy as the cost function of the reconstruction error:

$$\mathcal{L}(x, x') = -\sum_{k=1}^{d} x_k \ \log(x'_k) + (1 - x_k) \ \log(1 - x'_k)$$

which can be minimized by gradient descent.

DAEs are autoencoders trained to reconstruct a clean input from a corrupted version (Vincent et al., 2010). To create a corrupted version $\widetilde{x}$ of x, we use the **additive Gaussian noise** corrupting method (Chen et al., 2014). This method adds a random value v to each feature in x, in which $\widetilde{x}_k = x_k + v_k$, where $k = [1 \dots d]$ and $v_k \sim \mathcal{N}(0, \sigma^2)$. Encoder f encodes input $\widetilde{x}$ to z. The decoder function g decodes z to x' while minimizing the error of reconstructing the original (clean) input x.

3.2 Novel class detection

This section describes our approach to **detect** novel class data points and **classify** points that belong to existing classes.

A novel class is a new class that has not been modeled by the classifier. Before describing our novel class detector, we give an informal definition of the data stream classification problem. We assume that a data stream is a continuous flow of data $D = x_1, \ldots, x_n$, where x_i is the ith instance (i.e., data point) in the stream. Assuming that the class labels of all the instances in D are unknown, the problem is to predict their class labels. Let y_i and $\hat{y}_i$ be the actual and predicted class labels of x_i, respectively. If $\hat{y}_i = y_i$, then the prediction is correct; otherwise it is incorrect. The goal is to minimize the prediction error. The predicted class labels are then used for various purposes depending on the application.

We maintain an ensemble of multiple classification models. An unlabeled instance is classified by taking a majority vote among the classifiers in the ensemble. Concept-drift is handled by continuously updating the ensemble so that it represents the most recent concept in the stream. The update is performed as follows: As soon as a distributional change is detected, a new model is trained and it replaces the oldest existing model in the ensemble.

Our approach provides a solution to the concept-evolution problem by enriching each classifier in the ensemble with a novel class detector. If all of the classifiers discover a novel class, then arrival of a novel class is declared, and potential novel class instances are separated and classified as members of the novel classes. Thus, a novel class can be automatically identified without manual intervention.

The main concern with data stream classification is building the classification model and keeping it up-to-date by frequently updating the model with the most recent labeled data. Several ensemble-based methods address this concern (Masud et al., 2011; Haque et al., 2016). We present our ensemble-based novel class detection method below, in which each classifier in the ensemble integrates multi-class classification with outlier detection.

Overview Our approach combines several techniques to detect novel class instances, as illustrated in Figure 3.1. The first n_{init} instances of the stream are used to build an initial

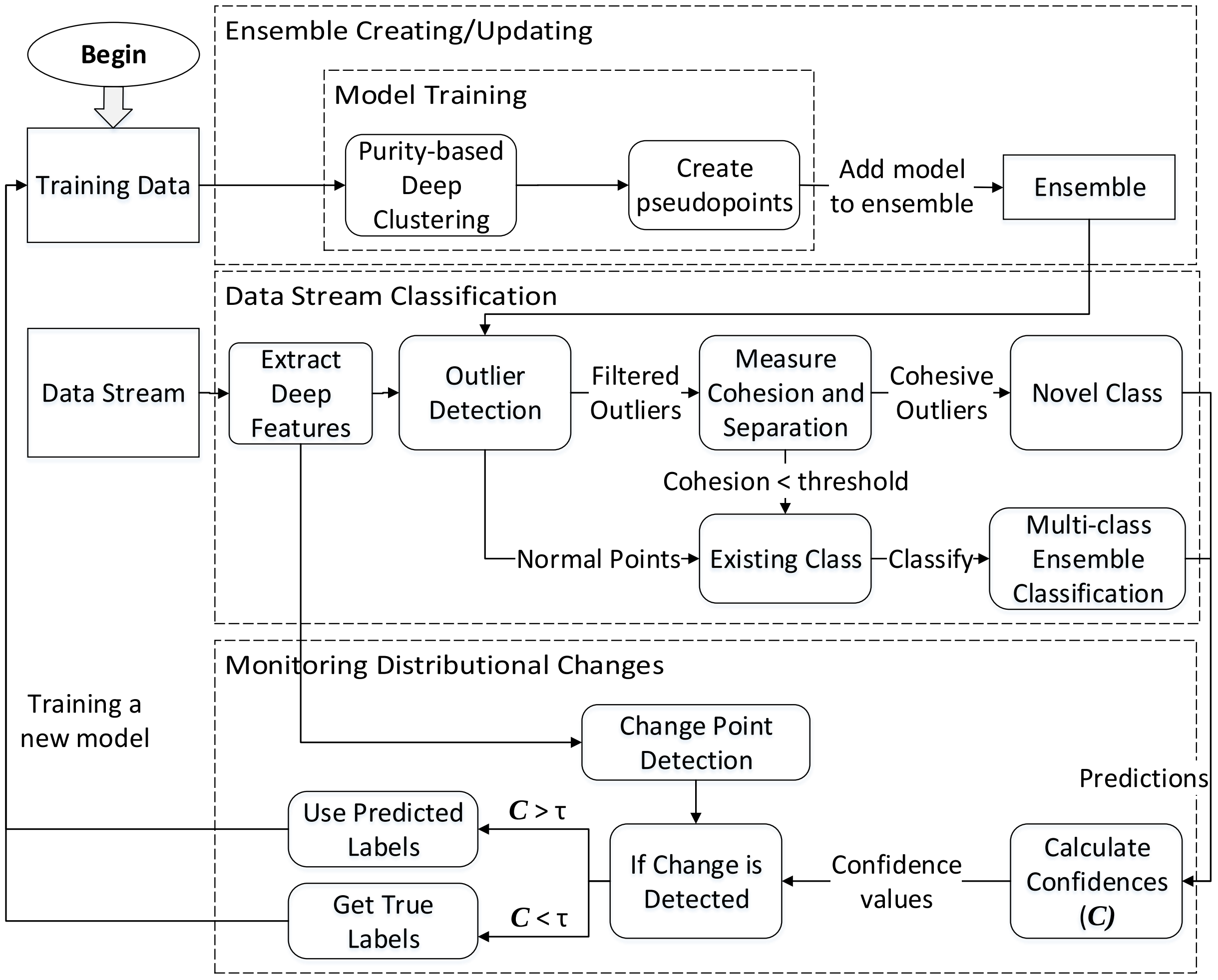


Figure 3.1: Our Approach for Novel Class Detection

ensemble and to train a DAE to extract deep abstract features. These instances are labeled. We apply *purity-based (or semi-supervised) deep clustering* to cluster these instances (see Figure 3.1). This clustering method uses deep abstract features extracted using the DAE. It aims to minimize both intra-cluster dispersion and cluster impurity. We keep a summary of the boundaries of each resultant cluster, called a *pseudopoint*. It consists of centroid, radius, and class frequencies. We use pseudopoints as models for both classification and outlier detection. When an instance is outside the boundaries of all pseudopoints it is labeled as an outlier.

Pseudopoints can be used for classification of existing classes. When an instance x is inside a pseudopoint, x takes the label of the most frequent class in the pseudopoint. Multiple chunks of instances create multiple models. An *ensemble* combines the results of these models. These models are replaced with new ones once changes in data distribution are detected.

During the classification of new unlabeled instances (the data stream in Figure 3.1), we use the ensemble to *detect outliers*. If all models in the ensemble declare an instance x an outlier, we call x a *filtered outlier*. Next, a silhouette coefficient measures the *cohesion* and *separation* of the filtered outliers. Instances that are not detected as outliers or have a silhouette coefficient less than a specified threshold are considered members of an *existing class* and classified using the ensemble (i.e., using the most frequent class of the nearest pseudopoint). We then calculate the confidence $\mathcal{C}$ of each classification.

While we classify new data points, a nonparametric multidimensional change point detection procedure keeps monitoring the incoming data. Once a change in the distribution of the data is detected, a set of new training data is formed using the recently classified instances to build a new model. This set of new training instances is labeled using two ways. The instances with low confidence values are labeled using their true labels. High confidence instances, on the other hand, are labeled using predicted labels. This labeling process reduces the dependency on human annotation because the use of the predicted labels minimizes the amount of true labeling, which requires human effort. The newly built model replaces the oldest one in the ensemble.

The rest of this section details the approach.

Ensemble Creation We build an ensemble of classifiers with the first n_{init} labeled data points, and we keep updating the classifiers with the new data. Each classifier in the ensemble uses a k-NN type classification model. A naïve approach to build such a model is to store all the training data in memory. However, this is inefficient and does not scale to real operating environments. To optimize space utilization and time performance, our approach uses a semi-supervised (purity-based) clustering technique based on Expectation Maximization (E-M)

(Masud et al., 2011), which minimizes both intra-cluster dispersion and cluster impurity, and caches a summary of each cluster (centroid and frequencies of data points belonging to each class), discarding the raw data points. We call a cluster's summary a *pseudopoint*.

We cluster data using their deep abstract features learned from the original data features using the DAE.

Deep Feature Extraction using the DAE To extract deep features, we compute DAE weights (W and b) from the original input features by training the DAE with the instances of the first n_{init} data. We keep the learned W and b to transform the feature values of the rest of stream instances, and denote the transformation of the features of instance $x \in \mathbb{R}^d$ to $z \in \mathbb{R}^p$as $z = \sigma(Wx + b)$.

Note that, we learn the weights (W and b) using the first few chunks and we do not update them during the stream, because updating DAE weights requires performing more time-consuming back propagation. This makes our approach faster, especially since the number of transformed features is significantly less than the original. The ensemble models and all our techniques use data points in deep feature dimensional space.

Outlier Detection using Pseudopoints Figure 3.2 illustrates pseudopoints of one model in the ensemble. The axes represent features. There are 4 groups of pseudopoints (A, B, C, and D). Each pseudopoint is labeled based on the most frequent class label. The pseudopoints of groups A and B are labeled *negative* because most of the labeled instances located inside its boundaries (i.e., during clustering) are negative. Similarly, the pseudopoints of groups C and D are labeled *positive*.

When a new unlabeled test instance emerges, the ensemble is used to classify the instance. A classifier labels the instance based on whether it is inside or outside the decision boundaries of the pseudopoints. For example, in Figure 3.2, X^i is located inside the boundaries of a pseudopoint, unlike X^j. We therefore call X^j a raw outlier (or *Routlier*). If all classifiers of the ensemble label an instance a Routlier, we identify the instance as a filtered outlier (or *Foutlier*) (see Figure 3.3). If the test instance is identified as a *Foutlier*, it is temporarily stored in a buffer *buf* for further inspection. Otherwise, if it is not *Foutlier*, it is classified as one of the existing classes using k-NN (i.e., the most frequent class of the nearest pseudopoint).

Detecting a Novel Class The buffer *buf* is periodically checked to see whether a novel class has appeared. The central concept of our novel class detection technique is that the data points belonging to a common class should be closer to each other (cohesion) and should be far apart from the data points belonging to other classes (separation). When *buf* is examined for novel classes, we look for strong cohesion among the outliers in *buf*, and large separation between the outliers and ensemble's pseudopoints. If such strong cohesion and separation is found, we declare a novel class. We estimate the q-Neighborhood Silhouette Coefficient, or q-NSC (Al-Khateeb et al., 2016). This is defined based on the q, c-neighborhood of a *Foutlier* x ($q, c(x)$ in short), which is the set of q instances from class c that are nearest to x. Parameter q is user defined.

Let $\bar{D}_{c_{out},q}(x)$ be the mean distance of a *Foutlier* x to its q-nearest *Foutlier* neighbors. Also, let $\bar{D}_{c,q}(x)$ be the mean distance from x to $q, c(x)$, and let $\bar{D}_{c_{min},q}(x)$ be the minimum among all $\bar{D}_{c,q}(x)$ for existing classes c. In other words, q, c_{min} is the nearest existing class neighborhood of x. Then q-NSC (Masud et al., 2011) of x is given by:

$$q\text{-NSC}(x) = \frac{\bar{D}_{c_{min},q}(x) - \bar{D}_{c_{out},q}(x)}{\max(\bar{D}_{c_{min},q}(x), \bar{D}_{c_{out},q}(x))} \tag{3.1}$$

q-NSC considers both cohesion and separation, and yields a value between -1 and $+1$. A positive value of q-NSC indicates that the *Foutliers* are closer to other *Foutlier* instances stored in the buffer (more cohesion), and farther away from the instances from existing classes (more separation). The q-NSC(x) value of an *Foutlier* x must be computed separately for each classifier M_i in ensemble $\mathcal{M}$. We declare emergence of a novel class if we find at least $q' > q$ *Foutlier*s having a positive q-NSC score for all the classifiers $M_i \in \mathcal{M}$.

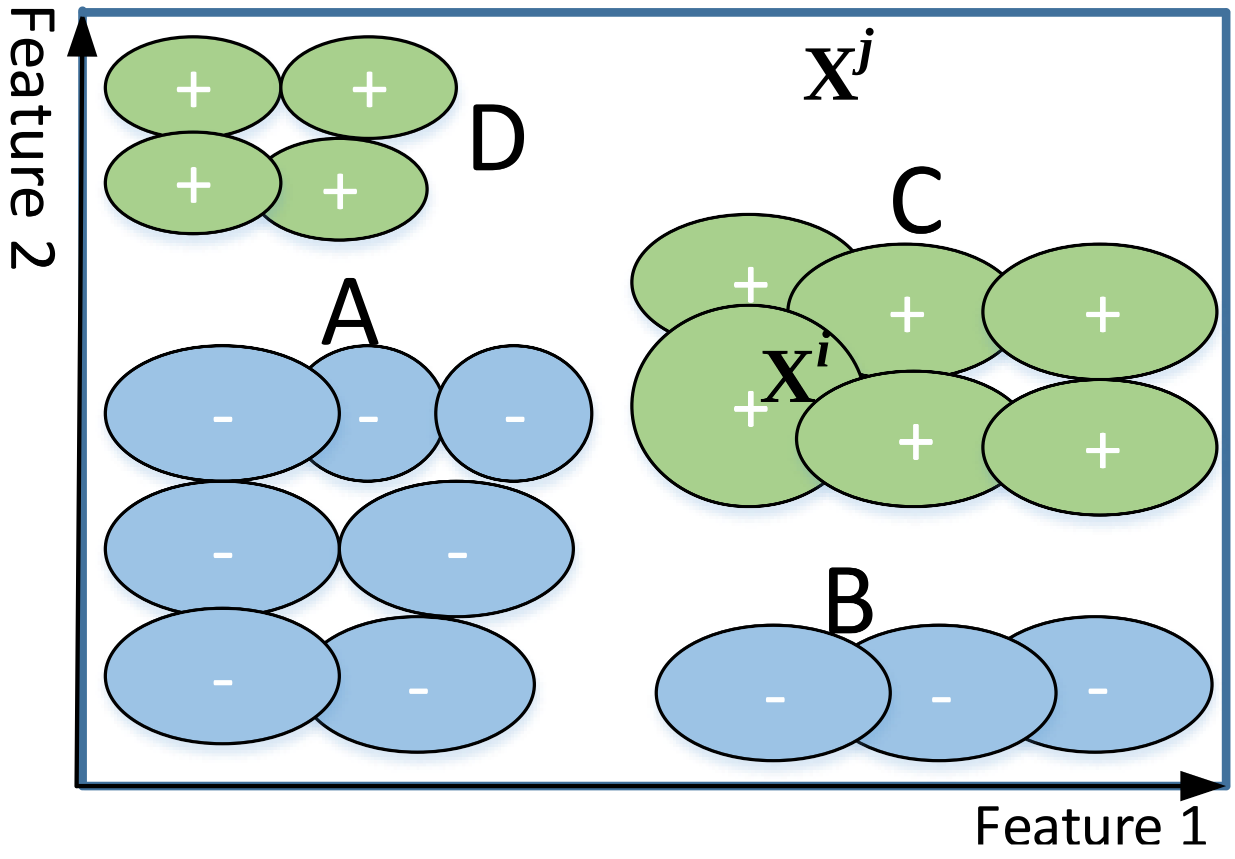


Figure 3.2: An example of a classifier in the ensemble. The circles represent the pseudopoints.

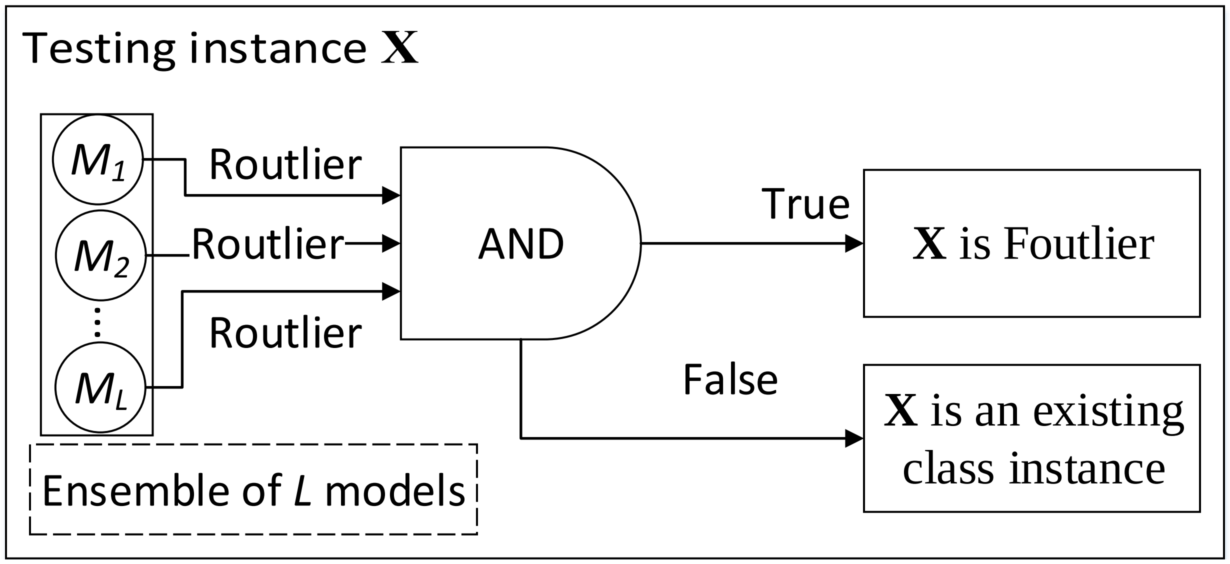


Figure 3.3: Using the ensemble to detect and filter outliers

Ensemble Updating We update the ensemble models when a change in the distribution of the data points is detected. During the classification of any instance x, we estimate the confidence of each individual model $M_i \in \mathcal{M}$ by calculating two measurements: *association* $\mathcal{A}_i$ and *purity* $\mathcal{P}_i$.

- $\mathcal{A}_i = \mathcal{R}(h_{ip}) - \mathcal{D}_{ip}(x)$, where $\mathcal{R}(h_{ip})$ is the radius of the pth pseudopoint (h_{ip}) in model M_i, assuming h_{ip} is the nearest pseudopoint to instance x, and $\mathcal{D}_{ip}(x)$ is the distance between x and the center of h_{ip}.

- $\mathcal{P}_i = \frac{|\mathcal{L}_{ip}(c_m)|}{|\mathcal{L}_{ip}|}$, where $|\mathcal{L}_{ip}|$ is the sum of all frequencies in h_{ip}, and $|\mathcal{L}_{ip}(c_m)|$ is the frequency of the most frequent class (c_m) in h_{ip}.

Once the true labels of instances are available, we create a vector v_i for each model M_i. If M_i classifies instance x correctly, then $v_i^x = 1$; otherwise $v_i^x = 0$. After v_i, $\mathcal{A}_i$, and $\mathcal{P}_i$ are calculated for model M_i over multiple instances, we calculate Pearson's correlation coefficients between v_i and both $\mathcal{A}_i$ and $\mathcal{P}_i$, resulting in values r_A and r_P, respectively. During classification of any instance x, we compute the confidence $\mathcal{C}_i^x$ for model M_i, $\mathcal{C}_i^x = \mathcal{A}_i^x * r_A + \mathcal{B}_i^x * r_P$. Similarly, we calculate $\mathcal{C}_i^x$ for other models M_i in the ensemble. Then the scores are normalized and the average confidence of models is taken $(\mathcal{C}^x)$.

As the data stream instances are being classified, we keep monitoring the distributional changes of the stream data. Nonparametric Change point detection is used (Section 3.3). Once a significant change is detected, we update the ensemble by replacing the oldest model with a new one. The new model is trained using the recently classified instances. We require the true labels of the instances that have confidence values less than a threshold τ. On the other hand, for instances with confidence greater than τ the predicted labels are used. Both sets are then used to train a new classifier.

3.3 Change Point Detection Algorithm

Change point detection (CPD) is used to detect abrupt changes in characteristics of data at unknown time instants. Abrupt changes arise when changes in characteristics occur very fast with respect to the sampling period of the measurements. Change point detection is particularly useful for quality control, system monitoring, and fault detection. We use this technique to detect changes in distribution parameters of the data points. Identifying a change point triggers training a new model to replace the oldest one in the ensemble.

Let $\{x_j\}$ be a sequence of data points, where x_j has a distribution F for $j \leq \nu$ and a distribution G otherwise. The problem of change point detection is to discover a change and to estimate the change point parameter ν from these data. CPD algorithms in general can be divided into two categories: parametric and nonparametric algorithms. Parametric CPD algorithms assume that the distribution families before and after the change point are known beforehand. However, in many applications, although one may know the distribution before the change, when the process is "in control," it is usually impossible to know the distribution after the change, when the process goes "out of control." It may also be the case that distribution families both before and after the change point are unknown. In those cases, nonparametric change point detection algorithms are used to detect the change point in data.

Several nonparametric change point estimation approaches have been proposed in the literature (Ferger, 1991). The pre- and post-change empirical distributions are usually compared for each $k = 1, \ldots, n$, where n is the number of data points. Then the point ν that maximizes some predefined metric or *measure of diversity* between these distributions is used as an estimator of the actual change point ν.

Prior work has established an efficient nonparametric change point detection method based on log-likelihood ratio random walk, which is mathematically proven to achieve high accuracy (Baron, 2000). It can be summarized as follows.

Let $X_1 \dots X_k$ and $X_{k+1} \dots X_n$ be the pre and post-change sequence of data points, where X_i is the ith element in the data sequence. Let $\chi_{1:k}$ and $\chi_{(k+1):n}$ be the histograms (of r number of bins) modeling $X_1 \dots X_k$ and $X_{k+1} \dots X_n$, respectively. For any potential change point k, let $p_{1:k}^{(m)}$ be the ratio of the number of points in bin m of $\chi_{1:k}$ to the total number of points k. Similarly, $p_{(k+1):n}^{(m)}$ is the ratio in bin m of $\chi_{(k+1):n}$. Note that $\sum_m p_{1:k}^{(m)} = \sum_m p_{(k+1):n}^{(m)} = 1$. The log-likelihood ratio is calculated by:

$$S_{k,n} = (n-k) \sum_{m=1}^{r} p_{(k+1):n}^{(m)} \log \frac{p_{(k+1):n}^{(m)}}{p_{1:k}^{(m)}} \tag{3.2}$$

We monitor changes in multidimensional data points ($X_i \in \mathbb{R}^D$) and estimate $S_{k,n}$ for each dimension separately. In other words, for each dimension d, the data values $X_{1:n}^{(d)}$ of dimension d are used to estimate $S_{k,n}^{(d)}$. We refer to the dimensional $S_{k,n}$ as $S_{k,n}^{(d)}$. The maximum $S_{k,n}^{(d)}$ in each dimension d is estimated using the following formula:

$$W_n^d = \max_{\gamma \leq k < n-\gamma} S_{k,n}^{(d)}, \forall d \in D \tag{3.3}$$

where $\gamma = n^\epsilon$ is a cushion distance and $0 < \epsilon < 1$. The maximal W_n^d across all dimensions is calculated (W_n):

$$W_n = \max_{1 \leq d \leq D} W_n^d \tag{3.4}$$

We record the dimension of the maximum change:

$$maxd = \arg\max_{1 \leq d \leq D} W_n^d \tag{3.5}$$

If W_n exceeds threshold h, defined by

$$h = -\log(\alpha) \tag{3.6}$$

then change point ν is declared, where α is the probability of false alarm. A probability of $\alpha = 0.05$ is commonly used in the literature.

$$\nu = \underset{\gamma \leq k < n-\gamma}{\arg\max} S_{k,n}^{(maxd)} \tag{3.7}$$

Algorithm 1 shows our nonparametric multidimensional change point detection algorithm. The algorithm takes α, γ, number of histogram bins (r), and monitored data points ($X_{1:n}$) as input. Lines 6 to 12 represent Equation 3.2. If $W_n > h$, the change is considered significant and the change point is estimated using Equation 3.7 (Line 17), otherwise -1 is returned.

The time complexity of detecting change is $O(\mathrm{D}\times\mathrm{n}\times\mathrm{r})$, where D is the number dimensions, n is the number of examined data points, and r is the number of bins of estimated histogram.

This change point detection technique is suitable for detecting distributional changes in the incoming data points. As the incoming data points keep emerging, we transform the data points into deep abstract feature space and store them in a queue-type sliding window S. We invoke our CPD algorithm to monitor instances in S. Once a change point is detected, we train a new model with the data points in S. However, instead of requesting true labels of all the instances, we get true labels only for the instances with low confidence value. Instances with high confidence are labeled using the predicted class.

3.4 Baseline Approaches

To evaluate the performance of our approach we compare it to other approaches. In this section we present these baseline approaches. The first two are considered the current state-of-the-art.

3.4.1 ECSMiner

ECSMiner (Masud et al., 2011) is an ensemble-based novel class detection approach that divides the data stream into equal-size chunks. The ensemble classifies each chunk and detects

Algorithm 1: Detect-Change (α, γ, r, X)

```
input  : $\alpha$: Probability of false alarm,
         $\gamma$: Cushion period size,
         $r$: The number of bins of histogram,
         $X_{1:n} \in \mathbb{R}^D$: Sequence of multidimensional data points.
output : The change point if exists; -1 otherwise
1  begin
2      $h \leftarrow -log(\alpha)$ // (Equation 3.6)
3      for $d \leftarrow 1 : D$ do
4          for $k \leftarrow \gamma : n - \gamma$ do
5              Estimate histograms $\chi_{1:k}$ and $\chi_{k+1:n}$ (with $r$ bins) using $X^d_{1:k}$ and $X^d_{k+1:n}$,
               respectively.
6              $S^{(d)}_{k,n} \leftarrow 0$
7              for $m \leftarrow 1 : r$ do
8                  Estimate $p^{(m)}_{1:k}$ using $\chi_{1:k}$
9                  Estimate $p^{(m)}_{k+1:n}$ using $\chi_{k+1:n}$
10                 $S^{(d)}_{k,n,m} = p^{(m)}_{k+1:n} \log \frac{p^{(m)}_{k+1:n}}{p^{(m)}_{1:k}}$
11                 $S^{(d)}_{k,n} \leftarrow S^{(d)}_{k,n} + S^{(d)}_{k,n,m}$
12             $S^{(d)}_{k,n} \leftarrow (n - k) * S^{(d)}_{k,n}$ // (Equation 3.2)
13         Calculate $W^d_n$ using Equation 3.3.
14     Calculate $W_n$ using Equation 3.4.
15     Calculate $maxd$ using Equation 3.5.
16     if $W_n \geq h$ then
17         Return $\nu$, where $S^{(maxd)}_{\nu,n} = W_n$
18     else
19         Return -1
```

novel classes. True labels are revealed periodically. Once they are revealed, a new model is trained using true labels. Then the new model replaces the model with the lowest accuracy. Unlike our approach, ECS-Miner does not exploit deep features, and does not detect changes.

3.4.2 ECHO

ECHO (Haque et al., 2016) utilizes change point detection to detect concept-drift. However, this change detection technique monitors the classifier's confidence estimates instead of the original data stream. The assumption is that a change in the classifier's confidence indicates occurrence of a concept-drift. Therefore, if there is a significant change in confidence estimates, a concept-drift is detected and the ensemble is updated. Using this approach requires prior knowledge of the distribution family of monitored data (i.e., confidence values). In other words, the user must identify the data distribution. This work proposes a nonparametric change detection technique, which operates on any data distribution family without the need to specify a particular distribution.

Furthermore, ECHO detects novel classes and classifies the data points in the original input feature space. It does not extract deep features.

3.4.3 One-class SVM Ensemble

In this approach, we build an ensemble of one-class SVM classifiers. One-class SVM classifier is an unsupervised learning method. It learns the decision boundary of training instances and is used to predict whether an instance is inside or outside the learned boundary (Al-Khateeb et al., 2016). We train one classifier for each class. For instance, if our training data consist of instances of C classes, our ensemble must contain C one-class SVM classifiers.

During classification, once a new unlabeled instance x emerges, we classify it using all the one-class SVM classifiers in the ensemble, as follows:

1. If all classifiers predict x as outside the boundary, we label it as a novel class instance.

2. If one classifier predicts x as inside the boundary, we label x as the class of that classifier.

3. If more than one one classifier predicts x as inside the boundary, we label x as the class of the classifier with highest confidence.

We build our ensemble using the first n_{init} chunks. During the classification of the stream, once new labeled (novel class or existing) instances emerge, we train new classifiers (i.e., one classifier for each class of the emerged instances). Then we add them to the ensemble without removing the old classifiers.

3.4.4 Denoising Autoencoder Ensemble

Several previous works have suggested using autoencoders for outlier detection (Sakurada and Yairi, 2014; Mazhelis, 2006). The main idea is as follows. By training an autoencoder with instances of a certain class (e.g., b), we expect that the reconstruction error ($\mathcal{L}$) for any instance of the class b will be less than $\mathcal{L}$ for instances from any other classes (not b). We extend this idea to detect novel class instances in data streams by using an ensemble of DAEs. Each DAE detects outliers from its perspective. The results of the DAEs are combined to determine the class of the tested instance and whether it is a novel class. We present this technique and compare it with our approach.

We build an ensemble of DAEs using the instances of the first n_{init} data points by training one DAE for each class of the C classes in n_{init} data. As a result, our initial ensemble contains C DAEs, where each DAE^b is trained with class b instances. Training is done using gradient descent after applying data corruption as described in Section 3.1. To predict whether an instance x is an instance of class b, we feed forward x in DAE^b and calculate the resulting reconstruction error, which can be computed using cross-entropy ($\mathcal{L}$ in Section 3.1). If $\mathcal{L}$ is less than a threshold h, we classify it as inside the boundary of DAE^b. Otherwise, it is outside (or *Routlier*). In our experiments, we assume that we know the best threshold value.

If all DAEs in the ensemble classify x as Routlier, we classify x as novel class instance. Otherwise, if at least one DAE classifies x as inside, we classify x as the class of the DAE with the minimum $\mathcal{L}$.

The ensemble is updated by adding new DAEs to the ensemble whenever the labels of new instances are revealed. In our experiments we assume the labels are revealed periodically.

Notice that in this approach we use DAEs as outlier detectors, not as feature transformers.

3.5 Results

We have conducted several experiments to evaluate our approach. Our novel class detection approach (here referred to as **NovelDetector**$^{\text{DAE}}$) is described in Section 3.2. We claim that using deep abstract features extracted using DAEs boosts the performance of detecting novel classes. We tested NovelDetector$^{\text{DAE}}$ against the four other approaches described in Section 3.4: **ECS-Miner**, **ECHO**, **OneSVM** Ensemble, and **DAE Ensemble**.

Datasets Our experiments are applied to two synthetic datasets and five real datasets of several types. Table 3.1 summarizes the datasets. We have used MOA (Bifet et al., 2010) to generate synthetic datasets with RandomRBFGeneratorDrift, 7 classes, 70 attributes, 100,000 instances, 12 centroids, and 0.002 drift value for dataset **Syn1** and 0.003 for **Syn2**. We show results on two security datasets: **Packets** and **SystemCalls** (Araujo, 2016; Araujo et al., 2014). These two datasets are from network attacks targeting a web server over a period of time. The goal is to detect the novel and existing attacks. Each one of the two datasets has about 6,000 numeric features and 28 classes (12 benign and 16 attacks). The Packets dataset consists of packets sizes, time, and counts (Al-Naami et al., 2015); and the SystemCalls dataset is system call N-grams. We have additionally tested our approach on Forest CoverType (**Fcover**) and Physical Activity Monitoring (**PAMAP2**) datasets. The latter contains sensory data from 18 different physical activities. Finally, the Internet Movie database (**IMDB**), which consists of textual reviews, is used to predict the authors of the reviews (i.e., author attribution). For all datasets, we normalized features between 0 and 1 and treat 0s and 1s in binary features as numeric. We transformed network packets into feature vectors (Al-Naami et al., 2016).

Table 3.1: Summary of the datasets

Dataset	# Classes	# Dims	# Samples
Syn1	7	70	100,000
Syn2	7	70	100,000
Packets	28	6,000	5,600
SystemCalls	28	6,000	5,600
Fcover	7	54	150,000
IMDB	15	1000	15,000
PAMAP2	6	53	150,000

Data Shuffling In our experiments, novel classes appear gradually. One or more concurrent novel classes may appear along with existing classes. Figure 3.4 shows the distribution of Packets dataset instances and their classes over time. The x-axis represents time (or instance number) and the y-axis records the classes of the instances. As time increases, new instances emerge. Each instance is represented as a point (blue). Each X indicates concurrent novel classes. For example, the first X shows the emergence of novel classes 6 and 7. At the same time, classes 4 and 5 are considered existing classes. At instance number 2,700 (indicated by XX), three concurrent novel classes emerge. Note that old classes in the Packets dataset disappear (Figure 3.4), but in other datasets like Fcover old classes do not disappear.

The Metrics We measured the performance of the approaches by calculating False Positive Rate (FPR%), False Negative Rate (FNR%), and Error (ERR%). False positives are instances incorrectly classified as novel class instances. False Negatives are novel class instances misclassified as belonging to an existing class. ERR is the total classification error, including

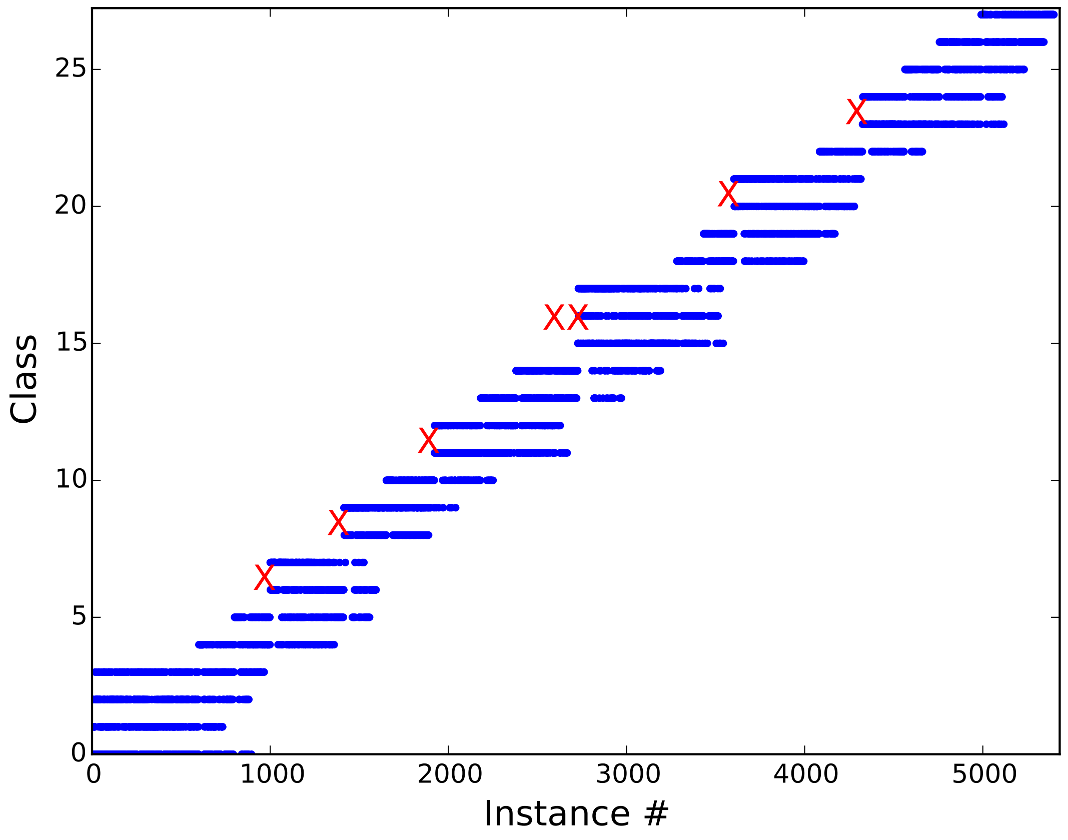


Figure 3.4: Distribution of class instances over time

multi-class misclassifications (e.g., misclassifying instance A as B, regardless of A and B being novel or existing).

The metrics are measured as the average over all points. More specifically, after we classify a test instance, we estimate the accuracy. Later we use the instance in building new classification model during training and update the existing ensemble.

Note that the first n_{init} data points are used to initialize the ensemble. These *labeled* points are not counted in the calculation of our metrics.

NovelDetector$^{\mathrm{DAE}}$ Parameters NovelDetector$^{\mathrm{DAE}}$ was applied with deep abstract features extracted using a DAE with two hidden layers (where the number of features in the first hidden layer is 2/3 the original features, and in the second is 1/3 the first hidden layer) and using additive Gaussian noise where $\sigma = 1.1$. The results are shown in Table 3.2.

Observations NovelDetector$^{\text{DAE}}$ outperforms other approaches in all metrics. FPR, FNR, and the total error are less than 3% on all datasets. On the other hand, the performance degrades using ECS-Miner (e.g., 34.89% FPR, 23.64% FNR, and 49.98% ERR for SystemCalls), whereas ECHO scored 6.11% FPR, 27.5% FNR, and 30.38% ERR. Also, our approach shows significant improvement on results of the synthetic datasets generated with concept-drift. The performance of One-class SVM Ensemble fluctuates depending on the dataset, but is always less than NovelDetector$^{\text{DAE}}$. We tested One-class SVM Ensemble with and without DAE features.

Impact of varying NovelDetector$^{\text{DAE}}$ parameters We have varied the number of DAE hidden layers, number of abstract features in the hidden layers, the cost function, and corruption amount σ^2. We have observed that the cost function significantly affects performance. For example, when using L2 norm instead of cross entropy as a cost function, FPR increases from 2% to 70.31%, FNR to 7.61%, and ERR to 70.59% for the Syn1 dataset. The number of layers also affects performance if we use less than two layers; using two or more layers yields significant improvement. On the other hand, the number of features has less performance impact for NovelDetector$^{\text{DAE}}$. We report the results of applying NovelDetector$^{\text{DAE}}$ with these variations in Table 3.3. Note that d in the table equals the number of original features.

The accuracy of NovelDetector$^{\text{DAE}}$ increases when we increase the amount of corruption in the DAE. We have tested the impact of varying the noise amount by changing the variance σ^2 of the additive Gaussian noise corrupting method. (See Section 3.1 for more details about the corrupting method.) We have evaluated the behavior by testing the performance of NovelDetector$^{\text{DAE}}$ with abstract features extracted using a DAE with one hidden layer on the Syn1 data. We set the number of abstract features to equal the original features. The results are reported in Table 3.4. They show that increasing the amount of noise leads to better performance of NovelDetector$^{\mathrm{DAE}}$. That is, the total error, FPR, and FNR significantly improve when using $\sigma > 1.0$. For example, using $\sigma = 1.9$ yields 0.01% FPR, 0.07% FNR, and 0.02% ERR.

Table 3.2: Summary result on all datasets

Metric	**Novel Class Detector**	**Syn1**	**Syn2**	**Packets**	**SystemCalls**	**Fcover**	**PAMAP2**	**IMDB**
FPR %	NovelDetector$^{\mathrm{DAE}}$	**2.01**	**2.01**	**1.01**	**1.01**	**1.82**	**1.01**	**0.01**
	ECS-Miner	71.04	71.63	26.66	34.89	4.64	18.01	**0.01**
	ECHO	14.90	15.21	**0.01**	6.11	2.83	4.97	**0.01**
	OneSVM (DAE)	5.17	4.09	85.61	71.50	7.16	88.50	79.99
	OneSVM (Original)	70.16	46.05	31.88	45.14	30.74	91.62	46.97
	DAE Ensemble	65.74	38.24	26.31	48.33	28.16	99.01	34.91
FNR %	NovelDetector$^{\mathrm{DAE}}$	**2.01**	**2.34**	**1.52**	**1.30**	**1.05**	**0.02**	**1.01**
	ECS-Miner	2.03	17.43	25.09	23.64	4.38	0.05	29.14
	ECHO	44.27	21.19	91.25	27.50	7.43	0.11	33.00
	OneSVM (DAE)	80.62	98.01	23.47	19.91	93.84	8.25	21.06
	OneSVM (Original)	39.91	61.87	55.94	45.75	28.86	0.01	35.12
	DAE Ensemble	66.31	57.14	63.29	52.13	35.12	0.64	42.26
ERR %	NovelDetector$^{\mathrm{DAE}}$	**2.02**	**2.06**	**1.46**	**1.26**	**1.97**	**1.58**	**2.46**
	ECS-Miner	89.64	86.37	42.53	49.98	6.99	17.38	12.93
	ECHO	31.19	27.49	75.52	30.38	5.09	5.37	9.78
	OneSVM (DAE)	97.52	97.28	78.59	63.08	87.18	94.83	71.05
	OneSVM (Original)	95.85	94.31	64.98	61.90	72.56	94.63	65.17
	DAE Ensemble	95.00	92.21	69.10	62.92	68.91	98.35	61.90

3.6 Discussion

The new abstract deep features generated using the DAE approach significantly boost the performance of novel class detection in our experiments. Moreover, the experiments show that increasing the amount of noise significantly improves DAE accuracy. This noise is randomly sampled from $\mathcal{N}(0, \sigma^2)$. Such noise corrupts the original features and the decision boundaries of the pseoudopoints (clusters). The DAE produces the weights W and biases b, which correct this corruption. These weights produce pseudopoints robust to noise and drift that may occur in the original input data (yielding better generalization). This explains the increase

Table 3.3: The impact of varying DAE parameters on NovelDetector$^{\text{DAE}}$ results

Metric	**Layers**	**Dimensions**	**Cost**	**Syn1**	**Syn2**	**Packets**	**SystemCalls**	**Fcover**	**PAMAP2**	**IMDB62**
FPR %	3	$(1/13)d$	entropy	2.03	2.01	1.01	1.01	1.01	1.01	0.01
	2	$(1/5)d$	entropy	2.01	2.01	1.01	1.01	1.82	1.01	0.01
	1	d	entropy	5.10	2.82	1.01	20.05	5.37	1.01	0.04
	1	$(2/3)d$	entropy	7.16	2.57	1.01	21.88	5.38	1.01	0.04
	1	$(1/3)d$	entropy	3.23	2.51	1.01	19.59	3.30	1.01	0.01
	1	d	L2	70.31	43.25	34.13	26.09	6.54	5.88	0.13
FNR %	3	$(1/13)d$	entropy	2.01	2.01	1.52	1.33	1.04	0.04	1.01
	2	$(1/5)d$	entropy	2.01	2.34	1.52	1.30	1.05	0.02	1.01
	1	d	entropy	2.07	20.17	16.77	8.11	1.61	0.04	1.25
	1	$(2/3)d$	entropy	6.92	2.31	16.27	8.90	1.01	0.02	1.27
	1	$(1/3)d$	entropy	2.03	2.03	17.01	4.48	1.02	0.01	1.10
	1	d	L2	7.61	52.54	17.48	22.69	3.16	0.05	32.14
ERR %	3	$(1/13)d$	entropy	2.06	2.03	1.46	1.28	1.06	1.61	2.5
	2	$(1/5)d$	entropy	2.02	2.06	1.46	1.26	1.97	1.58	2.46
	1	d	entropy	6.01	4.17	5.23	22.72	5.40	1.12	1.17
	1	$(2/3)d$	entropy	8.56	3.00	5.21	20.81	5.33	1.10	1.18
	1	$(1/3)d$	entropy	3.92	2.73	5.32	16.11	3.13	1.01	1.04
	1	d	L2	70.59	48.04	41.02	44.74	7.45	5.87	14.93

Table 3.4: The impact of variance σ^2 in additive Gaussian noise when applying NovelDetector$^{\text{DAE}}$ on Syn1 data

σ^2	**FPR%**	**FNR%**	**ERR%**
0.05	92.30	2.13	93.15
0.1	86.58	8.80	90.30
0.3	70.13	40.66	71.71
0.5	18.51	41.09	22.70
0.7	9.75	41.62	12.29
0.9	6.63	41.78	9.42
1.1	5.1	2.07	6.01
1.3	2.5	2.03	3.01
1.9	0.01	0.07	0.02
2.9	0.01	0.07	0.01
4.9	0.01	0.07	0.01
5.9	0.01	0.07	0.01

in the accuracy of NovelDetector$^{\text{DAE}}$ on all the tested datasets, and mainly the synthetic data which are generated using a random radial basis function with drift.

Since the DAE reduces the number of dimensions significantly, novel class detection becomes much faster. Note that the DAE learns weights only one time (during the first n_{init} chunks) without updating or retraining. In other words, DAE training can be done offline as a warm up phase while building the initial ensemble. So, DAE training time does not negatively affect the processing time after the first n_{init} chunks.

In our experiments, the nonparametric change point detection technique monitors the data in deep abstract feature space. However, the user can choose to detect changes in the original input space or can monitor other variables, such as the distribution of the classifier's confidence values or class labels (if available).

The results of NovelDetector$^{\text{DAE}}$ on the security datasets demonstrate that we can successfully detect novel types of attacks. Our work can be extended to address zero-day attacks. Such attacks have not been modeled by the classifier. However, our approach also identifies new benign classes. In future work, we want to enable NovelDetector$^{\text{DAE}}$ to distinguish between novel attacks and novel benign classes.

CHAPTER 4

BINARY CLASS STREAM DATA CLASSIFICATION FOR MULTIDIMENSIONAL DATA USING CHANGE POINT DETECTION

In this chapter we discuss our approach to classify sequences of binary class data instances using CPD (Mustafa et al., 2014).

4.1 The Problem

Most of the state of the art approaches to classify instances in evolving data stream divide stream data into fixed size chunks to accommodate data into memory and to process without storing data. However, using fixed sized chunks has some disadvantages. First, this may lead to failure of capturing the concept-drift/concept-evolution immediately. Second, quality of model may depend on size of the data chunks. Based on the size of data chunk, it may cause under-fitting or over-fitting problem. So, in this work we focus on using change point detection techniques to fix the chunk size dynamically.

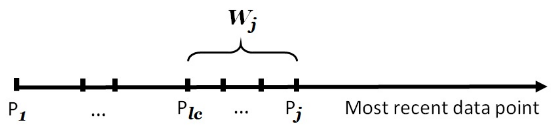
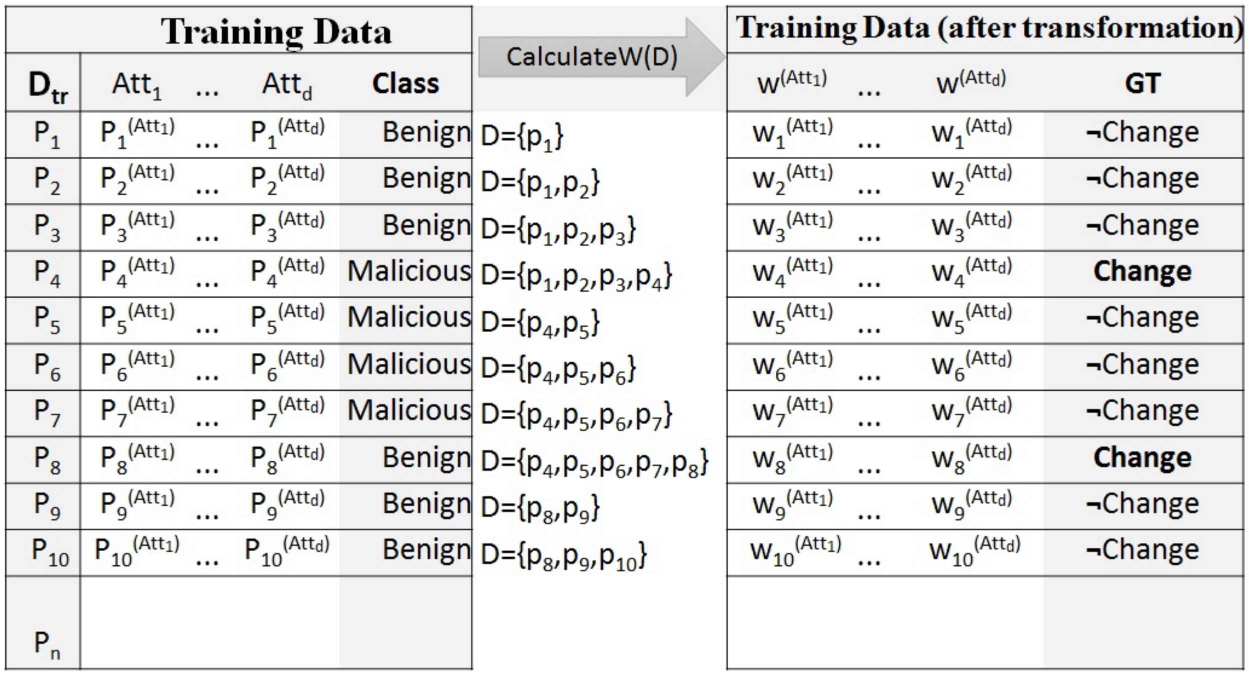


Figure 4.1: Calculation of cusum type process score

Change Point Detection (CPD) algorithms are generally efficient when used on one dimensional data. However, real life datasets typically have a large number of dimensions (e.g., network security datasets). In the case of multi-dimensional datasets, CPD algorithm is typically applied separately on each dimension. Thus, if there are d number of dimensions in the dataset, $w^{(1)} \dots w^{(d)}$ are computed, where $w^{(i)}$ is the score of CUSUM-type process of dimension i using Equation 3.3. A final score $w^{(total)}$ is then computed from the CUSUM-type process scores calculated on individual dimensions. There are several ways to compute $w^{(total)}$.

4.2 Our Approach



Training Data

D_{tr}	Att_1	...	Att_d	Class	CalculateW(D)
P_1	$P_1^{(Att_1)}$	...	$P_1^{(Att_d)}$	Benign	$D=\{p_1\}$
P_2	$P_2^{(Att_1)}$	...	$P_2^{(Att_d)}$	Benign	$D=\{p_1,p_2\}$
P_3	$P_3^{(Att_1)}$	...	$P_3^{(Att_d)}$	Benign	$D=\{p_1,p_2,p_3\}$
P_4	$P_4^{(Att_1)}$	...	$P_4^{(Att_d)}$	Malicious	$D=\{p_1,p_2,p_3,p_4\}$
P_5	$P_5^{(Att_1)}$	...	$P_5^{(Att_d)}$	Malicious	$D=\{p_4,p_5\}$
P_6	$P_6^{(Att_1)}$	...	$P_6^{(Att_d)}$	Malicious	$D=\{p_4,p_5,p_6\}$
P_7	$P_7^{(Att_1)}$	...	$P_7^{(Att_d)}$	Malicious	$D=\{p_4,p_5,p_6,p_7\}$
P_8	$P_8^{(Att_1)}$	...	$P_8^{(Att_d)}$	Benign	$D=\{p_4,p_5,p_6,p_7,p_8\}$
P_9	$P_9^{(Att_1)}$	...	$P_9^{(Att_d)}$	Benign	$D=\{p_8,p_9\}$
P_{10}	$P_{10}^{(Att_1)}$	...	$P_{10}^{(Att_d)}$	Benign	$D=\{p_8,p_9,p_{10}\}$
P_n					

Training Data (after transformation)

$w^{(Att_1)}$	...	$w^{(Att_d)}$	GT
$w_1^{(Att_1)}$	...	$w_1^{(Att_d)}$	¬Change
$w_2^{(Att_1)}$	...	$w_2^{(Att_d)}$	¬Change
$w_3^{(Att_1)}$	...	$w_3^{(Att_d)}$	¬Change
$w_4^{(Att_1)}$	...	$w_4^{(Att_d)}$	**Change**
$w_5^{(Att_1)}$	...	$w_5^{(Att_d)}$	¬Change
$w_6^{(Att_1)}$	...	$w_6^{(Att_d)}$	¬Change
$w_7^{(Att_1)}$	...	$w_7^{(Att_d)}$	¬Change
$w_8^{(Att_1)}$	...	$w_8^{(Att_d)}$	**Change**
$w_9^{(Att_1)}$	...	$w_9^{(Att_d)}$	¬Change
$w_{10}^{(Att_1)}$	...	$w_{10}^{(Att_d)}$	¬Change

Figure 4.2: Example of Training

One way of calculating $w^{(total)}$ is to take the summation of all the one-dimensional scores $w^{(1)}, \ldots, w^{(d)}$

$$w^{(total)} = \sum_{i=1}^{d} w^{(i)}$$

which we refer to as CP-SUM. This is motivated by the joint log-likelihood of multi-dimensional data if components of the observed vectors are independent.To continue controlling for the rate of false alarms, the threshold h will be replaced accordingly by $d * h$.

Another way of calculating $w^{(total)}$ is to take the maximum of $w^{(1)}, \ldots, w^{(d)}$

$$w^{(total)} = \max_{1 \leq i \leq d} w^{(i)}$$

which we refer as CP-MAX. This scheme is obtained by the Bonferroni approach for multiple comparisons, where the probability of a false alarm along at least one dimension is to be controlled at the given level. According to it, a change point will be detected when the most significant of the observed CUSUM values $w^{(1)}, \ldots, w^{(d)}$ exceeds the threshold h.

This final CUSUM-type score $w^{(total)}$ is then compared with a suitably chosen and pre-defined threshold to detect a change point. There are two major problems associated with these techniques. First, these techniques assume that all the dimensions have equal confidence to detect the change point which is not the case in practical situations. For example, in cyber security datasets, where malicious instances typically appear in between sequence of benign instances, different dimensions, i.e., features may have different discriminating abilities. Some features have similar values across all the benign and malicious instances. On the other hand, some features may have substantially different range of values in malicious instances than in benign instances. These later features are more useful to detect change between benign to malicious or vice versa for having more ability to discriminate between different classes. So, discriminating features should be given more importance, i.e., weight while detecting the change point. Second, we need to manually set a threshold to compare with the $w^{(total)}$ to detect the change point. However, it is difficult to set a threshold as both the range and the rate of change for $w^{(total)}$ are unpredictable.

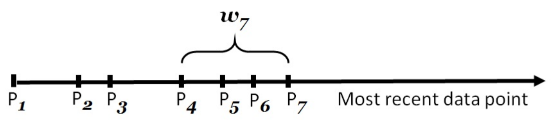


Figure 4.3: Example of Training on timeline

We propose a novel Stream Classification using Change Detection (SC^2D) approach which uses the idea of non-parametric change point detection method (described in Section 3.3). SC^2D goes through two phases, i.e., training and testing. In the training phase, it calculates a matrix W on training data points. Each row of this matrix is a vector which corresponds to CUSUM type process scores calculated on different dimensions of a data point. For example, the i^{th} row of matrix W is a vector W_i which corresponds to training point P_i. W_i consists of scores $w_i^{(1)} \dots w_i^{(d)}$ where d is the number of dimensions. Furthermore, our approach assigns either *change* or *no change* as a ground truth label to each training data point. If the class of data point P_i is same as the class of data point P_{i-1}, SC^2D assigns label *no change* to point P_i to indicate this as not a change point. On the other hand, if the class of data point P_i is different than the class of data point P_{i-1}, SC^2D considers point P_{i-1} as a change point and assigns label *change* to it. For instance, a malicious training data point is labelled as change point if it is preceded by a benign data point. While calculating W_j vector for any point P_j, SC^2D considers only the points P_{lc} to P_j as shown in Figure 4.1, where P_{lc} is the last change point.

After calculating matrix W and assigning corresponding ground truth labels to each of the training data points, a decision tree model is built using both W and assigned ground truth labels. To build the decision tree, each dimension of the matrix W is divided into multiple partitions. For each of these partitions, decision tree computes the information gain. Then the decision tree is built by selecting the dimensions in the ascending order of information gain along with the corresponding partitions.

An example of a training phase on a network security dataset is shown in Figure 4.2. Let's assume that this dataset contains data points from two classes, i.e., Benign and Malicious. Each point P_i in D_{tr} is represented by a vector of attribute values $P_i^{Att_1} \dots P_i^{Att_d}$ and a class label either $Benign$ or $Malicious$. As stated earlier, our proposed approach assigns either *no change* or *change* as label to each data point. *no change* is assigned as ground truth label to data point P_i if its class label is different from previous data point P_{i-1}. Ground truth label *change* is assigned if class label of data point P_i is different than class label of point P_{i-1}. In the given example, data points P_4 and P_8 are change points since these have different class labels than the previous data points. This is why the proposed approach assigns label *change* to each of these data points. These labels are assigned under column GT. Beside assigning labels, the proposed approach calculates CUSUM type process score for each attribute at each point. While calculating this score for a specific point, all the points from the last change point to the current point are considered. For all attributes i of point P_7 $w_7^{Att_i}$ is calculated by considering all the points from last change point P_4 to the current point P_7.

As shown in Figure 4.3, during the testing phase, the decision tree model formed in training phase is used to predict the *change/no change* labels of the testing data points. SC^2D then predicts the class label for each data point based on the prediction of *change/no change* label and class label predicted to the previous data point. Since we focus on data streams with binary classes, if a point is predicted as a change point then its class label is the opposite of the class of preceding data point. Otherwise, current data point is predicted to have the same class as the previous data point.

As described above, a decision tree model is built during the training phase. However, as the data stream evolves, the feature values and class boundaries may change. So, we update the decision tree frequently using the new data. Several approaches can be followed to update the model. One possibility is to append the new labeled data to the old points then use both of them to build a new decision tree. However, the number of data points in the stream is huge. Therefore, storing all the data is not possible. So, instead of keeping all the historical data we store a sample of the old data. Then we use the sample along with the new data to build a new decision tree. Keeping an ensemble of updated models is infeasible because the number of new *change* points is significantly less than the number of *no change* points. In other words, the new data is skewed and does not contain enough *change* points to learn from.

We assume that there exist at least three *no change* points between each two *change* points. Therefore, we do not calculate the change score w for the first three points in training and testing phases. In other words, we leave a cushion of three points because change detection is very sensitive when the distributions are calculated from few points.

Pseudocode for training phase is shown in Algorithm 2. The input is the training dataset D_{tr}. It contains the attribute values and the class labels of each data point. A vector of

```
Algorithm 2: Algorithm for Training
input  : Training Dataset $D_{tr}$, that contains data points from two classes
output : The Decision tree Model $M$
1  begin
2      $D \leftarrow \emptyset$;
3      for each point $P \in D_{tr}$ do
4          $D \leftarrow D \cup P$;
5          $w_p \leftarrow CalculateW\,(D)$;
6          $gt_p \leftarrow IsChangeOfClass\,(P)$;
7          if $gt_p$ is true then
8              $D \leftarrow \{P\}$;
9          $W \leftarrow W \cup w_p$;
10         $GT \leftarrow GT \cup gt_p$;
11     $M \leftarrow TrainDecisionTree\,(W, GT)$;
```

CUSUM type process scores w_p corresponding to point P is calculated at Line 5 using all the points in D where D contains all the data points starting from the last change point to the current point P as discussed above. $IsChangeOfClass\,(P)$ function checks if the class label of point P is different from the class label of the previous point. In other words, this function returns true if the current data point is a change point, otherwise, it returns false. The returned value is stored in gt_p (Line 6). If gt_p is true then current data point P is a change point and D is re-initialized containing only point P (i.e., a new D starting from P) in Line 8. On the other hand, if $false$ is assigned to gt_p, it means that point P is not a change point in the training data. In this way, values of w_p and gt_p are stored in W and GT respectively (Lines 9 and 10). Finally, a decision tree model is trained using W and GT (Line 11).

Pseudocode for testing phase is shown in Algorithm 3. We represent the testing data class labels by $+1$ and -1. In data streams, data points continuously enter into the system. Let D_{new} in Line 4 be the set of newly arrived data points which need classification. For each point P in D_{new}, score W_P is computed in Line 6. While calculating these scores, all the points from the last true change point to the current point are taken into account (see D in Line 5). we use *ResetD* in Line 13 to make sure that D starts from the true change point. W_P is then

Algorithm 3: Algorithm for Testing

```
input  : Decision Tree Model $M$ and set of newly arrived points $D_{new}$
output : label of newly arrived points
1  begin
2      $PrevLabel \leftarrow -1$;
3      $D \leftarrow \emptyset$;
4      for each point $P \in D_{new}$ do
5          $D \leftarrow D \cup P$;
6          $W_p \leftarrow CalculateW\,(D)$;
7          $IsChange \leftarrow ApplyModel\,(M, W_p)$;
8          if $IsChange$ then
9              $Label_p \leftarrow -1 * PrevLabel$;
10         else
11             $Label_p \leftarrow PrevLabel$;
12         $PrevLabel \leftarrow Label_p$;
13         $D \leftarrow ResetD\,(LastTrueChangePoint)$;
14         if $IsUpdateNeeded$ then
15             $S \leftarrow Sample(HistoricalData)$;
16             $R \leftarrow MostRecentpoints)$;
17             $M \leftarrow UpdateModelbyAlgo.2\,(R + S)$;
```

fed into the decision tree model M to detect if P is a change point (Line 7). If it is a change point, P is assigned a label by toggling the label of the previous data point (Line 9). On the other hand, if it is not a change point, then P is assigned the same label as the previous data point (Line 11). Please note that we focus on classifying data streams containing instances from only two classes. Classifying instances of a data stream containing more than two classes is beyond the scope of this work. We intend to investigate it in future work.

4.3 Experimental Results

We have evaluated performance of our approach SC^2D on a number of datasets which are shown in Table 4.1. The names of the datasets are System calls, Synthetic, KDD and PAMAP. In the *Systemcalls* dataset, each instance is a snapshot of system calls collected from a running executable during its first two minutes of execution. Each instance is a vector of continuous real values representing the frequency of system calls during execution. There are a total of 284 possible system calls. Each instance is also labeled as either benign (negative) or malicious (positive). Synthetic dataset is generated using $RandomRBFGenerator$ of MOA (Bifet et al., 2010). KDD dataset is formed by considering the normal instances as negative and other instances as positive from KDD Cup 99 (Bay et al., 2000) dataset. The last dataset PAMAP is formed by considering cycling as negative and lying as positive instances from Physical Activity Monitoring dataset (PAMAP) (Reiss and Stricker, 2012) dataset.

Table 4.1: Sizes of Datasets

Dataset	Training			Testing		
	#Negative	#Positive	#Change Points	#Negative	#Positive	#Change Points
System calls	64,185	64,185	2567	32,092	32,092	1,283
Synthetic	23,324	12,016	706	9,900	5,100	300
KDD	8,356	7,092	309	4,190	3,557	155
PAMAP	8,184	8,178	327	4,216	4,212	169

We have compared performance of our approach (SC^2D) with number of supervised and unsupervised approaches. To compare with supervised approaches, we have used MOA (Bifet et al., 2010) implementation of Hoeffding Adaptive Tree (HAT-ADWIN), Active Classifier, OzaBag, Weighted Majority Algorithms. On the other hand, we have also compared performance of our approach with two unsupervised approaches, i.e., CP-SUM and CP-MAX. We have used two different variations of CP-SUM and CP-MAX approaches. One of the variations is change detection with resetting, where W is calculated by considering the points from the last **true** change point. These variations are named as (CP-SUM$_{wr}$ and CP-MAX$_{wr}$). The other variation is change detection without resetting, where W is calculated by considering the points from the last **detected** change point. These variations are named as (CP-SUM$_{wor}$ and CP-MAX$_{wor}$). Change detection without resetting variations may include cumulative error because the classification is done by toggling whenever a change point is detected. If there is any delay in detecting current change point, this may contribute in delayed detection of subsequent change points as well. For example in Figure 4.2, assume that P_4 was not detected as a change point. All the points from P_4 to P_7 will be misclassified because we did not toggle the class label in P_4. Moreover, missing a true change point may result in missing subsequent change points because the examined data points may now include multiple distributions.

Both of CP-SUM and CP-MAX approaches need a threshold to detect change points. Threshold for CP-MAX approach h_{max} is calculated by taking $-\log(\alpha)$ where α is the probability of false alarm before $W_n = 0$. On the other hand, threshold for CP-SUM approach h_{sum} is calculated by taking $-\log(\alpha) * d$ where d is the number of dimensions. In our experiments, we have used $\alpha = 0.05$ to set the thresholds of the variations of CP-SUM and CP-MAX approaches.

For performance evaluation, we have used True Positive Ratio (TPR), False Positive Ratio (FPR), False Negative Ratio (FNR), and True Negative Ratio (TNR). To evaluate the overall performance of the methods we have used total accuracy and F_β-measure (Equation (4.1)) with $\beta = 2$. In network security domain, the ability to classify malicious data is more important than the ability to classify benign data. Because the misclassified malicious data may have harmful results. Using F_2 allows us to put more emphasis on the successful and failure attempts to protect the host from the malicious data because it weighs both true positive (TP) and false negative (FN) higher than false positive (FP).

$$F_\beta = \frac{(1+\beta^2) * TP}{(1+\beta^2) * TP + \beta^2 * FN + FP} \tag{4.1}$$

4.3.1 Overall Performance

Performance of all approaches in terms of accuracy is shown in Table 4.2. It shows that, our proposed approach SC^2D outperforms all the other approaches by a large margin in case of *System calls* dataset. SC^2D also shows better performance than other approaches on PAMAP dataset, where it classifies all the instances correctly. On KDD and Synthetic datasets also, SC^2D shows superior performance than the unsupervised approaches, i.e., CP-SUM, CP-MAX and competitive performance comparing with the supervised approaches, i.e., HAT-ADWIN, Active Classifier, OzaBag, Weighted Majority Algorithms. Finally, we present the average accuracy of these approaches on all the datasets. SC^2D outperforms all the other approaches by a large margin in terms of average accuracy.

Recall that we focus on a specific scenario where instances from each of the classes come in a sequence, i.e., sequence of instances from a class followed by sequence of instances from another class. *System calls* dataset has instances from two classes, i.e., benign and malicious. Instances from each of the classes come in a sequence. Original Physical Activity Monitoring dataset (PAMAP) dataset also contains sequence of instances from different classes since each of the physical activity, i.e., class label in this case is performed for some time. So, *System calls* and PAMAP datasets comply with the specific scenario we are considering. This is the reason why our proposed approach SC^2D shows very good performance on *System calls* and PAMAP datasets. Moreover, in PAMAP dataset each class has a unique distribution which differentiates each class from other classes. Recall that change point detection is used to find concept drift/concept evolution immediately over evolving stream in our approach. As a result, the number of discovered true positives has increased without adding false positives in case of SC^2D. On the other hand, in original KDD Cup 99 (Bay et al., 2000) and Synthetic datasets do not comply with the above scenario. In these datasets, instances from different classes do not come in sequence, still SC^2D shows competitive performance comparing with the other supervised approaches.

Table 4.2: Experimental Results

Data set	Method	Accuracy	F_2	TPR	FPR	FNR	TNR
System calls	SC^2D	**0.97**	**0.97**	**0.98**	0.04	**0.02**	0.95
	CP-SUM$_{wr}$	0.66	0.66	0.68	0.37	0.32	0.63
	CP-MAX$_{wr}$	0.55	0.54	0.55	0.456	0.443	0.543
	CP-SUM$_{wor}$	0.49	0.38	0.49	0.50	0.50	0.49
	CP-MAX$_{wor}$	0.50	0.39	0.49	0.49	0.50	0.50
	Act-Class	0.74	0.60	0.57	0.10	0.42	0.89
	HAT-ADWIN	0.76	0.68	0.66	0.15	0.33	0.84
	Ozabag	0.66	0.36	0.31	0.03	0.68	0.96
	W-Majorit	0.55	0.03	0.02	0.00	0.97	0.99
Synthetic	SC^2D	0.97	0.97	0.98	0.05	0.01	0.95
	CP-SUM$_{wr}$	0.59	0.54	0.619	0.448	0.38	0.55
	CP-MAX$_{wr}$	0.55	0.49	0.569	0.479	0.43	0.52
	CP-SUM$_{wor}$	0.51	0.46	0.51	0.49	0.49	0.51
	CP-MAX$_{wor}$	0.50	0.45	0.51	0.51	0.49	0.49
	Act-Class	0.98	0.97	0.96	0	0.03	1
	HAT-ADWIN	0.99	0.99	0.99	0	0.00	1
	Ozabag	0.99	0.99	0.99	0	0.00	1
	W-Majorit	0.99	0.99	0.99	0	0.00	1
KDD	SC^2D	0.94	0.94	0.94	0.059	0.058	0.94
	CP-SUM$_{wr}$	0.61	0.603	0.602	0.392	0.398	0.608
	CP-MAX$_{wr}$	0.54	0.55	0.544	0.457	0.456	0.543
	CP-SUM$_{wor}$	0.50	0.50	0.49	0.49	0.50	0.50
	CP-MAX$_{wor}$	0.50	0.50	0.50	0.49	0.49	0.50
	Act-Class	0.98	0.98	0.99	0.01	0.00	0.98
	HAT-ADWIN	0.99	0.99	0.99	0.00	0.00	0.99
	Ozabag	0.99	0.99	0.99	0.00	0.00	0.99
	W-Majorit	0.99	0.99	0.99	0.00	0.00	0.99
PAMAP	SC^2D	**1.0**	**1.0**	**1.0**	**0**	**0**	**1.0**
	CP-SUM$_{wr}$	0.59	0.569	0.56	0.386	0.437	0.614
	CP-MAX$_{wr}$	0.54	0.535	0.534	0.447	0.466	0.553
	CP-SUM$_{wor}$	0.46	0.47	0.47	0.56	0.53	0.44
	CP-MAX$_{wor}$	0.50	0.49	0.49	0.49	0.50	0.50
	Act-Class	0.99	0.98	0.98	0.00	0.01	0.99
	HAT-ADWIN	0.99	0.99	0.99	0	0.00	1
	Ozabag	0.99	0.99	0.99	0.00	0.00	0.99
	W-Majorit	0.99	0.99	0.99	0.00	0.00	0.99

Table 4.2 continued

Data set	Method	Accuracy	F_2	TPR	FPR	FNR	TNR
Average	SC^2D	**0.97**	**0.97**	**0.97**	0.03	**0.02**	0.96
	CP-SUM$_{wr}$	0.61	0.59	0.61	0.39	0.38	0.60
	CP-MAX$_{wr}$	0.54	0.52	0.54	0.45	0.44	0.53
	CP-SUM$_{wor}$	0.49	0.45	0.49	0.51	0.50	0.48
	CP-MAX$_{wor}$	0.50	0.45	0.49	0.49	0.49	0.49
	Act-Class	0.92	0.88	0.87	0.03	0.12	0.96
	HAT-ADWIN	0.93	0.91	0.91	0.03	0.08	0.96
	Ozabag	0.91	0.83	0.82	0.00	0.17	0.99
	W-Majorit	0.88	0.75	0.75	0.00	0.24	0.99

CHAPTER 5

SUPERVISED DUPLICATE DETECTION

In this chapter we discuss our supervised approach for detecting duplicate reports in news data stream.

5.1 Duplicate Report Detection

Figure 5.1 shows the architecture of our approach. It can be divided into three phases: training, testing, and model updating. During training we build our initial classification model using ***similarity tuples*** created by our ***feature extraction*** procedure from labeled training data (see steps 1, 2, and 3). Each similarity tuple is created from a pair of reports. We use the similarity tuples of the first l days of the stream to build the initial model. Then we test our testing data that is chronologically ordered in ***sliding window***. Feature extraction extracts similarity tuples and the classifier classifies the tuples into duplicate or not (follow the dashed arrows from step 4 to step 5). During this classification, the classifier calculates ***the confidence value*** of each classified point. A sequence of points results in a sequence of confidence values. Keep in mind that we need to update the model. So, we track the distributional changes in the sequence of confidence values using ***change detection*** (step 6). Once we detect a change point, we label the data points that emerged after the detected change point in the sliding window using human-labeling. The points that are before the change point are labeled using the classifier. We update the model using both sets (i.e., the human-labeled post-change data and predicted pre-change data) (step 7). In the following subsections we describe our approach.

5.1.1 Sliding Window Approach

Detecting duplicate reports requires comparing the newest published report with past reports. Since storing all the past reports is not conceivable in terms of memory and computations, we

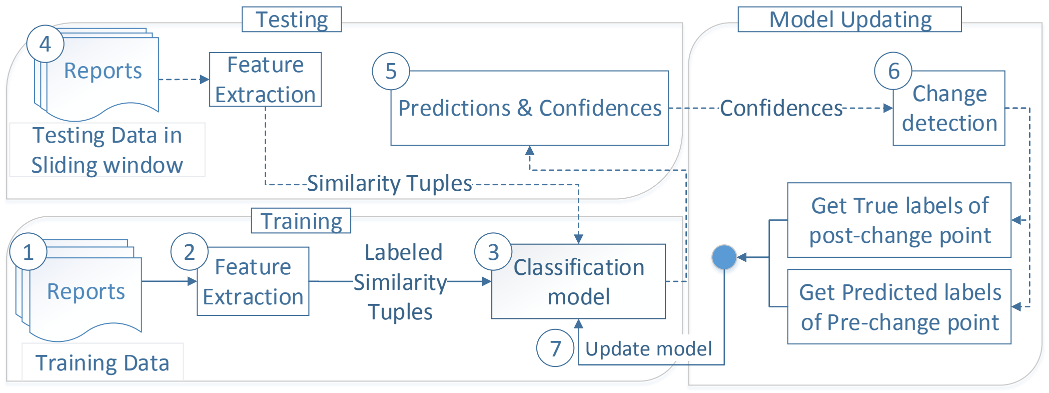


Figure 5.1: The architecture of our approach

use a sliding window. A sliding window is a queue data structure that acts as a time window. It stores the reports published in the past d days. As time goes, the window shifts to include new reports and simultaneously remove the oldest ones. Every report collected daily is tested for duplicate status against every other report from the current day as well as archived reports still within the sliding window.

5.1.2 Feature Extraction

As new news reports stream into the sliding window, they are continuously tested for duplicate status. To test whether two reports are duplicate, we apply the following feature extraction procedure. The procedure is demonstrated in Figure 5.2. First, we create *similarity tuple*, which is a vector of size 5. The five elements are similarity values between named entities mentioned in reports, as the following: (i) Jaccard similarity between locations in the two reports (see Equation 5.1). (ii) Jaccard similarity between person names in the two reports. (iii) Jaccard similarity between organizations in the two reports. (iv) Jaccard similarity between dates in the two reports. (v) The cosine similarity between Tf-Idf Uni-grams of the two reports. Then we concatenate the 5 values in the tuple. We use Clavin-Cliff (D'Ignazio et al., 2014) to extract locations, person names, and organizations from the reports, Clavin-Cliff is a tool for named entities extraction. We extract the dates by matching the story text with date patterns using regular expressions.

$$J\left(L\left(R_i\right), L\left(R_j\right)\right) = \frac{\left|L\left(R_i\right)\ \cap\ L\left(R_j\right)\right|}{\left|L\left(R_i\right)\ \cup\ L\left(R_j\right)\right|} \tag{5.1}$$

where $\left|L\left(R_i\right)\right|$ is the size of the set of locations in report R$_i$. The same equation is applied to person names, organizations, and dates sets by replacing $L\left(R_i\right)$ with $P\left(R_i\right)$, $O\left(R_i\right)$, and $D\left(R_i\right)$ respectively.

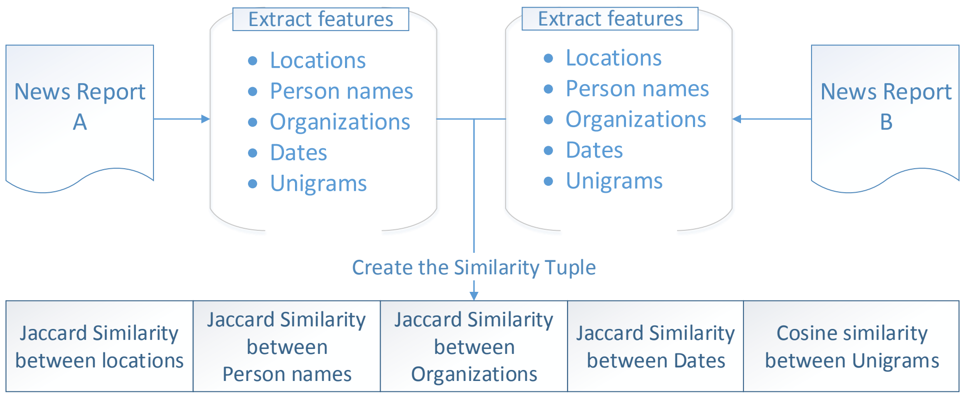

News Report A
Extract features
Locations
Person names
Organizations
Dates
Unigrams
Extract features
Locations
Person names
Organizations
Dates
Unigrams
News Report B
Create the Similarity Tuple
Jaccard Similarity between locations
Jaccard Similarity between Person names
Jaccard Similarity between Organizations
Jaccard Similarity between Dates
Cosine similarity between Unigrams

Figure 5.2: Features Extraction

Using the similarities between named entities as features, the main differentiating characteristics of the reports can be captured and recorded. News reports that talk about the same event typically mention similar sets of locations, person names, organizations, and dates. Conversely, different events usually differ in such features. The similarity tuple uses the sets of named entities and similarities thereof to distinguish between duplicate and non-duplicate reports.

5.1.3 Incremental (Online) Classifier

After creating the similarity tuple, we classify the tuple into duplicate or non-duplicate using incremental (online) logistic regression classifier. In order to implement our classifier we use stochastic gradient descent library of scikit learn python package (Pedregosa et al., 2011) with loss function equal to log.

The values of the features may change over time due to concept drift. The incremental classifier allows us to update the classification model, which leads to a more accurate model. When logistic regression classifies a data point (X_i) into class Y_i=y, it calculates the probability of the predicted class label ($\Pr(\mathrm{Y}_i=y \mid \mathrm{X}_i)$). This probability can be used to indicate the degree of classifier's confidence. To compute this probability, we use Scikit learn function *predict_proba*(X).

5.1.4 Duplicate Detection Over Textual Stream

Logistic regression is a supervised probabilistic classification method. It requires a set of training tuples to create a model which is then used to calculate the probability of predicted class label. For training and testing we use a dataset that is already labeled by human (see Section 5.3 for more details about the dataset).

News reports are continuously emerging. However, the features (the similarity values in the similarity tuples) may change in an unforeseen fashion and new values may appear (concept drift). In other words, changes can happen due to two scenarios: (1) New duplicate may emerge. (2) Existing duplicate may change. So, we use incremental classifier. This approach allows us to update the model frequently. Therefore, new training data is added to the model as the time passes. This makes our model adaptive to concept drift. However, more updates means more data labeling which, typically, requires human effort. We propose an approach that can handle concept drift without requiring all data to be human-annotated. We call it Duplicate Detection using Supervised Approach with Partially Human-Labeled Data (DSPL). In addition, we tested three other approaches: Supervised with Fully Human-Labeled Data (DSFL), Supervised with No Model Updates (DSNU), and finally, Unsupervised Approach (DUNS). The difference between them is based on the amount of human-annotated data (true-labeled) needed to update the classification model. In the following we present our approach and the other three approaches.

D̲uplicate Detection: S̲upervised, P̲artially Human-L̲abeled Data (DSPL)

When classifying a data point into a binary class label using logistic regression, the classifier calculates the probability of the two class label (i.e., positive and negative classes). Then it declares the class label which has the maximum value of the two probabilities as the class of the data point. The probability of the predicted class is also called *'confidence'*. If the probability of a given data point is low, this may indicate that the classifier is not confident. We utilize these probabilities to reduce the dependency on human-labeling. As time goes new pairs of reports are formed (i.e., similarity tuples). In the following description a data point is a similarity tuple.

The new data points are classified and confidence values are calculated by the classifier. These confidence values form a sequence of values. We keep tracking any distributional changes that may occur in the sequence. So, we use the change point detection technique described in Section 5.2. Once a change is detected, we select the points located after the change and we get their true labels from human-annotation. We update the classification model using these human-labeled points along with the pre-change data points labeled by the classifier.

Algorithm 4 shows the model updating in DSPL approach. First, we classify each similarity tuple in the sliding window using logistic regression and get the probability (i.e., confidence) of the predicted class label (Line 2). Then we apply change detection technique to check whether a change occurred in the confidence values of the data points (Conf_t). Note that we keep the confidence values calculated on the previous day (Conf_{t-1}). The change detection tries to find a change point in Conf_t considering the distribution of confidence values in Conf_{t-1}. Using Conf_{t-1} allows the change detection method to look at the past distribution and so gives more accurate results. The algorithm takes four parameters: (i) The *sensitivity* which is the probability that the detected change being a false alarm. It is used to calculate the threshold of the change score. We use 0.05 which is commonly used in the change detection literature. (ii) γ_{Pre} which is the index of the first potential change point to be examined. We assume that the change (if exists) must be in Conf_t not Conf_{t-1}. So, we start searching for change point from the first index of Conf_t. Therefore we set this parameter to the length of Conf_{t-1}. (iii) γ_{Post} which is the index of the last potential change point to be examined. (iv) The *sequence* that may include a change point. Here we pass the concatenation of the two lists of confidences (Conf_{t-1} and Conf_t). **Detect-Change** algorithm is described in Section 5.2 (see Algorithm 5). If the change detection procedure finds a change point (CP) (Line 3), we select all the points that are after the change (T_{postCh}) (Line 4). These points are then labeled by human-annotators (Line 5). Finally, we update the model using both the human-labeled post-change data and model predicted pre-change point data (Lines 6 and 7).

Duplicate Detection: Supervised, Fully Human-Labeled Data (DSFL)

In this approach we use human effort to label all the pairs of reports. In other words, we assume that after classifying the pairs of reports of a certain day, we get the pairs' ground truth and we use it to update the model. Getting the true labels of all pairs may not be realistic.

Duplicate Detection: Supervised, No model Updates (DSNU)

In this approach we build the initial model using the pairs of the first few days, and it remains without updating until the end of the stream.

Duplicate Detection: Unsupervised (DUNS)

In this unsupervised approach, we use a threshold to classify pairs as duplicate or non-duplicate. This approach does not require any human-labeling because we do not build a model. Here,

Algorithm 4: Algorithm for model updating using less number of labeled data

```
input  : A set of similarity tuples $T$ corresponding to the reports in the sliding window
         window, Logistic Regression Model $M$
output : An updated model $M_{new}$
1 begin
2     $(Y_{Pred}, Conf_t) \leftarrow Classify(T, M)$
3     CP $\leftarrow$ Detect-Change(Sensitivity, $\gamma_{Pre}$, $\gamma_{Post}$, [Conf$_{t-1}$, Conf$_t$]) // Algorithm 1
4     $T_{postC} \leftarrow T_{CP}\ \ldots\ T_n$ // CP = the index of the change point in $T$ and $n$ = length of
      $T$
5     $Y_{True\ [\ CP\ \ldots\ n\ ]} \leftarrow getTrueLabel(T_{postC})$
6     $T_{PreCh} \leftarrow T_0\ \ldots\ T_{CP}$
7     $M_{new} \leftarrow$ updateM$(M, T_{PreCh}, Y_{Pred\ [\ 0\ \ldots\ CP\ ]}, T_{postCh}, Y_{True\ [\ CP\ \ldots\ n\ ]})$
```

we classify a pair of reports as duplicate if the value of a specified feature is greater than a pre-fixed threshold set by the domain expert. We tested several of variations using each one of the features in the similarity tuple. For instance, we can use the similarity between locations as a feature. We can also use the similarity between person names, organizations, dates, or n-grams. Moreover, we can use an aggregated value of these similarities like minimum, maximum, or average value of the five features. Then we compare the feature to a threshold and classify accordingly.

5.2 Change Point Detection

Concept drift occurs in a stream when the underlying concept of data changes over time. So, the classification model needs to be updated regularly to adapt to such changes (Parker and Khan, 2015).

We utilize change detection technique (CDT) to detect change in classifier's confidence estimates (Haque et al., 2016; Baron, 1999). A change in the classifier confidence indicates occurrence of a concept drift. Therefore, if there is a significant change in confidence estimates, a concept drift is detected and we update the classifier using the labeled instances of the post-change data.

```
Algorithm 5: Detect-Change ($\alpha$, $\gamma_{Pre}$, $\gamma_{Post}$, $W$)
input  : $\alpha$: Sensitivity, $\gamma_{Pre}$: Pre-Cushion period size, $\gamma_{Post}$: Post-Cushion period
         size, $W$: Sequence of values.
output : The change point if exists; -1 otherwise
begin
    $T_h \leftarrow -log(\alpha)$, $n \leftarrow$size of $W$, and $\omega_n \leftarrow 0$
    for $k \leftarrow \gamma_{Pre} : n - \gamma_{Post}$ do
        Estimate pre and post-beta distributions, $beta(\hat{\alpha_0}, \hat{\beta}_0)$ and $beta(\hat{\alpha_1}, \hat{\beta}_1)$ from
        $X_1 \dots X_k$ and $X_{k+1} \dots X_n$ respectively.
        Calculate $S_{k,n}$ using Equation (5.2).
    Calculate $\omega_n$ using Equation (5.3).
    if $\omega_n \geq T_h$ then
        Return $kmax$, where $S_{kmax} = \omega_n$
    else
        Return -1
```

CDT monitors confidence of the classifier on recent data instances. Confidence scores are generated in the range of $[0, 1]$. We observe from our experiments that generated confidence scores tend to follow a *beta* distribution. We have carried out *Chi-Square* goodness of fit test on the generated confidence scores which also confirm that observation. Therefore, we use a CUSUM (Baron, 1999)-type change detection technique on beta distribution to use in this context. CDT detects any significant change of statistical properties within the values stored in W.

Algorithm 5 sketches CDT. Let n be the size of W, first CDT divides W into two sub-windows for each k between γ_{Pre} to $n - \gamma_{Post}$. We know that each of the sub-windows contains a beta distribution, however the parameters are unknown. So, $X_1 \dots X_k$ and $X_{k+1} \dots X_n$ constitute pre and post-beta distributions respectively, where X_i is the i^{th} element in W. The parameters are estimated using the method of moments. CDT estimates the parameters at Line 4. Let $(\hat{\alpha_0}, \hat{\beta}_0)$ and $(\hat{\alpha_1}, \hat{\beta}_1)$ be the estimated parameters for pre and post-beta distributions respectively. Then, sum of the log likelihood ratios is calculated at Line 5 using the following formula:

$$S_{k,n} = \sum_{i=k+1}^{N} log \left(\frac{f\left(X_i \mid \hat{\alpha}_1, \hat{\beta}_1\right)}{f\left(X_i \mid \hat{\alpha}_0, \hat{\beta}_0\right)} \right) \tag{5.2}$$

where $f\left(X_i|\hat{\alpha}, \hat{\beta}\right)$ is the probability density function (PDF) of the beta distribution having parameters $\left(\hat{\alpha}, \hat{\beta}\right)$ applied on X_i. Next, the cusum score ω_n for the values stored in W is calculated at Line 6 using the following formula:

$$\omega_n = \max_{\gamma_{Pre} \leq k \leq n - \gamma_{Post}} S_{k,n} \tag{5.3}$$

Let $kmax$ be the value of k for which the algorithm calculated the maximum $S_{k,n}$ value where $\gamma_{Pre} \leq k \leq n - \gamma_{Post}$. Finally, a change is detected at point $kmax$ if ω_n is greater than a pre-fixed threshold. CDT calculates the threshold based on the value of the desired sensitivity α. α is the probability that a detected change is a false alarm. In our experiments, we use $-log(\alpha)$ as the threshold value. We use $\alpha = 0.05$, which is also widely used in the literature.

5.3 Dataset

The atrocities event data (Schrodt and Ulfelder, 2009) is a collection of recent news reports talking about atrocities and mass killings in several locations. Human coders have read the reports and extracted metadata about the events reported in these reports. They annotated information like the event's victims, perpetrators, and the reports that reported the event. Multiple reports may mention the same event. We call these reports *duplicate* reports. Our goal is to detect duplicate reports from textual news reports.

5.4 Experimental Results

We apply our approach on the atrocities data collection (Section 5.3). We have calculated the total accuracy, F$_1$, precision, recall, and number of false positive/negative. Note that the duplicate class is the positive class and non-duplicate is the negative class.

We begin by showing DUNS results. This approach eliminates the usage of any training data. However, it requires using a threshold. Different variations of this approach are described in Section 5.1.4. None of them achieved greater than 35 % for F_1 except when using cosine similarity between uni-gram as feature, which is shown in Figure 5.3 with respect to the varied threshold. The figures show that it does not perform as good as our approach because it does not adapt to concept drift and the performance is very sensitive to the threshold. Note that we got similar results using bi-gram and tri-gram. The last two rows in Table 5.1 show the performance of DUNS with both: cosine similarity between unigrams as a feature and max value of the similarity tuple as a feature.

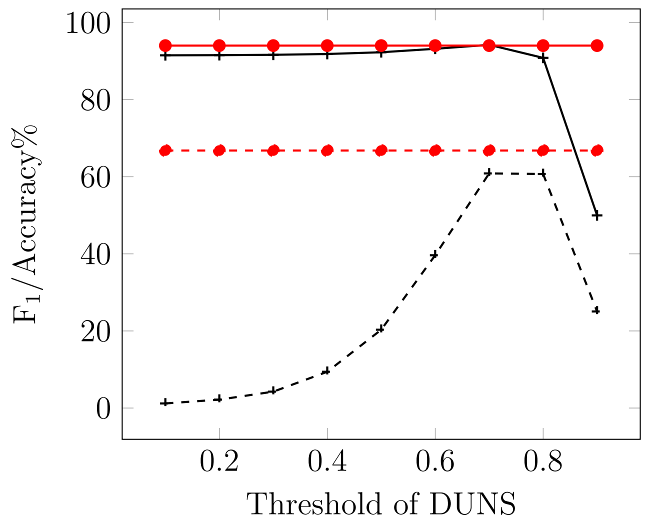
Next we use our approach (DSPL) to get better performance while using the minimum human effort needed during the model update. I.e., we select a subset of the pairs for human-labeling as described in Section 5.1.4. We have tested our approach against other methods, namely, the supervised without model updating (DSNU) and supervised with all labeled data (DSFL). When we select a set of pairs for labeling, we select from among the recent pairs. A recent pair is a pair formed by two reports, both were published in the latest day. Comparing between two **recent** reports makes the human-labeling easier compared to labeling two reports that are several days apart. Table 5.1 shows the results.

Table 5.1: Summary result on Atrocities data set

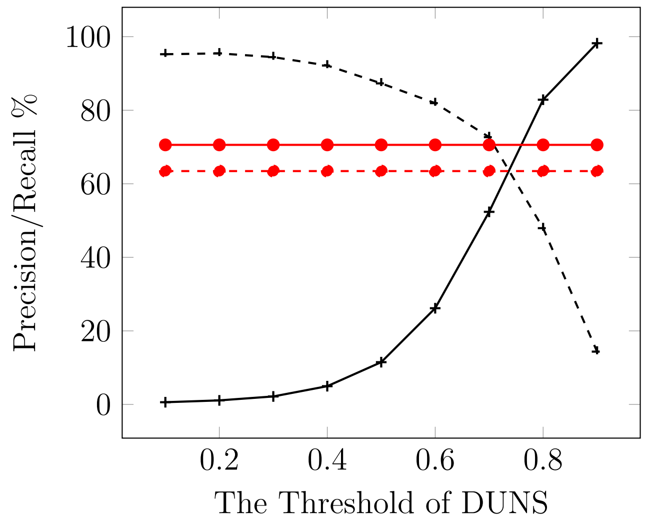
The approach	**# Pairs**	**Precision**	**Recall**	**Accuracy**	**F_1**	**FPR**	**FNR**
DSPL (Our proposed approach)	4,563	63.45	70.58	**94.05**	66.82	**3.77**	29.41
Use random sampling instead of change detection in DSPL	4,854	51.77	72.43	91.92	60.38	6.2	27.56
DSNU (No model updating)	0	26.05	**96.06**	76.51	40.98	25.30	**3.93**
DSFL (Updating using all human-labeled data)	28,609	**67.83**	73.52	**94.79**	**70.56**	**3.23**	26.47
DUNS (Cosine similarity between Unigrams)	0	47.93	82.86	90.87	60.73	8.37	17.13
DUNS (Max value in similarity tuple)	0	38.71	30.39	90.00	34.05	4.46	69.60

The change point detection (CDT) procedure has detected a change 121 times out of the 790 times that it was called (CDT is called once every day). As a result, the procedure selected 4,563 pairs for human-labeling out of 28,609. We did not consider the **not recent** pairs of reports, those that are not published in the same day (including them will result in a total of

155,710 pairs). The results show that human-labeling of a subset of the pairs using DSPL achieves results comparable to selecting all the pairs (DSFL). For example, DSFL requires all updating pairs (28,609 pairs) to be human-labeled to be used in updating the model. It results in 67.83 %, 73.52 %, 94.79 %, and 70.56 % for precision, recall, accuracy, and F_1 respectively. However, the result of DSPL is 63.45 %, 70.58 %, 94.05 %, and 66.82 % respectively. This shows that we can reduce annotation by 85% without substantial change in accuracy, whereas baseline measure do substantially worse. Note that replacing change point detection procedure with randomly-sampled subsets of pairs (4,854 pairs) for human labeling results in only 91.92 % accuracy and 60.38 % F_1.



(a) - -+- - F_1 for DUNS, —+— Accuracy for DUNS , - -●- - F_1 for DSPL, —●— Accuracy for DSPL



(b) - -+- - Precision for DUNS, —+— Recall for DUNS , - -●- - Precision for DSPL, —●— Recall for DSPL

Figure 5.3: F_1, accuracy, precision, and recall for DUNS vs. DSPL. The unsupervised approach (DUNS) uses Cosine similarity between Uni-grams as a feature

CHAPTER 6

UNSUPERVISED DUPLICATE DETECTION

In this chapter we present our approach to detect duplicate news reports in an unsupervised way.

6.1 Background

Word Embeddings are numerical vectors representing words. Figure 6.1 shows examples of such vectors. These vectors encode many linguistic patterns. For example vec[“Woman”] - vec[“Man”] = vec[“Queen”] - vec[“King”].

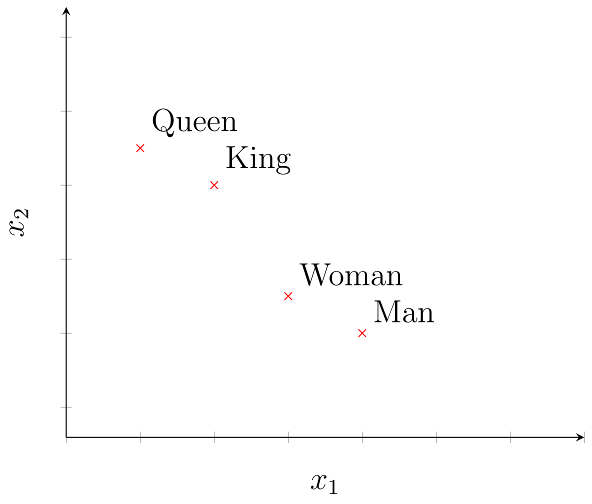


Figure 6.1: Example of Word Embeddings.

Several models have been proposed to learn vector representations of words such as skip-gram which is introduced by word2vec (Mikolov et al., 2013) and used in FastText (Bojanowski et al., 2016). We describe Skip-gram and FastText learning approach. Let W be the number of words in a word vocabulary. The goal is to learn a vector representation for each word w. Given a a large corpus represented as a sequence of words $w_1, \ldots, w_T$, the vector representation should accurately predict the context words of each w. So, skip-gram learns this representation by maximizing the probability of observing a context word (w_c) given a word (w_t) that has appeared in its context in the corpus. Formally,

$$\sum_{t=1}^{T} \sum_{c \in C_t} \log p(w_c | w_t)$$

where the context words C_t are the surrounding of word w_t. The basic skip-gram defines this probability using softmax as:

$$p(w_c | w_t) = \frac{\mathrm{e}^{s(w_t, w_c)}}{\sum_{j=1}^{W} \mathrm{e}^{s(w_t, w_j)}}$$

However, calculating the denominator is impractical because W is very large. So, instead of estimating this probability, skip-gram reduces the problem to binary classification task, where the set of context words of w_t are positive instances, whereas the negative instances ($\mathcal{N}_{t,c}$) are words not mentioned in the context c, and are sampled from vocabulary. The goal then becomes to correctly classify positive and negative instances in the context of w_t.

Let $\ell(x) = \log(1 + \mathrm{e}^{-x})$ be the logistic loss function and let $s(w_\alpha, w_\beta)$ be a scoring function that maps a pair of words (w_α and w_β) to a scalar score. For each word w in the vocabulary, two vectors $\mathbf{u}_w$ and $\mathbf{v}_w$ are defined. $\mathbf{v}_w$ is the vector of word w when it is a context word (w_c) and $\mathbf{u}_w$ is the vector when w is a target word (w_t). The scoring function s is measured by the scalar product $s(w_t, w_c) = \mathbf{u}_{w_t}^{\intercal} \mathbf{v}_{w_c}$. Skip-gram uses binary logistic loss as objective function to be minimized:

$$\sum_{t=1}^{T} \left[\sum_{c \in C_t} \ell(s(w_t, w_c)) + \sum_{n \in \mathcal{N}_{t,c}} \ell(-s(w_t, w_n)) \right]$$

FastText represents each word w as a bag of characters n-gram. For example, the word "going" is represented with 3-grams as {goi, oin, ing}. Let G_w be the set of n-grams for a word w. FastText modifies the scoring function s to represent a word by the sum of its n-grams.

$$s(w, w_c) = \sum_{g \in G_w} \mathbf{z}_g^{\intercal} \mathbf{v}_{w_c}$$

This allows morphological variations of words and rare words to be represented. To speed up the access to words, a hashing function is used to map n-grams to index of word in the word dictionary.

Multilingual word embeddings provide embeddings aligned across multiple languages (Smith et al., 2017; Ammar et al., 2016). which could be used to calculate meaningful distance between words of different languages. For example, the distance between embeddings of the Spanish word "*documento*" and English word "*document*" is less than "*documento*" and "*music*".

We chose a simple and recently proposed method to train multilingual word embeddings (Smith et al., 2017). Using this method, we learn linear mapping between two monolingual embeddings bringing them to the same vector space. First a list of D words is formed. The Words in the list are translated using Google API from their source language to target language. Embeddings for the words and their translations are obtained in matrices X_D and Y_D, respectively. Words vectors in X_D are aligned with their translations in Y_D row by row. As suggested by authors, we use FastText (Bojanowski et al., 2016) word embeddings, which is available for multiple languages. Before learning the mapping, X_D and Y_D should be centered around the origin point (i.e., shift the mean to origin coordinates). This is done by subtracting the mean of X_D (i.e., μ_X) from each column of X_D to obtain $X_D^{'}$ and similarly subtract μ_Y from Y_D to obtain $Y_D^{'}$.

To learn mappings between source and target languages, SVD is applied to $X_D^{'}$ and $Y_D^{'}$ to obtain $Q_D \Sigma_X V_X^\intercal$ and $W_D \Sigma_Y V_Y^\intercal$ respectively. Then perform SVD on matrix $M = Q_D^T W_D = U \Sigma V^\intercal$.

The learned matrices transform source and target words (x and y). First subtract μ_X from x and μ_Y from y to obtain $x^{'}$ and $y^{'}$. Finally, the transformed x and y are obtained by $x^{'} V_X \Sigma_X^{-1} U$ and $y^{'} V_Y^\intercal \Sigma_Y^{-1} V$ respectively. Figure 6.2 sketches the series of SVD operations.

Using the previous transformation, word embeddings that can be used for measuring the cross-lingual similarity between words. In order to estimate the similarity between reports, we

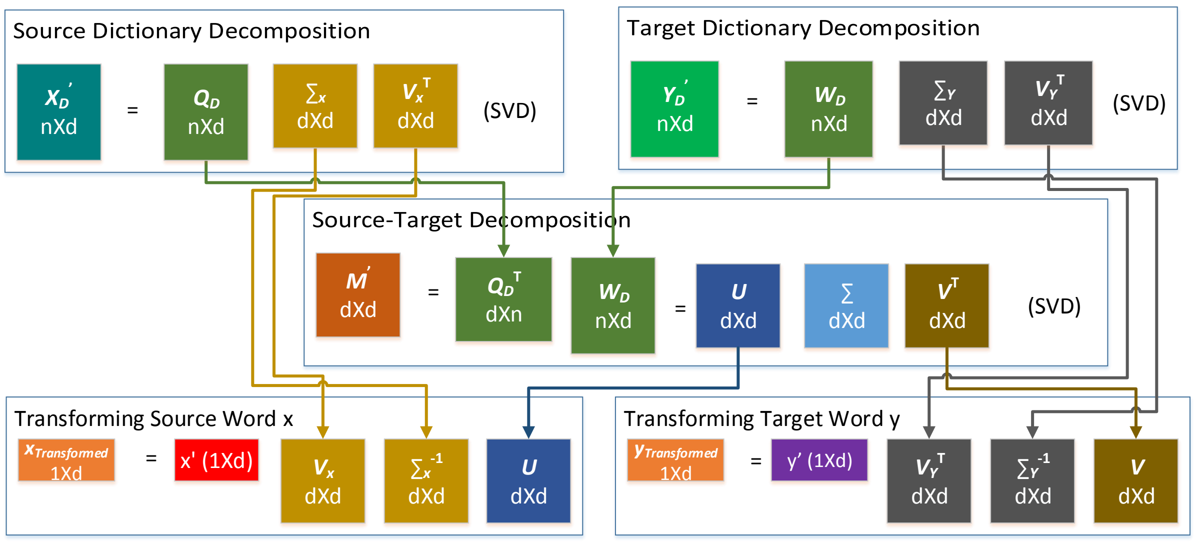


Figure 6.2: Bilingual Orthogonal Transformation

represent each report with word embedding matrix. To form the embedding matrix, we remove stop-words from the report then obtain embeddings for each one of the remaining words. NLTK (Bird et al., 2009) python package provides lists of stop-words for multiple languages. In the following sections, we describe cross-lingual similarity between reports in more details.

6.2 Duplicate Detection

The motivation behind our work is that an intuitive way to measure the similarity/distance between two documents in different languages is to compute the amount of concepts (topics) overlap/disjoint of the two documents. For this, we compute the amount of concepts overlap/disjoint of the two documents. Duplicate reports address common (shared) topics, unlike non-duplicate reports, which tend to discuss disjoint concepts.

Our goal is to calculate similarity/distance score between a pair of reports across languages. Knowing that news reports are text documents written using words of different languages, we break the problem into two parts; *words-level* distance and *documents-level* distance. The former represents measuring the distance between words across and within documents. The latter is to calculate the distance between documents to determine duplicates.

Words-level distance: The distance between words of the same language could be measured by computing cosine distance between word embeddings. Word Embeddings are numerical vectors representing words. Examples of Word Embeddings include Word2vec (Mikolov et al., 2013) and FastText (Bojanowski et al., 2016). Calculating the distance between words of different languages is a challenging task. Multilingual word embeddings provide embeddings aligned across multiple languages (Smith et al., 2017; Ammar et al., 2016). Such alignment (or mapping) enables us to calculate meaningful distance between words of different languages. For example, the distance between embeddings of the Spanish word "*documento*" and English word "*document*" is less than "*documento*" and "*music*". We adopt multilingual word embeddings (Smith et al., 2017) to obtain word embeddings for words of different languages and use cosine distance between embeddings to measure the distance between words.

Documents-level distance: The distribution of word embeddings for a given document represents the mentioned concepts. To capture this distribution we use density-based clustering (DBSCAN). DBSCAN (Ester et al., 1996) is a density-based clustering method that clusters data points into highly-dense clusters and discovers noise. We use DBSCAN rather than other clustering methods because it does not require specifying the number of clusters and its ability to cluster data that contain noise. It can detect dense areas (i.e., clusters) in the distribution of data points, hence define boundaries of concepts.

Given two documents, we analyze clusters produced by applying DBSCAN on the embeddings of words of both documents. Overlapping clusters indicate the existence of shared concepts. We define a concept as a set of similar words. A concept is *shared* when it appears in multiple documents. Disjoint clusters represent concepts mentioned by only one of the documents. Duplicate reports tend to have more overlapping clusters than disjoint.

To summarize, we, first, remove stop-words and obtain the multilingual word embedding of each document. The result is two matrices of size ($n_i \times d$), where n_i is the number of words in document i and d is the number of embeddings dimensions. Each vector (row) is embedding of a word mentioned in the document. Then we concatenate both matrices in one (n $\times$ d) matrix (n is the total number of words in both documents). Next we cluster the words using DBSCAN. As a result, we get clusters of vectors (words). There are three types of clusters, i.e., *Pure*, *Impure*, and *Noise* clusters.

- A Pure cluster is a cluster in which all its members belong to only one document. We denote pure clusters of a document (A) as $pure_A$ and a document (B) as $pure_B$.
- An Impure cluster contains words mentioned in both documents ($impure$).
- Noise clusters are sparse clusters. Each of these clusters may contain one or few words. This happens when the number of words within ϵ proximity is less than $mPts$. We set $\epsilon = 0.4$ and $mPts = 2$. So, Noise clusters are actually singleton clusters. We exclude these clusters from our further analysis.

A $pure_A$ cluster represents a concept addressed by only report A. Similarly, A $pure_B$ cluster is a report B concept. However, $impure$ clusters are shared concepts.

Figure 6.4 shows the clustering of a pair of documents (A and B). The Axis (x_1 and x_2) are word embeddings (showing only 2 dimensions for visualization). The data points are the words of the two documents represented by their embeddings.

Purity-based Distance: We count the number of pure and impure clusters ($|pure_A|$, $|pure_B|$, $|impure|$). Then we calculate distance using our purity-based distance measure which is inspired by Szymkiewicz-Simpson coefficient, where $|impure| = |A \cap B|$, $|pure_A| = |A - B|$, and $|pure_B| = |B - A|$ as in Figure 6.5.

$$\text{Purity-based Distance} = \frac{\min(|pure_A|, |pure_B|)}{\min(|pure_A|, |pure_B|) + |impure|} \tag{6.1}$$

A small distance value means that we have more impure clusters versus pure, which suggests that the reports are duplicates. On the other hand, more pure clusters indicates that the reports are addressing different concepts, hence not duplicates. We classify a pair of reports as duplicate if purity-based distance between them is less than a threshold (h), otherwise they are non-duplicates. We suggest using a value of $h = 0.5$.

In Figure 6.4, $|pure_A| = 2$, $|pure_B| = 1$, and $|impure| = 1$. The distance is 0.5, hence non-duplicate.

Algorithm 6 sketches the main steps of duplicate detection. First, we remove stop-words from the documents in Lines 2 and 3. Then we get the embeddings for each of the remaining words in Lines 4 and 5 which results in two embeddings matrices. We concatenate them in one matrix X (Line 6). We cluster vectors in X using DBSCAN (in Line 7). We suggest setting $mPts = 2$ and $\epsilon = 0.4$ and use Cosine distance as measuring metric. $Pure_A$, $pure_B$, and $impure$ clusters are collected in Lines 8, 9, and 10. Then we calculate distance between the documents (PD) using Equation 6.1 in Line 11. Finally, in Line 13 we classify the documents as duplicates if $PD < h$, otherwise they are non-duplicates. We use $h = 0.5$.

We implemented our approach and all baseline approaches with Python using scikit-learn (Pedregosa et al., 2011) package.

6.2.1 Analytical Derivation of Purity-based Distance

We prove the validity of our purity-based distance by comparing it to Szymkiewicz-Simpson coefficient (or overlap coefficient). Here, we refer to Szymkiewicz-Simpson coefficient as SS and purity-based distance as PD.

$$SS = \frac{|A \cap B|}{\min(|A|, |B|)}$$

We prove that PD is the complement of SS, i.e.,

$$SS = 1 - PD$$

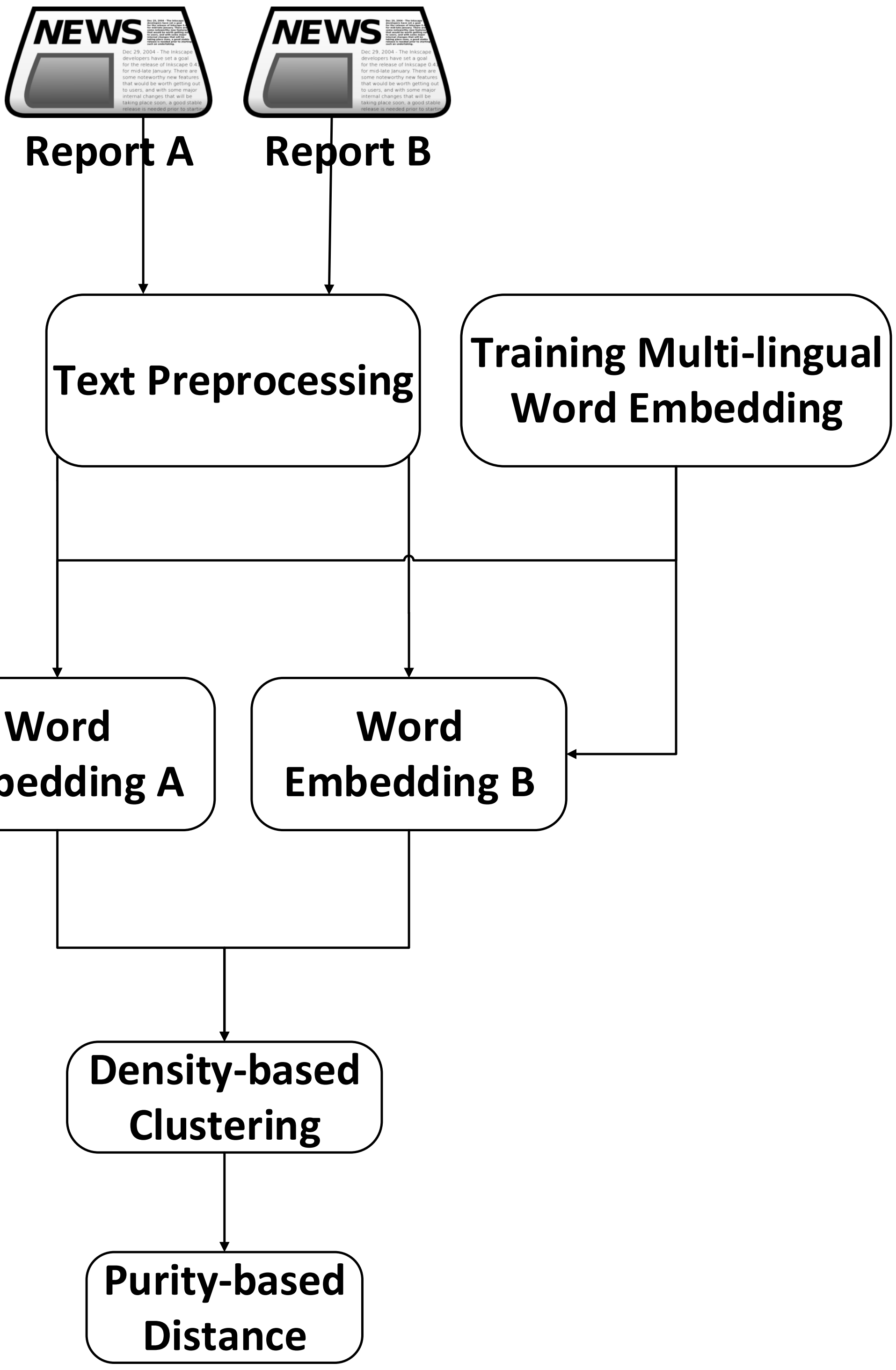


Figure 6.3: Purity-based distance between two documents

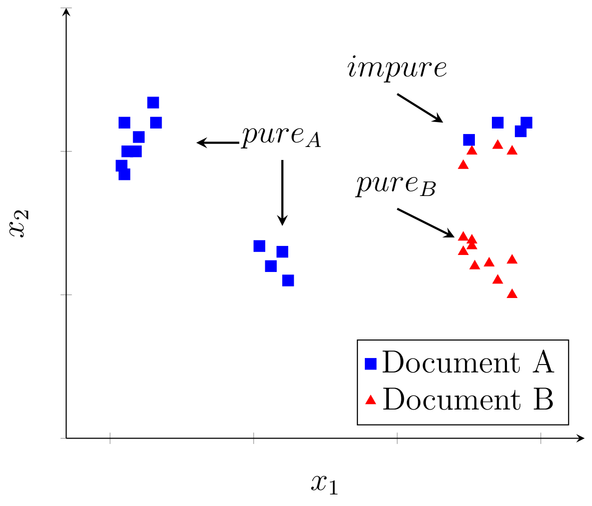


Figure 6.4: Example of Clustering words of two documents (A and B). The language of A is different than B. x_1 and x_2 are word embeddings dimensions.

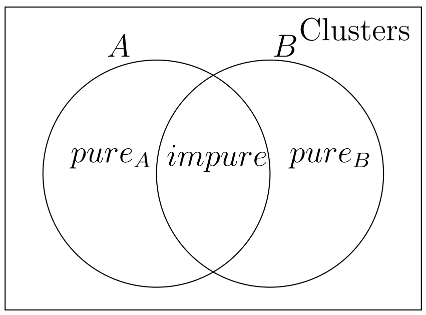


Figure 6.5: Clusters types based on their members. A and B are sets of clusters containing words from document A and document B respectively. $Impure$ clusters contain words from both documents

Keep in mind that:

$$|pure_A| = |A - B|$$

$$|pure_B| = |B - A|$$

$$|impure| = |A \cap B|$$

Case 1:

if $\min(|A|, |B|) = |A| \implies \min(|pure_A|, |pure_B|) = |pure_A|$

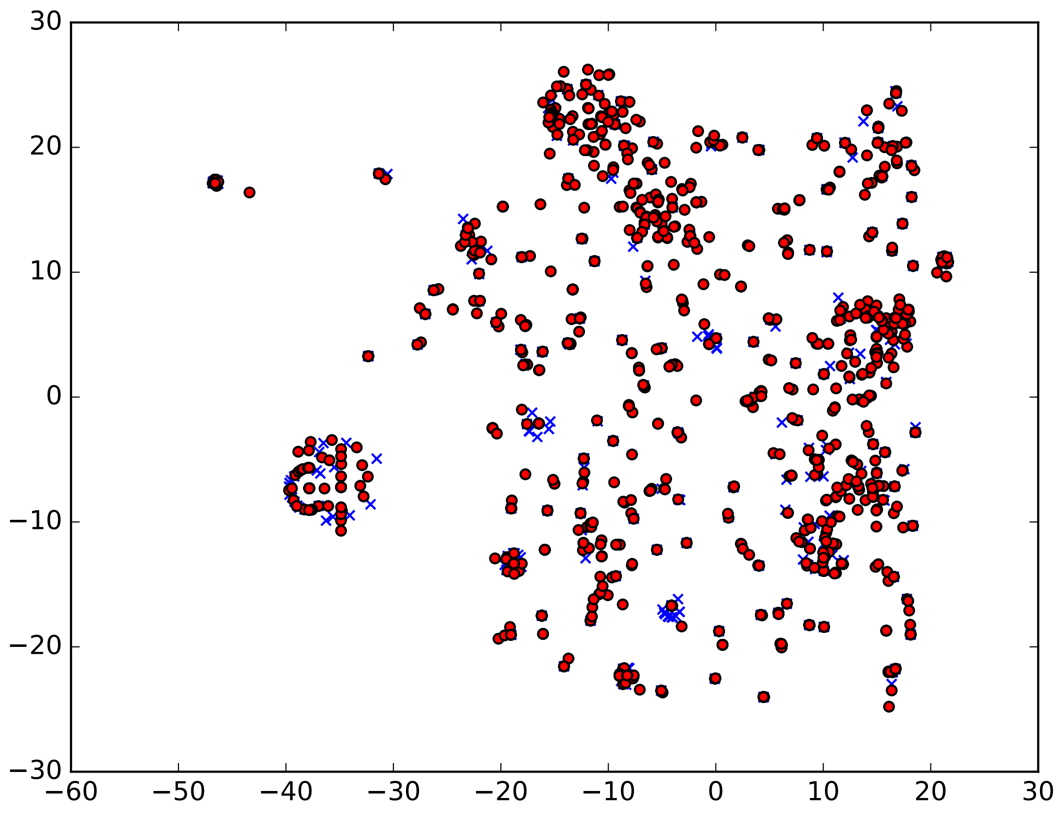


Figure 6.6: True example of a duplicate pair of documents. Embedding dimensions are reduced using t-sne. x and o represent words of two documents

$$\begin{aligned}
&1 - \frac{\min(|pure_A|, |pure_B|)}{\min(|pure_A|, |pure_B|) + |impure|} \\
&= 1 - \frac{|pure_A|}{|pure_A| + |impure|} \\
&= 1 - \frac{|pure_A|}{|A|} \\
&= \frac{|A| - |pure_A|}{|A|} \\
&= \frac{|A \cap B|}{|A|}
\end{aligned} \tag{6.2}$$

Case 2:

$$\text{if } \min(|A|, |B|) = |B| \implies \min(|pure_A|, |pure_B|) = |pure_B|$$

Algorithm 6: Detect-Duplicates (Doc$_A$, Doc$_B$, ϵ, $mPts$, h)

```
input  : Doc$_A$: Textual report,
         Doc$_B$: Textual report,
         $\epsilon$: DBSCAN proximity parameter,
         $mPts$: DBSCAN minimum number of points,
         $h$: Purity-based distance threshold.
output : 1 if Duplicates; 0 otherwise
1  begin
2      $W_A \leftarrow \{Word \in Doc_A \mid Word \notin \text{STOPWORDS}\}$
3      $W_B \leftarrow \{Word \in Doc_B \mid Word \notin \text{STOPWORDS}\}$
4      $X_A \leftarrow \{Embeddings_{Word} \mid Word \in W_A\}$
5      $X_B \leftarrow \{Embeddings_{Word} \mid Word \in W_B\}$
6      $X \leftarrow X_A \cup X_B$
7      $C \leftarrow \text{DBSCAN}(X, \epsilon, mPts)$ // returns clusters
8      $pure_A \leftarrow \{clst \in C \mid (\forall wrd \in clst)(wrd \in W_A \wedge wrd \notin W_B)\}$
9      $pure_B \leftarrow \{clst \in C \mid (\forall wrd \in clst)(wrd \notin W_A \wedge wrd \in W_B)\}$
10     $impure \leftarrow \{clst \in C \mid (\exists wrd \in clst)(wrd \in W_A) \wedge (\exists wrd \in clst)(wrd \in W_B)\}$
11     $P - D \leftarrow$ Purity-based Distance is calculated using Equation 6.1
12     if $PD < h$ then
13         Return 1
14     else
15         Return 0
```

$$\begin{aligned} &1 - \frac{\min(|pure_A|, |pure_B|)}{\min(|pure_A|, |pure_B|) + |impure|} \\ &= 1 - \frac{|pure_B|}{|pure_B| + |impure|} \\ &= \frac{|A \cap B|}{|B|} \end{aligned} \tag{6.3}$$

Equations 6.2 and 6.3 show that our purity-based distance is the complement of Szymkiewicz-Simpson coefficient.

6.3 Datasets

We tested our approach using two datasets: (i) Event registry (ii) EuroNews. Event registry (Leban et al., 2014) is a system that analyzes news articles across languages. It is able to match groups of articles that mention the same event and represent them as a single event. Event registry provides a dataset of duplicate and non-duplicate reports. These reports are collected from a set of different agencies and were manually labeled by two human annotators (Leban et al., 2014). The dataset is publicly available at "`https://github.com/rupnikj/jair_paper.git`" in the "dataset" sub-folder.

We have selected English, Spanish, and Deutsch pairs of reports. We have discarded pairs of same language as they are out of the scope of this study. Unlike our approach, Event registry system relies on a-priori grouping. Each one of these groups consists of duplicate reports all in the same language. Event registry system compares between two *groups* of reports and determines whether the two groups are duplicates or not. In other words, it calculates the similarity between two groups of reports. However, our approach compares between two individual reports not groups. We do not rely on a-priori reports clustering. Our clustering method is applied on word embeddings to measure the similarity between two reports.

In order to build our testing dataset from Event Registry data, we have extracted duplicate and non-duplicate reports by linking reports across groups. For example, assume that a group (U) is *duplicate* with another group (V). Each group consists of duplicate reports. We link each report in U with every report in V and label each pair as *duplicate* (Figure 6.7). Similarly, if two groups are not duplicate, we pair reports and label them non-duplicate. The number of resulting pairs is $|U| \times |V|$.

Table 6.2 summarizes the number of duplicate and non-duplicate pairs in Event registry dataset. The data is skewed especially in the cases of English-Spanish and English-Deutsch pairs.

EuroNews (`https://euronews.com`) is a multilingual news media service that covers Worldwide news. It was created in 1993 in Lyon, France. In this dataset, although the reports are publish by the same agency, we observed that duplicate reports may significantly vary in the size and the amount of details. Duplicate reports are aligned. We downloaded duplicate reports written in Arabic, Deutsch, English, Spanish, Farsi, and French. Our collection covers more than three months between June 1st and October 18th, 2017. In order to extract non-duplicate reports, we randomly paired reports from our collection as long as they did not appear as duplicate in the collection.

Table 6.3 shows the total number of pairs in every language. EuroNews data is balanced. I.e., the number of duplicates = non-duplicates. For instance, English-Arabic pairs are 1,662, among them 831 are duplicates and the same number are non-duplicates.

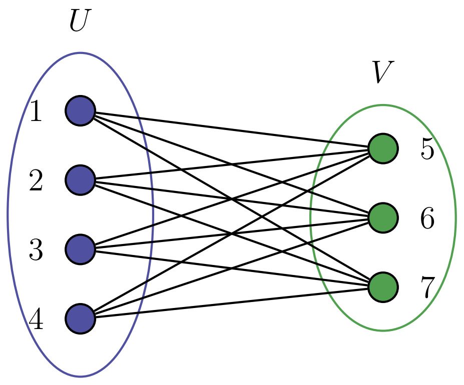


Figure 6.7: Extracting pairs of reports from Event registry groups

Table 6.1: Languages in our experiments

ar	Arabic
de	Deutsch (German)
en	English
es	Spanish
fa	Farsi (Persian)
fr	French

Table 6.2: The number of Positive and Negative pairs in *Event Registry* dataset (#Duplicate - #Not-Duplicate)

es	41,566 - 13,684	
de	50,004 - 8,633	4,071 - 5,910
	en	es

Table 6.3: The number of pairs in *EuroNews* dataset (#Duplicate = #Not-Duplicate)

ar	1,662				
fa	1,366	912			
fr	3,580	1,382	1,166		
es	3,558	1,348	1,148	3,024	
de	3,930	1,490	1,218	3,180	3,194
	en	ar	fa	fr	es

6.4 Baseline Approaches

To evaluate the performance of our approach, we compare it to other approaches. In this section we present these baseline approaches.

6.4.1 Distance between Means of Word Embeddings

Give two documents A and B, we obtain embeddings of the words. This will create two matrices: X_A for document A and X_B for document B, where the number of rows equals the number of words in the document (after eliminating stop-words) and the number of columns is the number of embedding's dimensions r.

Each matrix is then averaged by taking the mean of each dimension resulting in two vectors μ_A and μ_B, both of the same size (r).

Then we calculate euclidean distance (d) between the two vectors. If d is less than a threshold, we classify the documents as duplicates, otherwise they are non-duplicates.

In our experiments, we chose the threshold for all baseline approaches by fixing recall value. In other words, we chose the threshold that gives recall value as close as possible to the recall value obtained by our approach. The threshold of our approach is 0.5.

Although this approach is simple, it is widely used in practice. Some implementations use summation instead of mean. In the experiments section we name this approach "Word Embeddings Means".

6.4.2 Similarity between Tf-idf vectors matched using CCA

Tf-idf is a well-known method can be used to convert documents to vectors. Unfortunately, it can not be directly used for calculating the similarity between documents of different languages because the vocabulary sets do not match across languages. To solve this problem, several approaches have applied Canonical Correlation Analysis (CCA) (Leban et al., 2014). CCA finds latent dimensions that maximally correlate. It connects two sets of data points by transforming them to a new dimensional space.

We take a subsample from our duplicate Tf-idf document-pairs. Then we use them to train CCA model, and transform the features of the remaining pairs according to the trained CCA model. Finally, we calculate cosine similarity between transformed vectors. If it is less than a threshold, the pair are classified as duplicate.

In the experiments section we name this approach "CCA-tfidf".

6.4.3 Similarity between Named Entities

In this approach Named Entities (NE) like locations, organizations, and person names are extracted from reports (using CoreNLP NER (Manning et al., 2014)) and used for calculating similarity between reports. For that we need to measure the amount of overlapping between

NE's in a document (A) (i.e., NE$_A$) and NE$_B$ for a document B. The assumption is that NE's are similar across languages (which may happen in some European languages, e.g., London).

In our experiments, we apply this approach on only English-Spanish-Deutsch pairs. That is because obtaining NE for all languages is not possible and matching NE's which have different character sets is difficult. We assume that two NE's (NE_1 and NE_2) match if:

$$\frac{\text{Levenshtein distance}(NE_1, NE_2)}{|NE_1| + |NE_2|} < 0.25 \tag{6.4}$$

So, NE Similarity=$\frac{|NE_A \cap NE_B|}{|NE_A \cup NE_B|}$, where $|NE_A \cap NE_B|$ is the number named entities that are matching according to Equation 6.4. If the similarity is less than a threshold, we consider documents A and B as being duplicates. We call this approach "Named Entities".

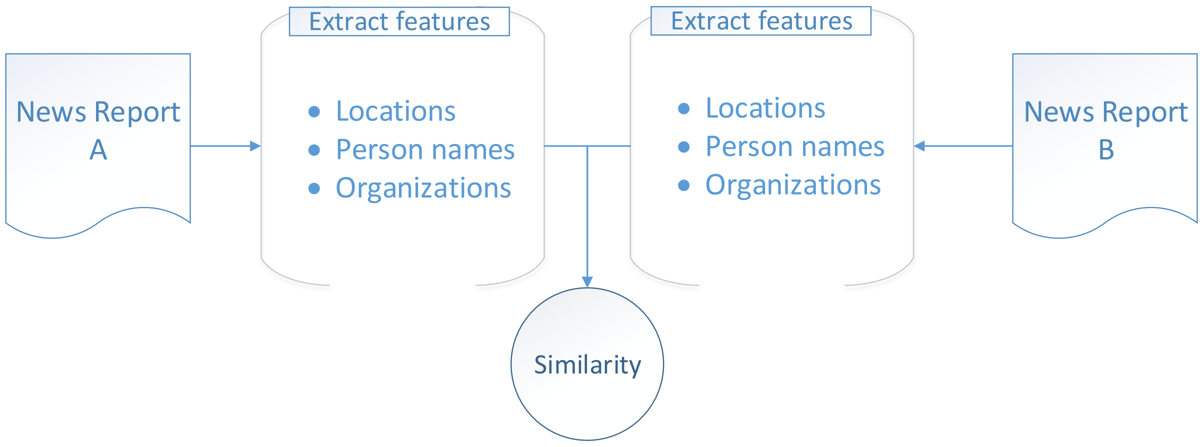


Figure 6.8: Similarity between Named Entities

6.4.4 Similarity between Tf-idf vectors matched using machine translation

Using machine translation for measuring cross-lingual text similarity has been used by multiple approaches (Potthast et al., 2011; Alzahrani et al., 2010; Gottschalk and Demidova, 2017). Inspired by these approaches, we developed the following method to detect duplicate reports. First, we translate the documents into English using Google translate API. Then we use Tf-idf to represent the translated documents. Finally, we calculate the similarity between Tf-idf vectors using cosine similarity. We refer to this approach as “Translation-Tfidf”.

6.5 Experimental Results

We apply our approach on EuroNews and Event registry datasets (described in Section 6.3). To evaluate our duplicate detection approach, we have calculated the number of false positive/negative (Table 6.5), total accuracy, F_1, F_2, precision, and recall. Note that the duplicate class is the positive class and non-duplicate is the negative class.

$$F_\beta = \frac{(1+\beta^2) * TP}{(1+\beta^2) * TP + \beta^2 * FN + FP},$$

where $\beta = 1$ for F_1 and $\beta = 2$ for F_2.

We compare our approach to several baseline approaches (Section 6.4). Results for EuroNews dataset are shown in Table 6.6. Table 6.7 shows Event registry results.

We set the threshold used in our approach (Purity-based distance between words embeddings) to $h = 0.5$. However, we determine the thresholds of other approaches by fixing recall. In other words, we vary the threshold value until the recall value is as close as possible to our approach's recall. This way we make the comparison easier especially when data is not balanced.

We did not apply similarity between NE's approach for pairs other than English-Spanish-Deutsch. That is because the spelling is significantly different in other languages.

The number of duplicate pairs tested using CCA-tfidf is less than other approaches. That is because, unlike other approaches, we train CCA to learn feature transformation. For that we use 1/3 of the duplicate pairs. We exclude these pairs during testing. For example, we have 831 duplicate pairs in English-Arabic. We use 276 for CCA training and the remaining (in addition to non-duplicates) for testing. Since the accuracy of CCA-tfidf approach is low, we show it only for languages for which NE similarity is not applicable.

Our approach outperforms baseline approaches in terms of accuracy, F_1, and F_2. For example, in EuroNews dataset, our accuracy for English-Arabic pairs is 81.77 %, while it is 75.51 % for embeddings means and 64.19 % for CCA-Tfidf. With Event registry data on English-Spanish, our approach achieved 71.93 % for accuracy, 79.38 % F_1, and 74.64 % F_2; whereas, embedding means achieved 69.05 %, 77.73 %, and 74.06 % respectively. NE's approach result is 21.70 %, 33.39 %, and 28.26 % respectively. Machine translation approach result is 63.51 %, 16.74 %, and 29.97 % respectively.

We measure the time consumed to predict whether two documents are duplicates. Our approach is less time consuming as compared to Named Entity approach. That is because the complexity of extracting entities is high as it requires parsing sentences with Named Entity Recognizer. CCA requires more time than our approach. CCA has to perform several matrix multiplication operations in order to transform the input vectors then compute the similarity between new vectors. Our approach consumes more time than mean embeddings approach. The main bottleneck in our approach is performing DBscan clustering. Table 6.4 shows the average time required to examine whether two reports are duplicate or not.

Table 6.4: Average time taken to examine duplication between two documents

Method	**Time (sec)**
Purity-based distance (Word Embeddings)	8×10^{-3}
Word Embeddings Mean	5×10^{-3}
CCA-Tfidf	1.2×10^{-2}
Named Entities	4.087

Table 6.5: Measurements for duplicate detection

TP	The number of pairs that are duplicate and classified as duplicate.
FP	The number of pairs that are not duplicate but classified as duplicate.
TN	The number of pairs that are not duplicate and classified as not duplicate.
FN	The number of pairs that are duplicate but classified as not duplicate.

Table 6.6: Summary result on EuroNews dataset

English-Arabic			
Approach:	**Our Approach**	**Word Embeddings Mean**	**CCA-TfIdf**
Accuracy%	**81.77**	75.51	64.19
F_1%	**79.35**	74.09	32.43
F_2%	**73.48**	71.60	24.83
Precision%	91.51	78.65	66.11
Recall%	70.04	70.04	21.48
FPR%	6.50	19.01	7.34
FNR%	29.96	29.96	78.52

Table 6.6 continued

English-Farsi			
Approach:	**Our Approach**	**Word Embeddings Mean**	**CCA-TfIdf**
Accuracy%	**80.67**	70.72	64.44
F_1%	**82.14**	75.22	33.93
F_2%	**86.05**	82.86	26.25
Precision%	76.35	65.20	66.24
Recall%	88.87	88.87	22.81
FPR%	27.53	47.44	7.76
FNR%	11.13	11.13	77.19

Table 6.6 continued

English-French			
Approach:	**Our Approach**	**Word Embeddings Mean**	**CCA-TfIdf**
Accuracy%	**88.21**	77.32	63.61
F_1%	**88.22**	79.56	30.03
F_2%	**88.25**	84.56	22.69
Precision%	88.17	72.41	65.08
Recall%	88.27	88.27	19.51
FPR%	11.84	33.63	6.98
FNR%	11.73	11.73	80.49

Table 6.6 continued

English-Spanish				
Approach:	**Our Approach**	**Word Embeddings Mean**	**Named Entities**	**CCA-TfIdf**
Accuracy%	**86.85**	77.54	74.04	64.55
F_1%	**86.47**	78.91	64.88	30.81
F_2%	**84.99**	81.91	53.99	23.05
Precision%	89.04	74.38	97.74	70.27
Recall%	84.04	84.04	48.55	19.73
FPR%	10.34	28.95	1.09	5.56
FNR%	15.96	15.96	51.45	80.27

Table 6.6 continued

English-Deutsch				
Approach:	**Our Approach**	**Word Embeddings Mean**	**Named Entities**	**CCA-TfIdf**
Accuracy%	**83.64**	72.96	71.82	62.96
F_1%	**83.05**	74.77	60.12	29.11
F_2%	**81.29**	77.91	48.77	22.07
Precision%	86.16	70.08	98.25	62.09
Recall%	80.15	80.14	43.31	19.01
FPR%	12.88	34.22	0.74	7.74
FNR%	19.85	19.86	56.69	80.99

Table 6.6 continued

Arabic-Farsi			
Approach:	**Our Approach**	**Word Embeddings Mean**	**CCA-TfIdf**
Accuracy%	67.87	**73.79**	64.21
F_1%	70.01	**74.11**	32.67
F_2%	72.92	**74.64**	25.08
Precision%	65.64	73.23	66.00
Recall%	75.00	75.00	21.71
FPR%	39.25	27.41	7.46
FNR%	25.00	25.00	78.29

Table 6.6 continued

Arabic-French			
Approach:	**Our Approach**	**Word Embeddings Mean**	**CCA-TfIdf**
Accuracy%	**79.53**	73.88	64.32
F_1%	**78.40**	74.01	33.39
F_2%	**75.91**	74.23	25.75
Precision%	82.95	73.64	66.03
Recall%	74.33	74.38	22.34
FPR%	15.28	26.63	7.67
FNR%	25.67	25.62	77.66

Table 6.6 continued

Arabic-Spanish			
Approach:	**Our Approach**	**Word Embeddings Mean**	**CCA-TfIdf**
Accuracy%	**77.42**	72.03	64.06
F_1%	**76.72**	72.66	31.76
F_2%	**75.30**	73.65	24.20
Precision%	79.20	71.06	66.20
Recall%	74.38	74.33	20.89
FPR%	19.54	30.27	7.12
FNR%	25.62	25.67	79.11

Table 6.6 continued

Arabic-Deutsch			
Approach:	**Our Approach**	**Word Embeddings Mean**	**CCA-TfIdf**
Accuracy%	**75.23**	68.59	63.93
F_1%	**75.05**	70.34	29.78
F_2%	**74.72**	72.78	22.31
Precision%	75.61	66.63	67.38
Recall%	74.50	74.50	19.11
FPR%	24.03	37.32	6.17
FNR%	25.50	25.50	80.89

Table 6.6 continued

Farsi-French			
Approach:	**Our Approach**	**Word Embeddings Mean**	**CCA-TfIdf**
Accuracy%	**78.56**	72.98	64.30
F_1%	**77.88**	73.64	31.29
F_2%	**76.42**	74.73	23.62
Precision%	80.44	71.90	68.10
Recall%	75.47	75.47	20.31
FPR%	18.35	29.50	6.35
FNR%	24.53	24.53	79.69

Table 6.6 continued

Farsi-Spanish			
Approach:	**Our Approach**	**Word Embeddings Mean**	**CCA-TfIdf**
Accuracy%	**78.05**	69.69	65.94
F_1%	**78.24**	72.25	35.06
F_2%	**78.65**	76.11	26.65
Precision%	77.57	66.62	73.95
Recall%	78.92	78.92	22.98
FPR%	22.82	39.55	5.40
FNR%	21.08	21.08	77.02

Table 6.6 continued

Farsi-Deutsch			
Approach:	**Our Approach**	**Word Embeddings Mean**	**CCA-TfIdf**
Accuracy%	**74.71**	67.57	64.24
F_1%	**73.90**	68.82	29.79
F_2%	**72.50**	70.46	22.19
Precision%	76.36	66.26	69.37
Recall%	71.59	71.59	18.97
FPR%	22.17	36.45	5.58
FNR%	28.41	28.41	81.03

Table 6.6 continued

French-Spanish

Approach:	**Our Approach**	**Word Embeddings Mean**	**CCA-TfIdf**
Accuracy%	**84.56**	74.64	63.81
F_1%	**85.06**	77.61	30.17
F_2%	**86.74**	83.47	22.75
Precision%	82.39	69.47	66.11
Recall%	87.90	87.90	19.54
FPR%	18.78	38.62	6.68
FNR%	12.10		
	12.10	80.46	

Table 6.6 continued

French-Deutsch

Approach:	**Our Approach**	**Word Embeddings Mean**	**CCA-TfIdf**
Accuracy%	**82.23**	74.25	64.83
F_1%	**81.06**	74.70	34.18
F_2%	**77.97**	75.50	26.33
Precision%	86.79	73.41	67.98
Recall%	76.04	76.04	22.83
FPR%	11.57	27.55	7.17
FNR%	23.96	23.96	77.17

Table 6.6 continued

Spanish-Deutsch

Approach:	**Our Approach**	**Word Embeddings Mean**	**Named Entities**	**CCA-TfIdf**
Accuracy%	**80.71**	72.20	68.25	63.79
F_1%	**79.40**	72.78	52.79	29.01
F_2%	**76.28**	73.70	41.20	21.63
Precision%	85.21	71.29	99.46	67.24
Recall%	74.33	74.33	35.93	18.50
FPR%	12.90	29.93	0.19	6.01
FNR%	25.67	25.67	64.07	81.50

Table 6.7: Summary result on Event Registry dataset

English-Spanish					
Approach:	**Our Approach**	**Word Embeddings Mean**	**Named Entities**	**CCA-TfIdf**	**Translation-TfIdf**
Accuracy%	**71.93**	69.05	21.70	31.44	63.51
F$_1$%	**79.38**	77.73	33.39	22.43	16.74
F$_2$%	**74.64**	74.06	28.26	15.85	29.97
Precision%	88.76	84.73	47.88	72.82	9.64
Recall%	71.79	71.79	25.63	13.26	63.33
FPR%	27.62	39.29	91.14	14.67	36.48
FNR%	28.21	28.21	74.37	86.74	36.67

Table 6.7 continued

English-Deustch					
Approach:	**Our Approach**	**Word Embeddings Mean**	**Named Entities**	**CCA-TfIdf**	**Translation-TfIdf**
Accuracy%	**71.35**	70.98	64.10	24.66	64.85
F$_1$%	**80.86**	80.68	77.21	20.46	78.11
F$_2$%	**74.63**	74.57	73.34	13.85	69.59
Precision%	93.94	93.44	84.65	99.91	98.13
Recall%	70.98	70.98	70.97	11.40	64.87
FPR%	26.53	29.03	76.98	0.06	35.87
FNR%	29.02	29.02	29.03	88.60	35.13

Table 6.7 continued

Spanish-Deutsch					
Approach:	**Our Approach**	**Word Embeddings Mean**	**Named Entities**	**CCA-TfIdf**	**Translation-TfIdf**
Accuracy%	**61.59**	54.29	25.37	59.06	55.00
F$_1$%	56.17	51.83	31.73	13.26	**58.46**
F$_2$%	**58.51**	56.52	40.63	10.44	55.88
Precision%	52.65	45.53	23.25	24.06	63.33
Recall%	60.19	60.14	49.96	9.15	54.29
FPR%	37.45	49.76	87.71	15.01	44.00
FNR%	39.81	39.86	50.04	90.85	45.71

CHAPTER 7
CONCLUSION AND FUTURE WORK

In this chapter we summarize our proposed approaches and give pointers to future work.

7.1 Unsupervised deep embedding for novel class detection

We presented a novel class detection approach that combines deep learning, outlier detection, and ensemble-based classification techniques. We also presented a multidimensional nonparametric change point detection. Our approach outperforms current state-of-the-art methods.

In the future we would like to investigate efficient ways to update DAE weights incrementally. Also we want to extend our work to detect zero-day attacks. Zero-day attacks are novel attacks that are not previously modeled. Such attacks can be very harmful to systems.

7.2 Binary Class Stream Data Classification for Multidimensional Data using Change point detection

We have presented an approach to classify data points in data streams having two classes. To do this, we have proposed an efficient method to detect change point in a multi-dimensional data set without using any pre defined threshold value. Moreover, our approach does not have the criteria of dividing the infinite length data stream into fixed size chunks. It uses the idea of change point detection in the data stream by sequential partitioning that determines an appropriate length for a chunk on the fly. Experimental result shows that the proposed approach is an efficient framework to classify data points of data stream containing two classes, e.g., data stream containing data points having classes benign/malicious.

In the future, we intend to continue our work to classify data points and separate novel classes from concept drift in data stream having multi classes by exploiting ensemble based techniques.

7.3 Supervised Duplicate Detection

Our supervised duplicate detection approach is a significantly improved technique for detecting duplicate reports within data streams, all while drastically reducing the amount of human effort required. By using a sliding window, incremental classifier to label report pairs,and change point detection technique, this method lowers labor costs and responds effectively to the concept drift problem that has plagued this field.

In the future we want to investigate more unsupervised approaches and use clustering to create boundaries between concepts. These boundaries can be used to detect duplicate reports and concept drift.

7.4 Unsupervised Duplicate Detection

We propose an unsupervised approach for detecting duplicate news reports across languages. Our approach combines multilingual word embeddings, density-based clustering, and our purity-based distance to determine whether two documents are duplicate without requiring labeled data. We compared our approach to several baseline approaches using publicly available datasets. The results show that our approach outperforms other approaches.

In the future we want to extend this work to measure similarity between groups of multi-lingual documents. We would like to study the effect of using deep clustering (or clustering using features learned using deep neural networks). We also want to detect and track events. We intend to investigate fake news detection using our technique of calculating cross-lingual similarity.

APPENDIX
DATA STREAM ANOMALY DETECTION USING CLUSTERED MARKOV NETWORKS

Anomaly detection is a common component in computer security. In this appendix we discuss our host-based anomaly detection method (Mustafa et al., 2013). First, the training data are clustered. Then, in each cluster, a separate Markov network is built to model the underlying benign behavior. During testing, each Markov network predicts the probability for each instance. If the probability from multiple Markov networks is low, the point is classified as malicious. We experimentally show that our proposed approach outperforms several other approaches, while being less sensitive to mislabeled points in the training data.

A.1 Introduction

Anomaly detection (Chandola et al., 2009) refers to the problem of detecting patterns in data that do not conform to a normal or expected behavior. These patterns are called anomalies or outliers. In network security, anomaly detection can be the basis for an intrusion detection system, which aims to identify illegitimate/malicious activities that deviate from normal "benign" activities.

The intrusion detection process can be divided into two categories: 1) host-based intrusion detection (Freeman et al., 2002), in which the system monitors operating system events like system calls on each computer in the network; and 2) network-based intrusion detection (Khan et al., 2007), in which the system monitors the network traffic data. Traditionally, network-based intrusion detection focuses on data received from an outsider (Hu, 2010). Therefore, it cannot detect malicious data generated by an insider or malicious data that is being sent without generating abnormal-looking network traffic.

Our system is a host-based intrusion detection system which collects periodic snapshots of system calls from machines on the network. Then, it examines them by comparing each snapshot to a prior model which is trained using historical data. Typical approaches to training such a model assume that there is enough training data to represent both the benign and malicious classes, so that the model can learn accurate representations of both classes. However, there is an inherent class imbalance problem in intrusion detection: malicious data instances are exceptionally rare compared to benign data instances. We examine two approaches to overcoming this imbalance. The first uses unsupervised learning and is trained with only benign data, and the second uses supervised learning and is trained with mostly benign data with a small proportion of malicious data.

We present a host-based intrusion detection method, *clustered Markov networks* (CMN). CMN first applies k-means clustering to the benign training data, and then uses Markov networks to probabilistically model each cluster. During testing, each Markov network predicts the probability of test instances. If the average predicted probability is below a threshold, the system declares the instance malicious.

We experimentally compare our methods with existing methods, including *label propagation* (LP) (Bengio et al., 2006), *Markov networks* (MN)(Koller et al., 2007), *k-means outlier detection* (KMOD) (Aggarwal, 2013), one class SVM(Schölkopf et al., 2001), *Local Outlier Factor* (LOF)(Breunig et al., 2000), and Naive Bayes(John and Langley, 1995). We also investigate *clustered label propagation* (CLP) and compare it to our CMN approach. CLP starts by applying k-means clustering to training data with both benign and malicious instances. It then labels each cluster based on the cluster's central-most point. At testing time, these points are added to the testing data as labeled points and label propagation, a semi-supervised learning algorithm, is used to label the testing data. Our experiments show that CMN outperforms these other methods, because it is able to model the local characteristics of the benign data.

A.2 Related Work

Anomaly detection methods typically fit into one of a number of broad categories:

- *Clustering-based* (He et al., 2003): Classification is based on clustering results. Instances that are not members in any cluster are considered as outliers.

- *Distance-based* (Hu and Sung, 2003): Classification is done based on the idea that the distances of anomaly data point to its K nearest neighbors are much larger than the distances of normal data points.

- *Density-based* (Jin et al., 2001): classification is done based on the density of the instances neighborhood. So if the neighborhood of the instance is dense then it is considered normal; otherwise it is anomaly.

- *Statistical-based* (Petrovskiy, 2003): classification is based on a probabilistic model; if the test instance is given sufficiently low probability by the model, it is considered an anomaly.

Distance-based methods provide more local details than clustering-based methods (Aggarwal, 2013). In other words, distance based methods can detect outliers that lie within noisy data. We review two popular distance-based methods: nested-loop method and grid-based method.

In the nested-loop (Aggarwal, 2013) method, two arrays of data instances are maintained, the first contains candidate outliers and the other contains the instances to which these candidates are compared. When more than k data instances are identified to lie within a distance d from an instance in the first array, that instance is marked as a normal.

In the grid-based (Aggarwal, 2013) method, the data set is divided into cells with length: $\frac{D}{(2\sqrt{d})}$ where d is the dimensionality, and D is the distance. Then Layer-1 neighbors are defined by all the intermediate neighbor cells and Layer-2 neighbors are defined by the cells within 3 cells of a certain cell. The outliers then are determined by four If-Then-Else conditions: 1) if the number of instances in the examined cell is greater that some threshold M, then all the instances in the cell are normal; 2) if there are M instances in a cell and its Layer-1 neighbors, then all the instances in the cell are normal; 3) if there are less than M instances inside a cell, its Layer-1 neighbor cells, and its Layer-2 neighbor cells, all the instances in the cell are outliers; 4) otherwise, if there are M or more instances in the cell and its Layer-2 neighbor cells and those instances are within distance D, then they are normal instances.

Knorr *et al.* (Knorr et al., 2000) suggest a method that combines the distance and density approaches, wherein a test instance is considered an anomaly if the fraction of training instances that lie within a given radius is below some threshold. However, this approach requires setting the radius and the threshold, which may be hard to estimate *a priori*. In our approach, we do not set a threshold, instead it is calculated from the data.

Distance-based methods are not effective when the data set contains clusters of data with different densities. Density-based methods partition the data space into regions then compute local densities of particular regions and declare instances in low density regions as outliers. Breunig *et al.* (Breunig et al., 2000) present a density-based approach called the *local outlier factor* (LOF) in which data instance are assigned an outlier score which is equal to the ratio of average local density of the k-nearest neighbors of the instance and the local density of the data instance itself. However, anomalous instances that are not far enough from other instances are missed and the use of k-nearest neighbors can be very expensive with certain distance functions.

High dimensional data is problematic for the clustering, distance, and density based methods (Aggarwal, 2013). Another challenge is determining the appropriate level of locality. A fully global analysis may miss important outliers, whereas a fully local analysis may miss small clustered groups of outliers (Aggarwal, 2013).

The statistical approach of (Petrovskiy, 2003) has the following advantages: first, it is mathematically justified by using well-established statistical methods to detect outliers; second, if the model represents normal data, then the results are accurate and the methods are very efficient; third, there is no need to keep the training data after building the model. The challenge is how to build the best model that represents the normal data from the training data.

We present a host-based anomaly detection algorithm, CMN, which uses only normal data to train the model. In CMN, the clustering-based approach is combined with the statistical approach. The training data is clustered and a separate Markov network is built from each cluster. All of the networks are then used as an ensemble to classify the instances such that an outlier is an instance that does not "fit" into any of the models. In this way, we produce models that together have better ability to detect malicious data when compared to using Markov networks without clustering. Clustering allows the models to consider the local characteristics of the data and not be confused by the possible heterogeneity of the entire dataset. Empirical study shows that it is more effective for anomaly detection when compared to other similar approaches.

A.3 Our approach

We present two novel approaches for anomaly detection: Clustered Markov Networks and Clustered Label Propagation.

A.3.1 Clustered Markov Networks

Our Markov network is trained using the benign training data only. Each factor in it is defined over two features and is built by counting the number of co-occurrences of each value of its first feature with each value of its second feature. Then, the network is used as a model to predict the probabilities of the instances to be examined. If the probability of the instance is high, then the instance is classified as benign. Conversely, if its probability is lower than some threshold h, then it is classified as malicious. The threshold h can be specified in multiple ways. In all of our networks, we take h to be the average probability of all the testing instances plus the variance of their probabilities.

We call the previous approach a "global" Markov network, because it learns from the whole training dataset without considering that the training data may have multiple types of benign characteristics. The results show that the false negative rate is high and the true positive rate is low, which indicates that the model is missing multiple types of malicious data. This drawback may be due to the heterogeneity of the training data, which may lead to Markov networks that overestimate the probability of anomalous data.

To solve this problem, we use the Clustered Markov Networks (CMN) approach. We assume that there are multiple types of benign data, and that each type will tend to be more homogeneous than the data as a whole. In CMN, the training data are first clustered using k-means, then "local" Markov Networks are trained on each cluster independently, as shown in Algorithm 7. Line 3 shows that k-means clustering is applied to the training data D_{tr}. In Line 4, a Markov network is built using each cluster c and the resulting networks are stored in an ensemble E.

During testing, each local network independently predicts the probability of the instance. Then, the resulting probability predictions are averaged to obtain an overall prediction. If the average probability of an instance is less than the threshold h, it is classified as malicious; otherwise, it is classified as benign, as shown in Algorithm 8. In Line 2, each testing instance d_i is retrieved from the testing dataset D_t. Then, in Lines 3 and 4, each Markov network M_j in ensemble E calculates the probability p_{ji} of d_i. In Line 5, the average probability, P_{avg} is calculated. In Lines 6 to 9, the predicted classification is made by comparing P_{avg} to the threshold h.

Clustering boosts the performance of the networks. It allows Markov Networks to be built from data that have similar characteristics. This leads to a classifier that is more capable of detecting malicious data. The experimental results show that CMN is able to detect a higher number of malicious instances.

Algorithm 7: Training Clustered Markov Networks

```
input : Training data $D_{tr}$, consisting of "benign" data; $k$, the number of clusters.
output: An ensemble $E$ of Markov networks.
1 begin
2     $E \leftarrow \emptyset$
3     for $c \in k$-Means($D_{tr}$) do
4         $E \leftarrow E \cup$ BuildMN($c$)
```

Algorithm 8: Testing with Clustered Markov Networks

```
input : Testing data $D_t$, consisting of "benign" data; $E$, an ensemble of Markov
        networks; $h$, the threshold.
output: $\widehat{D_t} = \{\widehat{d_i}\}$, the predicted classifications of the data points $D_t$.
1 begin
2     for $d_i \in D_t$ do
3         for $M_j \in E$ do
4             $p_{ji} \leftarrow$ Test($M_j, d_i$)
5         $P_{avg} \leftarrow \frac{\sum_{j=1}^{|E|} p_{ji}}{|E|}$
6         if $P_{avg} > h$ then
7             $\widehat{d_i} \leftarrow -1$
8         else
9             $\widehat{d_i} \leftarrow 1$
```

A.3.2 Clustered Label Propagation

Next, we present a variation of the label propagation (LP) algorithm called Clustered Label Propagation (CLP). In this method, k-means clustering is combined with the label propagation algorithm. The procedure is shown in Algorithm 4.

Here, the training dataset, which contains both benign and malicious instances, is intelligently selected using clustering. Initially, there are two empty sets: a set P_l of labeled points and the corresponding label set Y_l. k-means clustering is applied to the training dataset D_{tr} to create k clusters $c_1, c_2, \ldots, c_k$. For each cluster, the closest point p_{min} to the centroid p_c (using Euclidean distance) is selected and inserted into P_l while its corresponding label y is inserted into Y_l. Finally, label propagation is applied using P_l and Y_l to label the unlabeled points.

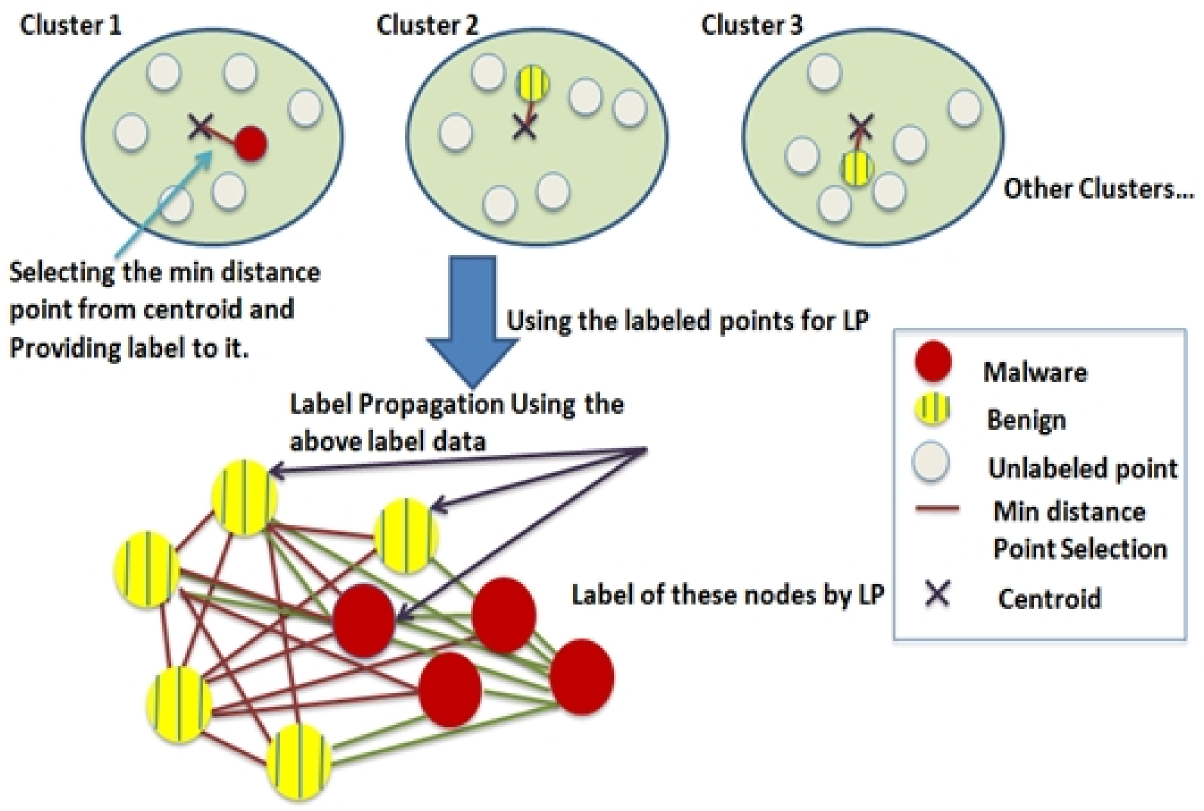

Cluster 1
Cluster 2
Cluster 3
Other Clusters...
Selecting the min distance point from centroid and Providing label to it.
Using the labeled points for LP
Label Propagation Using the above label data
Label of these nodes by LP
Malware
Benign
Unlabeled point
Min distance Point Selection
Centroid

Figure A.1: Clustered Label Propagation (CLP)

Figure A.1 illustrates the clustered label propagation method. The top of the figure shows the clusters for the training dataset. For each cluster, the closest point to the centroid and its label are selected. In the figure, the centroid is represented by a cross. One can see in cluster 1 that the closest point to its centroid is labeled as malicious (i.e., the shaded circle). Similarly, cluster 2 and 3 have points nearest to the centroid which are benign (i.e., the striped circles). These labeled points are then used to run the label propagation (LP), which labels all the testing data.

The label propagation (LP) algorithm is an efficient algorithm for classification. However, LP needs some labeled data initially and finding these labeled data can be an expensive

Algorithm 9: Training Clustered Label Propagation (CLP)

```
input  : Training Dataset $D_{tr}$.
output : Labeled point set $P_l$ and its corresponding label set $Y_l$.
1 begin
2     $P_l$={ }, $Y_l$={ }
3     $C_s \leftarrow$ Cluster ($D_{tr}$) where $C_s$ ={$c_1$,$c_2$,...,$c_k$}
4     for each cluster $c_i \in C_s$ do
5         $p_c \leftarrow$ Center($c_i$)
6         $p_{min} \leftarrow$ FindMinDistPoint( $p_c$ )
7         $y \leftarrow$ Label($p_{min}$)
8         $Y_l \leftarrow Y_l \cup \{y\}$
9         $P_l \leftarrow P_l$ { $p_{min}$ }
```

Algorithm 10: Testing with Clustered Label Propagation (CLP)

```
input  : Testing Dataset $D_t$, labeled point set $P_l$ and its corresponding label set $Y_l$.
output : The predicted labels for unlabeled points.
1 begin
2     $D_t \leftarrow D_t \cup P_l$
3     Run label propagation with $Y_l$
```

task. Clustered label propagation (CLP) has introduced the clustering feature into the label propagation algorithm. CLP requires fewer labeled data instances and these instances are more influential to the label propagation algorithm. Thus, CLP reduces the total number of initial labels needed. (Note that LP and CLP require both benign and malicious data for training and there are few instances in the malicious class.)

A.3.3 K-means Outlier Detection (KMOD)

For this approach (Aggarwal, 2013), only benign instances are used to train the model. During training, k-means clustering is first applied to the dataset. Then, to classify testing instances. We use two variations: KMOD using Nearest Centroid's Cluster (KMOD-NCC) and KMOD using All Clusters (KMOD-AC). In KMOD-NCC, for each test instance, the cluster with the nearest centroid is found using Euclidean distance. If the distance between the point and

Table A.1: Dataset Sizes

	Training		Testing	
Method	# Benign	# Malicious	# Benign	# Malicious
MN	20,700	0	2,300	1,000
CMN	20,700	0	2,300	1,000
KMOD-NCC	20,700	0	2,300	1,000
KMOD-AC	20,700	0	2,300	1,000
SVM	20,700	0	2,300	1,000
LOF	20,700	0	2,300	1,000
LP	19,837	863	3,163	137
CLP	19,837	863	3,163	137
Naive Bayes	19,837	863	3,163	137

the centroid of the cluster is smaller than the radius of the cluster (the distance between the centroid and farthest point of its cluster), it is normal. Otherwise, it is classified as anomalous. There is a limitation to KMOD-NCC. Consider a test instance that is just outside a small cluster but inside a larger cluster. The centroid of the small cluster is closer than the centroid of the large cluster. Therefore, the instance is classified as an anomaly while it is inside the large cluster. To prevent this kind of error, we use KMOD-AC. In KMOD-AC, for each test instance, if it is inside any cluster then it is normal. Otherwise, it is classified as anomalous. An instance is considered inside a cluster if the distance to its centroid is smaller than its radius.

A.4 Experimental Results

We have evaluated the discussed methods on a dataset of system calls, where each instance is a snapshot of system calls collected from a running executable during its first two minutes of execution. Each instance is a vector of continuous real values representing the frequency of system calls during execution; there are a total of 284 possible system calls. Each instance is also labeled as either benign or malicious. The benign samples were collected from live feeds of all files that crossed a corporate network border. In order to mitigate the risk of having potentially malicious samples in this feed, the samples were filtered through anti-virus scanners before labeling the data stream as "good". For malware, we used a daily feed from Arbor Networks, a security company that collects malicious software from the many network sensors that they own. The dataset contains 24,000 instances, of which 23,000 are benign and 1,000 are malicious.

We have compared the discussed methods with one class SVM (SVM), Local Outlier Factor (LOF), and Naive Bayes. Weka (Hall et al., 2009) was used to implement LOF, Naive Bayes, and k-means in the CMN and KMOD experiments. The CLP and LP experiments were implemented in Python using the Scikit-Learn(Pedregosa et al., 2011) implementation of label propagation and k-means. LibSVM(Chang and Lin, 2011) was used to implement SVM.

We have calculated the number of true/false positive/negative. A brief description of the used measurements is in Table A.2. To test the overall performance of the methods we have used F_β-measure which is calculated by:

$$F_\beta = \frac{(1+\beta^2) * TP}{(1+\beta^2) * TP + \beta^2 * FN + FP},$$

where we set $\beta = 2$. In our case, the ability to classify malicious data is more important than the ability to classify benign data. Because the misclassified malicious data may have harmful results. Using F_2 allows us to put more emphasis on the successful and failure attempts to protect the host from the malicious data because it weighs both true positive (TP) and false negative (FN) higher than false positive (FP).

A.4.1 Markov Network Setup and Analysis

Each instance of the training and testing data sets is a vector of features. These features correspond to system calls and their values are continuous real numbers. We discretize each feature into four bins: *small* (S), *medium* (M), *large* (L), and *very large* (V). For example, the feature that corresponds to system call *NtOpenFile* is discretized into (-inf-0.006692], (0.006692-0.010131], (0.010131-0.020283], (0.020283-inf) for S, M, L, and V respectively.

Table A.2: Measurements

TPR	$\frac{TP}{TP+FN}$
TNR	$\frac{TN}{TN+FP}$
FPR	$\frac{FP}{FP+TN}$
FNR	$\frac{FN}{FN+TP}$
TP	The number of instances that are malicious and classified as malicious.
FP	The number of instances that are benign but classified as malicious.
TN	The number of instances that are benign and classified as benign.
FN	The number of instances that are malicious but classified as benign.

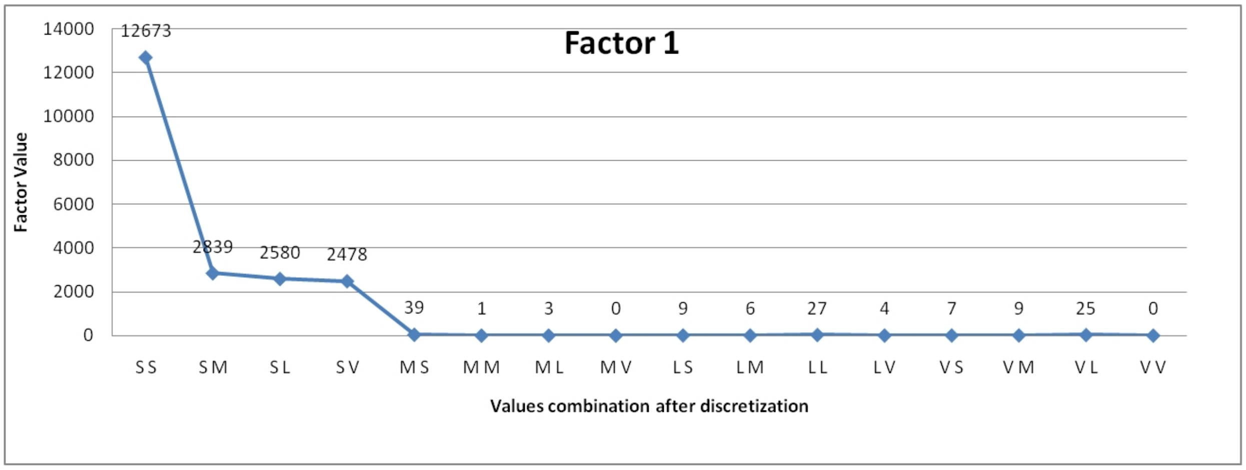


Figure A.2: Factor 1: *NtAccessCheckByTypeResultListAndAuditAlarm* and *NtOpenFile*

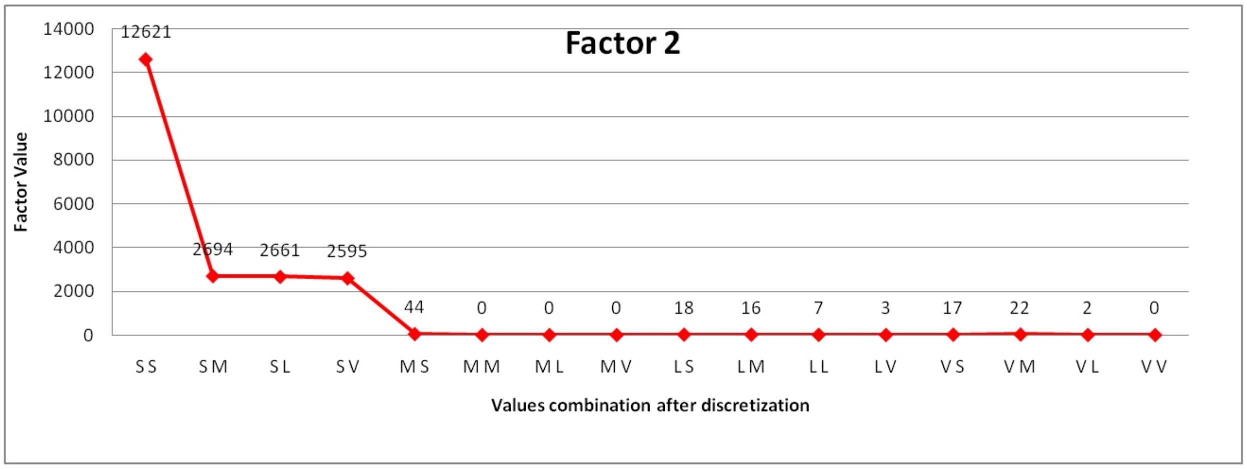


Figure A.3: Factor 2: *NtOpenFile* and *NtUnmapViewOfSection*

Markov network consists of variables and factors. Each variable corresponds to a feature. Since we have discretized the features, the values of each variable will be in the range of $\{S,\ M,\ L,\ V\}$. Our factors measure the distribution between each two variables by capturing the occurrence frequency of each combination of values in the training data set. For example, Figure A.2 shows the distributions captured by the factor of variables $NtAccessCheckByTypeResultListAndAuditAlarm$ and $NtOpenFile$. Since our Markov networks are built from benign training data, the factors capture the benign characteristics. i.e, a high factor value corresponds to a highly frequent combination of variables values in the benign training data set. For example, in Figure A.2 the combination of SS where $NtAccessCheckByTypeResultListAndAuditAlarm{=}S$ and $NtOpenFile{=}S$ is highly frequent and very common in the training data with a factor value of 12673 occurrences. On the other hand, the combination of VV is not frequent with a value of zero.

To calculate the probability of a test instance, we multiply the values of factors that correspond to the combinations of values of the test instance's features vector.If the instance contains many combinations that correspond to high factors values, then the probability will be high. On the other hand, if many combinations correspond to low factors values then the probability will be low. In other words, the instance with combinations that are highly frequent in the training data set has high probability and is more likely to be benign instance and vice versa. For example, Let X be an instance with the following combinations of features values: (1) $NtAccessCheckByTypeResultListAndAuditAlarm{=}S$, (2) $NtOpenFile{=}S$, and (3) $NtUnmapViewOfSection{=}S$. The factor of variables (1) and (2) gives a value of 0 for combination SS and the factor of variables (1) and (3) gives the combination SS a value of 0 too as in Figure A.3. Both values are low, in other words, both combinations are not frequent in the training data set. As a result, the probability of the instance is low. So, The instance is considered malicious.

A.4.2 Overall Performance

We trained the MN, CMN, KMOD-NCC, KMOD-AC, LOF, and SVM algorithms using 20,700 benign instances and tested on the remaining 1,000 malicious and 2,300 benign instances. The LP, CLP, and Naive Bayes approaches were trained with 19,837 benign and 863 malicious randomly-selected instances. These numbers are summarized in Table A.1.

For CLP, we have set the number of clusters k to 2,070. In CLP, we take only one point (the nearest point to the centroid) with its label from each cluster. If there are a smaller number of clusters, then there are fewer labeled points for the label propagation algorithm. So, the algorithm does not spread the labels to the unlabeled points appropriately with these fewer labeled points. For KMOD-NCC and KMOD-AC, we have tested multiple values of k. The best results are produced by using 1700 clusters for KMOD-NCC and 1600 clusters for KMOD-AC. For LOF, we varied the lower bound on k nearest neighbors. The best results are produced by setting $k = 4$. For SVM, we have used radial basis kernal function. For the CMN, we varied the number of clusters k from 2 to 10 and have reported the results in Figure A.5.

Our results show that CMN outperforms the other methods in terms of maximizing the true positive rate and F_2 measure while minimizing the false negative and false positive rates. The best results of all approaches are shown in Table A.3. The value of F_2 in CMN is 0.828 and it is higher than MN, KMOD-NCC, KMOD-AC, LP, CLP, LOF, Naive Bayes, and SVM which all have F_2 values of 0.483, 0.401, 0.126, 0.113, 0.0, 0.027, 0.5447, and 0.453 respectively. This confirms our claim that using clustering and then building a Markov network on each cluster increases the number of discovered true positives without adding false positives.

A.4.3 Sensitivity to Mislabeled Data

We have also investigated the effect of having label noise in the training data. Here, label noise refers to malicious instances that have been labeled as benign and benign instances that have been labeled as malicious. We have varied the percentage of noise from 0.01% to 1% and

Table A.3: Experimental Results

Method	F_2	TPR	FPR	FNR	TNR
CMN	**0.828**	**0.845**	0.112	**0.155**	0.887
MN	0.483	0.571	0.582	0.429	0.417
KMOD-NCC	0.401	0.361	0.0635	0.639	0.937
KMOD-AC	0.126	0.104	0.0161	0.896	0.983
LP	0.113	0.188	0.179	0.811	0.82
CLP	0.0	0.0	0.172	1.0	0.827
LOF	0.027	0.022	0.0222	0.978	0.978
Naive Bayes	0.5447	0.489	**0.0**	0.5109	**1.0**
SVM	0.453	0.58	0.7935	0.42	0.2065

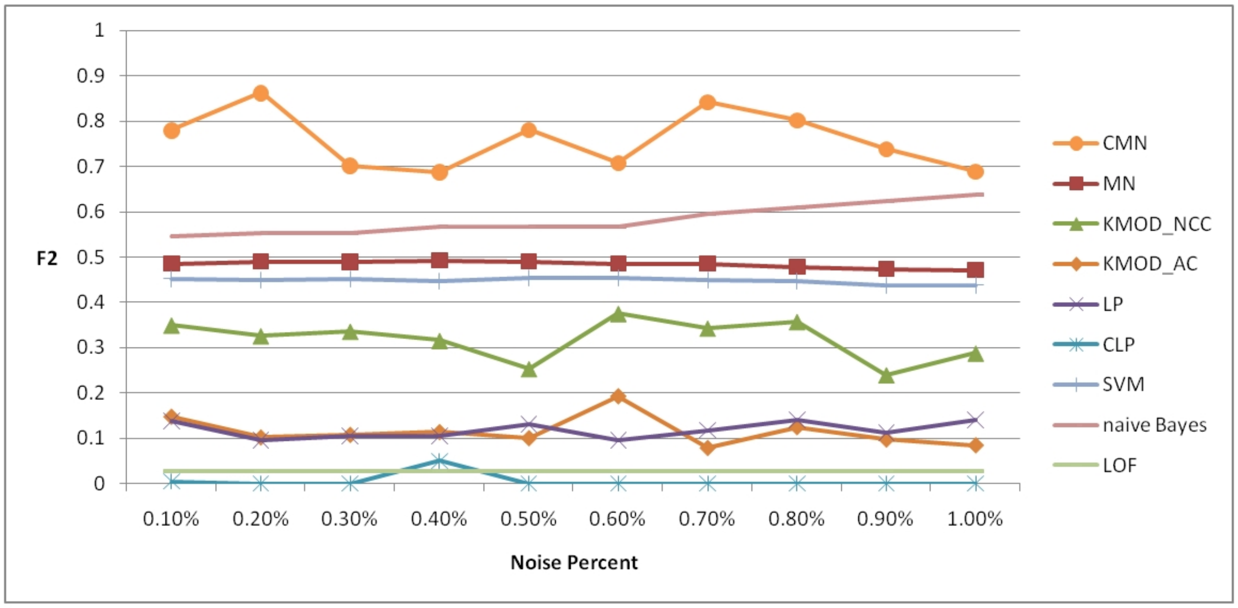


Figure A.4: Effect of Label Noise on F_2 Measure

reported the results in Figure A.4. The x-axis is the percentage of noisy data embedded in the training data and the y-axis is the F_2 values for each method. The figure shows that CMN outperforms MN, KMOD-NCC, KMOD-AC, LP, CLP, LOF, Naive Bayes, and SVM. For example, when 1% of the training data is mislabeled, F_2 value of CMN is 0.689 while it is 0.471, 0.288, 0.084, 0.055, 0.0, 0.0275, 0.636, and 0.438 for MN, KMOD-NCC, KMOD-AC, LP, CLP, LOF, Naive Bayes, and SVM respectively. Naive Bayes results are close to CMN that is because generally Naive Bayes classifiers are robust to isolated noise points(Tan et al., 2005). This shows that CMN is less sensitive to noise.

A.4.4 Sensitivity to Number of Clusters

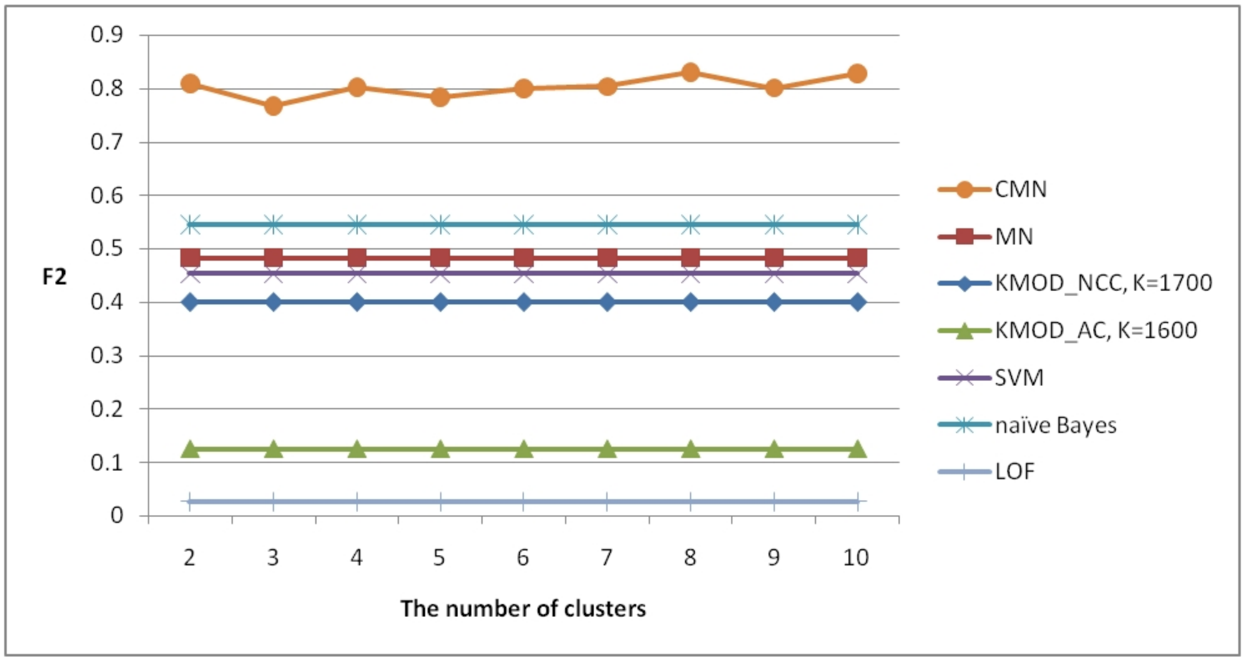


Figure A.5: Effect of the number of Clusters on F_2 Measure

Additionally, we have varied the number of clusters k in CMN from 2 to 10 to estimate the effect of cluster size on performance. F_2 results are reported in Figure A.5. The x-axis represents k and the y-axis represents F_2. The value of F_2 has increased from 0.5 using MN to

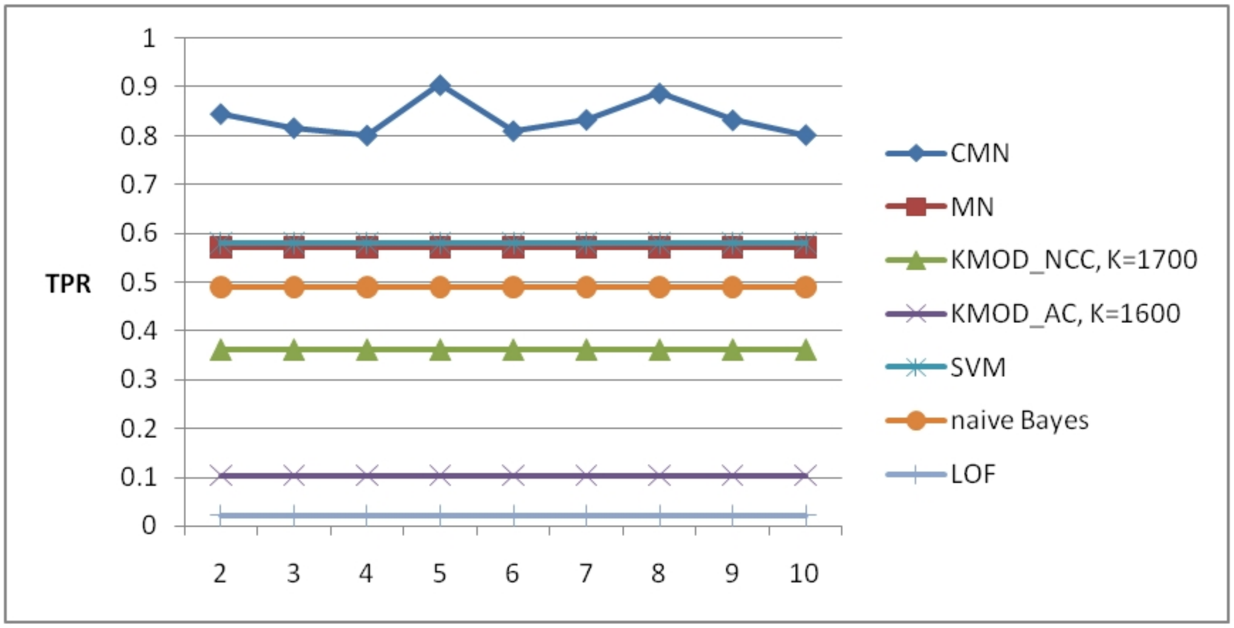


Figure A.6: Impact of varying the number of clusters on TPR

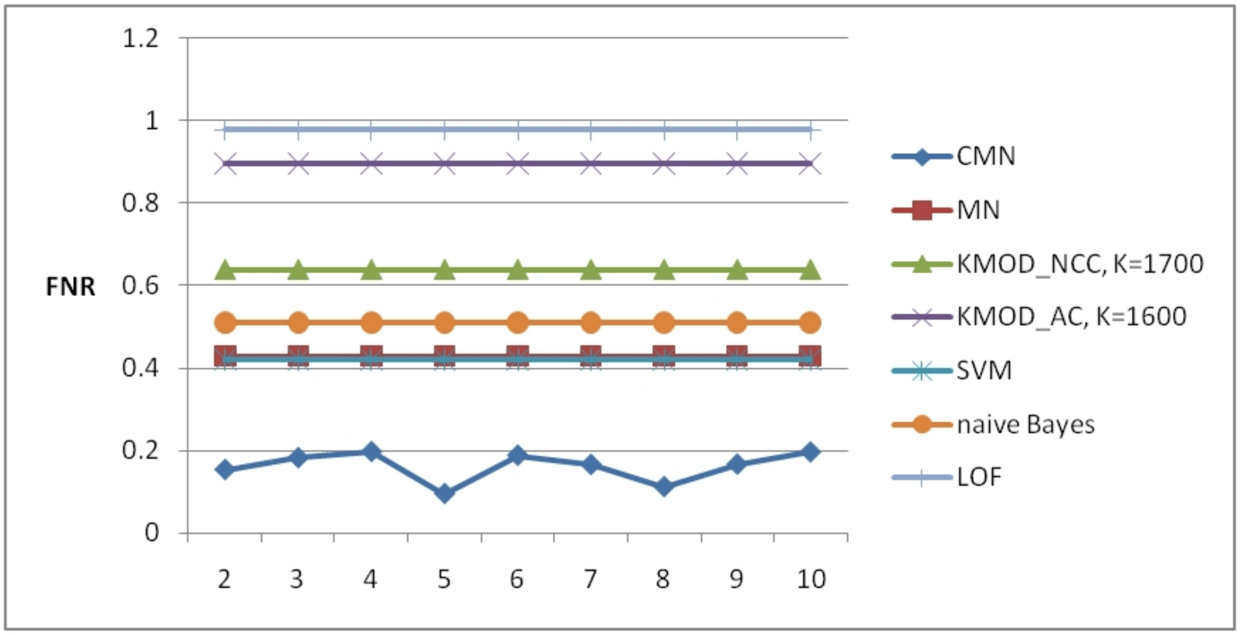


Figure A.7: Impact of varying the number of clusters on FNR

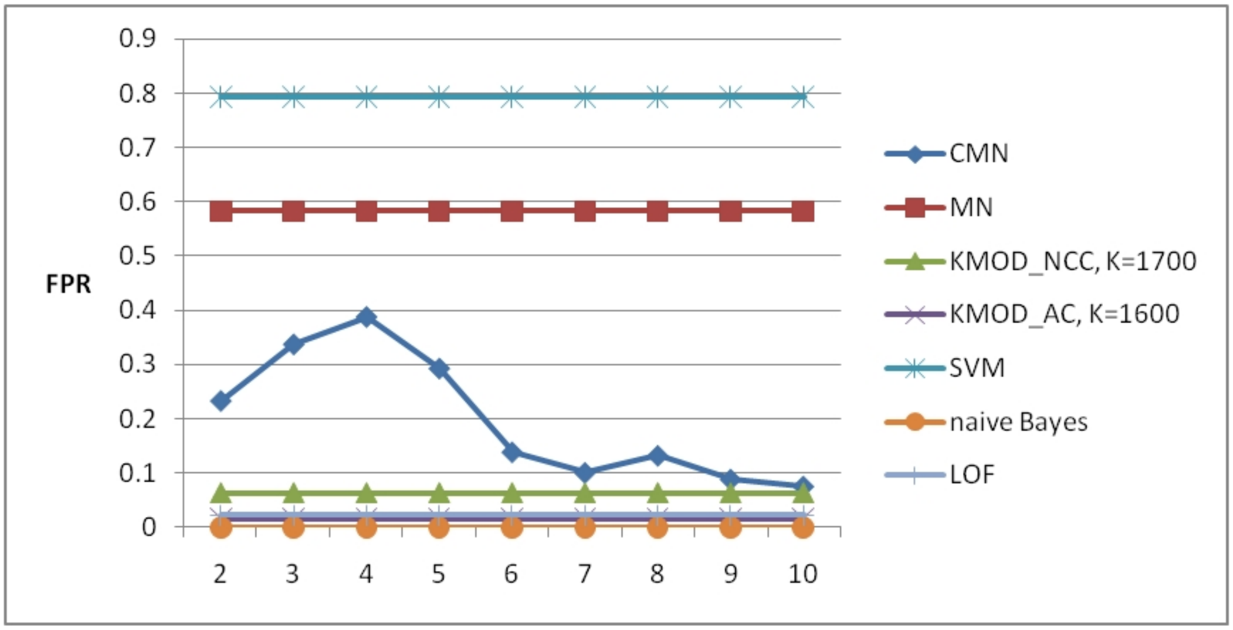


Figure A.8: Impact of varying the number of clusters on FPR

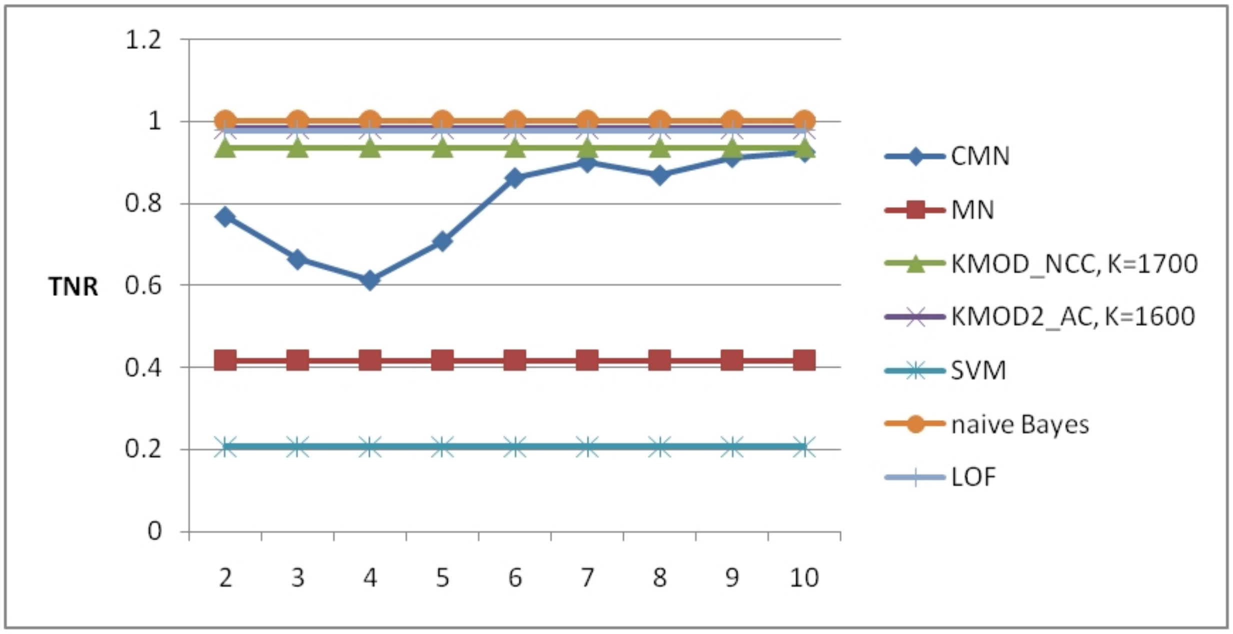


Figure A.9: Impact of varying the number of clusters on TNR

0.8 on average using CMN. MN is represented in Figure A.5 by a straight line, because it does not use the clustering method. We have varied k in KMOD-NCC and KMOD-AC from 2 to 2000. Their F_2 results that are reported in Figure A.5 are the highest values. They are produced using 1700 clusters for KMOD-NCC and 1600 clusters for KMOD-AC. F_2 results are 0.401 and 0.126 for KMOD-NCC and KMOD-AC respectively. For LOF, Naive Bayes, and SVM, F_2 results are 0.027, 0.5447, and 0.453 respectively.

We also have reported TPR, FPR, FNR, and TNR in Figure A.6,A.7,A.8,A.9. The x-axis is the number of clusters used by CMN. Figure A.6 shows that CMN's value of TPR using 10 clusters is 0.8. So, in terms of TPR, CMN outperforms MN, KMOD-NCC, KMOD-AC, LOF, Naive Bayes, and SVM by 0.229, 0.439, 0.696, 0.781, 0.314, and 0.223 respectively. Figure A.7 shows that FNR of CMN using 10 clusters is 0.197. Therefore, CMN outperforms MN, KMOD-NCC, KMOD-AC, LOF, Naive Bayes, and SVM in terms of FNR by 0.232, 0.442, 0.699, 0.824, 0.357, and 0.266 respectively.

Figure A.8 and Figure A.9 show the performance of CMN as compared to other methods in terms of FPR and TNR. Figure A.8 shows that for k greater than 4, as k is increasing, CMN's FPR is decreasing. Figure A.9 shows that for k greater than 4, as k is increasing, CMN's TNR is increasing too. For example, using 10 clusters FPR of CMN is 0.076 and TNR is 0.924. FPR values for MN, KMOD-NCC, KMOD-AC, LOF, Naive Bayes, and SVM are 0.583, 0.063, 0.016, 0.022, 0, and 0.793 respectively and TNR values of MN, KMOD-NCC, KMOD-AC, LOF, Naive Bayes, and SVM are 0.417, 0.937, 0.984, 0.9778, 1, and 0.207 respectively. So KMOD-NCC, KMOD-AC, LOF, and Naive Bayes are outperforming CMN by 0.013, 0.06, 0.211, and 0.233 respectively in terms of both FPR and TNR. This difference is not significant as compared to the large difference in terms of TPR and FNR. Moreover, TPR and FNR are more important in our application because the ability to classify malicious data is more important than the ability to classify benign data as stated above.

A.5 Conclusion

We have presented a novel anomaly detection approach: Clustered Markov Networks (CMN) in which we first use k-means clustering, then build an ensemble of Markov networks (one per cluster) which independently predict the probabilities of the test instances, and if the average probability of the test instance passes a threshold, it is classified as normal.

We experimentally compared our proposed approaches to several other approaches on a real dataset of system calls, and show that CMN outperforms the other methods and it is less sensitive to noise.

In the future, we want to improve our approaches by analyzing the quality of both the clusters and the Markov networks. Then we can use this analysis to discard or combine some clusters or Markov networks.

REFERENCES

Aggarwal, C. (2013). *Outlier Analysis.* Springer-Verlag New York Incorporated.

Aggarwal, C. C. and P. S. Yu (2010). On classification of high-cardinality data streams. In *SDM*, pp. 802–813. SIAM.

Al-Khateeb, T., M. M. Masud, K. Al-Naami, S. E. Seker, A. M. Mustafa, L. Khan, Z. Trabelsi, C. C. Aggarwal, and J. Han (2016). Recurring and novel class detection using class-based ensemble for evolving data stream. *IEEE Trans. Knowledge and Data Engineering (TKDE) 28*(10), 2752–2764.

Al-Naami, K., G. Ayoade, A. Siddiqui, N. Ruozzi, L. Khan, and B. Thuraisingham (2015, Dec). P2v: Effective website fingerprinting using vector space representations. In *2015 IEEE Symposium Series on Computational Intelligence*, pp. 59–66.

Al-Naami, K., S. Chandra, A. M. Mustafa, L. Khan, Z. Lin, K. W. Hamlen, and B. M. Thuraisingham (2016). Adaptive encrypted traffic fingerprinting with bi-directional dependence. In *Proc. 32nd Annual Computer Security Applications Conf. (ACSAC)*, pp. 177–188.

Alonso, O., D. Fetterly, and M. Manasse (2013). *Duplicate News Story Detection Revisited*, pp. 203–214. Berlin, Heidelberg: Springer Berlin Heidelberg.

Alsulami, B. S., M. F. Abulkhair, and F. E. Eassa (2012). Near duplicate document detection survey. *Bassma S Alsulami et al, International Journal of Computer Science & Communication Networks 2*(2), 147–151.

Alzahrani, S., N. Salim, C. K. Kent, M. S. Binwahlan, and L. Suanmali (2010, Nov). The development of cross-language plagiarism detection tool utilising fuzzy swarm-based summarisation. In *2010 10th International Conference on Intelligent Systems Design and Applications*, pp. 86–90.

Ammar, W., G. Mulcaire, Y. Tsvetkov, G. Lample, C. Dyer, and N. A. Smith (2016). Massively multilingual word embeddings. *arXiv preprint arXiv:1602.01925*.

Araujo, F. (2016). *Engineering Cyber-deceptive Software.* Ph. D. thesis, The University of Texas at Dallas, Richardson, Texas.

Araujo, F., K. W. Hamlen, S. Biedermann, and S. Katzenbeisser (2014). From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation. In *Proc. 21st ACM Conf. Computer and Communications Security (CCS)*, pp. 942–953.

Azar, E. E. (1980). The conflict and peace data bank (COPDAB) project. *Journal of Conflict Resolution 24*(1), 143–152.

Baron, M. (1999). Convergence rates of change-point estimators and tail probabilities of the first-passage-time process. *Canadian J. of Statistics 27*, 183–197.

Baron, M. I. (2000). Nonparametric adaptive change point estimation and on line detection. *Sequential Analysis 19*(1-2), 1–23.

Basseville, M. and I. V. Nikiforov (1993). *Detection of Abrupt Changes: Theory and Application*. Englewood Cliffs, NJ: PTR Prentice-Hall, Inc.

Baum, M. A. and Y. M. Zhukov (2015). Filtering revolution reporting bias in international newspaper coverage of the Libyan civil war. *Journal of Peace Research*, 0022343314554791.

Bay, S. D., D. F. Kibler, M. J. Pazzani, and P. Smyth (2000). The uci kdd archive of large data sets for data mining research and experimentation. *SIGKDD Explorations 2*, 81.

Bengio, Y., O. Delalleau, and N. Le Roux (2006). Label propagation and quadratic criterion. In O. Chapelle, B. Schölkopf, and A. Zien (Eds.), *Semi-Supervised Learning*, pp. 193–216. MIT Press.

Bhattacharyya, G. K. and R. A. Johnson (1968). Nonparametric tests for shift at an unknown time point. *The Annals of Mathematical Statistics 39*(5), 1731–1743.

Bifet, A. and R. Gavaldà (2009). Adaptive learning from evolving data streams. In *Proceedings of the 8th International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis VIII*, IDA '09, Berlin, Heidelberg, pp. 249–260. Springer-Verlag.

Bifet, A., G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl (2010). MOA: Massive online analysis, a framework for stream classification and clustering. In *Journal of Machine Learning Research (JMLR) Workshop and Conference Proceedings, Volume 11: Workshop on Applications of Pattern Analysis*, pp. 44–50. Journal of Machine Learning Research.

Bifet, A., J. Read, B. Pfahringer, G. Holmes, and I. Zliobaite (2013). CD-MOA: Change detection framework for massive online analysis. In *IDA*, pp. 92–103.

Bird, S., E. Klein, and E. Loper (2009). *Natural Language Processing with Python* (1st ed.). O'Reilly Media, Inc.

Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Breunig, M. M., H.-P. Kriegel, R. T. Ng, and J. Sander (2000). Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, New York, NY, USA, pp. 93–104. ACM.

Broder, A. Z. (2000). Identifying and filtering near-duplicate documents. In *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, COM '00.

Brzezinski, D. and J. Stefanowski (2014). Combining block-based and online methods in learning ensembles from concept drifting data streams. *Information Sciences 265*(0), 50 – 67.

Chandola, V., A. Banerjee, and V. Kumar (2009, July). Anomaly detection: A survey. *ACM Comput. Surv. 41*(3), 15:1–15:58.

Chandra, S., A. Haque, L. Khan, and C. Aggarwal (2016). An adaptive framework for multistream classification. In *Proc. 25th ACM Int. Conf. Information and Knowledge Management (CIKM)*, pp. 1181–1190.

Chang, C.-C. and C.-J. Lin (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology 2*, 27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chen, J. and A. K. Gupta (2012). *Parametric Statistical Change Point Analysis: With Applications to Genetics, Medicine, and Finance.* Boston, MA: Birkhäuser.

Chen, M., K. Q. Weinberger, F. Sha, and Y. Bengio (2014). Marginalized denoising autoencoders for nonlinear representations. In *Proc. 31st Int. Conf. Machine Learning (ICML)*, pp. 1476–1484.

Deng, J., Z. Zhang, F. Eyben, and B. Schuller (2014). Autoencoder-based unsupervised domain adaptation for speech emotion recognition. *IEEE Signal Processing Letters 21*(9), 1068–1072.

Duh, K., C.-M. A. Yeung, T. Iwata, and M. Nagata (2013, March). Managing information disparity in multilingual document collections. *ACM Trans. Speech Lang. Process. 10*(1), 1:1–1:28.

D'Ignazio, C., R. Bhargava, E. Zuckerman, and L. Beck (2014). Cliff-clavin: Determining geographic focus for news. *NewsKDD: Data Science for News Publishing, at KDD 2014*.

Ester, M., H.-P. Kriegel, J. Sander, and X. Xu (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of 2nd International Conference on Knowledge Discovery and*, pp. 226–231.

Ferger, D. (1991). *Nonparametric change-point detection based on U-statistics.* Ph. D. thesis, University of Giessen.

Ferrero, J., L. Besacier, D. Schwab, and F. Agnès (2017). Using word embedding for cross-language plagiarism detection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 415–421. Association for Computational Linguistics.

Freeman, S., A. Bivens, J. Branch, and B. Szymanski (2002). Host-based intrusion detection using user signatures. In *Proceedings of the Research Conference. RPI, Troy, NY.*

Gibson, J., B. Wellner, and S. Lubar (2008). Identification of duplicate news stories in web pages. In *Proceedings of the 4th Web as CorpusWorkshop.*

Gottschalk, S. and E. Demidova (2017). Multiwiki: Interlingual text passage alignment in wikipedia. *ACM Transactions on the Web (TWEB) 11*(1), 6.

Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten (2009, November). The weka data mining software: an update. *SIGKDD Explor. Newsl. 11*(1), 10–18.

Hamlen, K. W., V. Mohan, M. M. Masud, L. Khan, and B. Thuraisingham (2009, April). Exploiting an antivirus interface. *Computer Standards & Interfaces 31*(6), 1182–1189.

Haque, A., L. Khan, and M. Baron (2016). SAND: Semi-supervised adaptive novel class detection and classification over data stream. In *Proc. 13th AAAI Conf. Artificial Intelligence*, pp. 1652–1658.

He, Z., X. Xu, and S. Deng (2003, June). Discovering cluster-based local outliers. *Pattern Recogn. Lett. 24*(9-10), 1641–1650.

Hinkley, D. V. (1970). Inference about the change-point in a sequence of random variables. *Biometrika 57*(1), pp. 1–17.

Hu, J. (2010). Host-based anomaly intrusion detection. In P. Stavroulakis and M. Stamp (Eds.), *Handbook of Information and Communication Security*, pp. 235–255. Springer Berlin Heidelberg.

Hu, T. and S. Y. Sung (2003, December). Detecting pattern-based outliers. *Pattern Recogn. Lett. 24*(16), 3059–3068.

Jin, W., A. K. H. Tung, and J. Han (2001). Mining top-*n* local outliers in large databases. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, New York, NY, USA, pp. 293–298. ACM.

John, G. H. and P. Langley (1995). Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, San Mateo, pp. 338–345. Morgan Kaufmann.

Khan, L., M. Awad, and B. Thuraisingham (2007, October). A new intrusion detection system using support vector machines and hierarchical clustering. *The VLDB Journal 16*(4), 507–521.

Knorr, E. M., R. T. Ng, and V. Tucakov (2000, February). Distance-based outliers: Algorithms and applications. *The VLDB Journal 8*(3-4), 237–253.

Koller, D., N. Friedman, L. Getoor, and B. Taskar (2007). Graphical models in a nutshell. In L. Getoor and B. Taskar (Eds.), *Introduction to Statistical Relational Learning.* MIT Press.

Kolter, J. Z. and M. A. Maloof (2007, December). Dynamic weighted majority: An ensemble method for drifting concepts. *J. Mach. Learn. Res. 8*, 2755–2790.

Kuncheva, L. I. (2013, May). Change Detection in Streaming Multivariate Data Using Likelihood Detectors. *Knowledge and Data Engineering, IEEE Transactions on 25*(5), 1175–1180.

Kuncheva, L. I. and W. J. Faithfull (2014, Jan). Pca feature extraction for change detection in multidimensional unlabeled data. *IEEE Transactions on Neural Networks and Learning Systems 25*(1), 69–80.

Leban, G., B. Fortuna, J. Brank, and M. Grobelnik (2014). Event registry: Learning about world events from news. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, New York, NY, USA, pp. 107–110. ACM.

Li, P., Y. Liu, and M. Sun (2013). Recursive autoencoders for ITG-based translation. In *Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP)*, pp. 567–577.

Mahajan, R., S. Gupta, and M. R. Bedi (2014). A survey of duplicate and near duplicate techniques. *International Journal of Engineering Research & Technology (IJERT) 5.*

Manku, G. S., A. Jain, and A. Das Sarma (2007). Detecting near-duplicates for web crawling. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, New York, NY, USA, pp. 141–150. ACM.

Manning, C. D., M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky (2014). The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55–60.

Markou, M. and S. Singh (2003). Novelty detection: A review. *Signal Processing 83*(12), 2481–2521.

Masud, M. M., T. M. Al-Khateeb, L. Khan, C. Aggarwal, J. Gao, J. Han, and B. Thuraisingham (2011). Detecting recurring and novel classes in concept-drifting data streams. In *Proc. 11th IEEE Int. Conf. Data Mining (ICDM)*, pp. 1176–1181.

Masud, M. M., J. Gao, L. Khan, J. Han, and B. Thuraisingham (2008, Dec). A practical approach to classify evolving data streams: Training with limited amount of labeled data. In *2008 Eighth IEEE International Conference on Data Mining*, pp. 929–934.

Mazhelis, O. (2006). One-class classifiers: A review and analysis of suitability in the context of mobile-masquerader detection. *South African Computer J. (SACJ) 36*, 29–48.

Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119.

Mustafa, A., A. Haque, L. Khan, M. Baron, and B. Thuraisingham (2014, Oct). Evolving stream classification using change detection. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on*, pp. 154–162.

Mustafa, A., M. Solaimani, L. Khan, K. Chiang, and J. Ingram (2013, Dec). Host-based anomaly detection using learning techniques. In *2013 IEEE 13th International Conference on Data Mining Workshops*, pp. 1153–1160.

Mustafa, A. M., G. Ayoade, K. Al-Naami, L. Khan, K. W. Hamlen, B. Thuraisingham, and F. Araujo (2017, Dec). Unsupervised deep embedding for novel class detection over data stream. In *2017 IEEE International Conference on Big Data (Big Data)*, pp. 1830–1839.

Pamulaparty, L., D. M. S. Rao, and D. C. G. Rao (2013). A survey on near duplicate web pages for web crawling. *International Journal of Engineering Research & Technology (IJERT)*, 2278–0181.

Parker, B. and L. Khan (2015, Jan). Detecting and tracking concept class drift and emergence in non-stationary fast data streams. In *Twenty-Ninth AAAI Conference on Artificial Intelligence.*

Parker, B., A. M. Mustafa, and L. Khan (2012). Novel class detection and feature via a tiered ensemble approach for stream mining. In *ICTAI*, pp. 1171–1178.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and D. E. (2011). Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research 12*, 2825–2830.

Petrovskiy, M. I. (2003, July). Outlier detection algorithms in data mining systems. *Program. Comput. Softw. 29*(4), 228–237.

Potthast, M., A. Barrón-Cedeño, B. Stein, and P. Rosso (2011, Mar). Cross-language plagiarism detection. *Language Resources and Evaluation 45*(1), 45–62.

PR, A. and S. MS (2015, Dec). Near-duplicate web page detection by enhanced tdw and simhash technique. In *2015 International Conference on Computing and Network Communications (CoCoNet)*, pp. 765–770.

Qahtan, A. A., B. Alharbi, S. Wang, and X. Zhang (2015). A PCA-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams. In *Proc. 21th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD)*, pp. 935–944.

Reiss, A. and D. Stricker (2012, June). Introducing a new benchmarked dataset for activity monitoring. In *Wearable Computers (ISWC), 2012 16th International Symposium on*, pp. 108–109.

Rinser, D., D. Lange, and F. Naumann (2013, September). Cross-lingual entity matching and infobox alignment in wikipedia. *Inf. Syst. 38*(6), 887–907.

Rupnik, J., A. Muhic, G. Leban, P. Skraba, B. Fortuna, and M. Grobelnik (2016). News across languages-cross-lingual document similarity and event tracking. *Journal of Artificial Intelligence Research 55*, 283–316.

Sakurada, M. and T. Yairi (2014). Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proc. 2nd Work. Machine Learning for Sensory Data Analysis (MLSDA).*

Schölkopf, B., J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson (2001, July). Estimating the support of a high-dimensional distribution. *Neural Comput. 13*(7), 1443–1471.

Schrodt, P. and J. Ulfelder (2009). Political instability task force worldwide atrocities dataset. *Lawrence, KS: Univ. Kansas, updated 8.*

Schrodt, P. A., J. Beieler, and M. Idris. Three's a charm?: Open event data coding with EL:DIABLO, PETRARCH, and the Open Event Data Alliance.

Schrodt, P. A. and D. Van Brackle (2013). Automated coding of political event data. In *Handbook of Computational Approaches to Counterterrorism*, pp. 23–49. Springer.

Smith, S. L., D. H. Turban, S. Hamblin, and N. Y. Hammerla (2017). Offline bilingual word vectors, orthogonal transformations and the inverted softmax. *arXiv preprint arXiv:1702.03859*.

Song, C., F. Liu, Y. Huang, L. Wang, and T. Tan (2013). Auto-encoder based data clustering. In *Proc. 18th Iberoamerican Congress Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications (CIARP)*, pp. 117–124.

Song, X., M. Wu, C. Jermaine, and S. Ranka (2007). Statistical change detection for multi-dimensional data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, New York, NY, USA, pp. 667–676. ACM.

Steorts, R., H. R. and S. Fienberg (2014, Apr). SMERED: A Bayesian Approach to Graphical Record Linkage and De-duplication. . In *"AISTATS"*.

Steorts, R. C. (2015). Entity resolution with empirically motivated priors. *Bayesian Anal. 10*(4), 849–875.

Street, W. N. and Y. Kim (2001). A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, New York, NY, USA, pp. 377–382. ACM.

Tan, P.-N., M. Steinbach, and V. Kumar (2005). *Introduction to Data Mining, (First Edition).* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Uyar, E. (2009). Near-duplicate news detection using named entities.

Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Machine Learning Research 11*, 3371–3408.

Wang, H., W. Fan, P. S. Yu, and J. Han (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, New York, NY, USA, pp. 226–235. ACM.

Wang, H., J. Yin, J. Pei, P. S. Yu, and J. X. Yu (2006). Suppressing model overfitting in mining concept-drifting data streams. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, New York, NY, USA, pp. 736–741. ACM.

Wang, P., H. Wang, X. Wu, W. W. 0009, and B. Shi (2007). A low-granularity classifier for data streams with concept drifts and biased class distribution. *IEEE Trans. Knowl. Data Eng. 19*(9), 1202–1213.

Weninger, F., S. Watanabe, Y. Tachioka, and B. Schuller (2014). Deep recurrent de-noising auto-encoder and blind de-reverberation for reverberated speech recognition. In *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4623–4627.

Xiao, C., W. Wang, X. Lin, J. X. Yu, and G. Wang (2011, August). Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst. 36*(3), 15:1–15:41.

Xie, J., R. B. Girshick, and A. Farhadi (2015). Unsupervised deep embedding for clustering analysis. *CoRR abs/1511.06335*, 478–487.

Xing, C., L. Ma, and X. Yang (2016). Stacked denoise autoencoder based feature extraction and classification for hyperspectral images. *Journal of Sensors 2016.*

Yang, Y., X. Wu, and X. Zhu (2005). Combining proactive and reactive predictions for data streams. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pp. 710–715. ACM.

Zhang, P., J. Li, P. Wang, B. J. Gao, X. Zhu, and L. Guo (2011). Enabling fast prediction for ensemble models on data streams. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, New York, NY, USA, pp. 177–185. ACM.

Zliobaite, I., A. Bifet, B. Pfahringer, and G. Holmes (2014, Jan). Active learning with drifting streaming data. *IEEE Transactions on Neural Networks and Learning Systems 25*(1), 27–39.

BIOGRAPHICAL SKETCH

Ahmad Mustafa finished his undergraduate studies from Yarmouk University, Irbid, Jordan in 2007. In 2010, he received a Master of Science from Jordan University of Science and Technology, Irbid, Jordan. Ahmad received his Master of Computer Science from The University of Texas at Dallas in 2015.

CURRICULUM VITAE

Ahmad M. Mustafa

February 26, 2018

Contact Information:

Department of Computer Science
The University of Texas at Dallas
800 W. Campbell Rd.
Richardson, TX 75080-3021, U.S.A.

Email: `ahmad.mustafa@utdallas.edu`

Educational History:

B.S., Computer Information Systems, Yarmouk Univeristy, 2007
M.S., Computer Science, Jordan University of Science and Technology, 2010
M.S., Computer Science, The University of Texas at Dallas, 2015

Employment History:

Research Assistant, The University of Texas at Dallas, May 2012 – present

RESEARCH INTERESTS:

Machine Learning, Data Stream Mining, Natural Language Processing, Cybersecurity

PUBLICATIONS:

A. Mustafa et al., "Unsupervised deep embedding for novel class detection over data stream," *2017 IEEE International Conference on Big Data (Big Data), Boston, MA, 2017, pp. 1830-1839.*

T. Al-Khateeb et al., "Recurring and Novel Class Detection Using Class-Based Ensemble for Evolving Data Stream," *in IEEE Transactions on Knowledge and Data Engineering, vol. 28, no. 10, pp. 2752-2764, Oct. 1 2016.*

A. Mustafa et al., "Evolving stream classification using change detection," *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, Miami, FL, 2014, pp. 154-162.*

A. Mustafa et al., "Host-Based Anomaly Detection Using Learning Techniques," *2013 IEEE 13th International Conference on Data Mining Workshops, Dallas, TX, 2013, pp. 1153-1160.*

B. Parker, A. M. Mustafa and L. Khan, "Novel Class Detection and Feature via a Tiered Ensemble Approach for Stream Mining," *2012 IEEE 24th International Conference on Tools with Artificial Intelligence, Athens, 2012, pp. 1171-1178.*