ON-LINE FAST KERNEL BASED METHODS FOR CLASSIFICATION OVER STREAM DATA (WITH CASE STUDIES FOR CYBER-SECURITY)

YE (GARY) CHEN

A thesis submitted to
Auckland University of Technology
in fulfillment of the requirements for the degree of
Doctor of Philosophy (PhD)

February, 2012

Knowledge Engineering and Discovery Research Institute
Faculty of Design & Creative Technologies

Primary Supervisor:  Prof. Nikola Kasabov
Secondary Supervisor: Dr. Russel Pears
Third Supervisor:  Dr. Tao Ban
8.1 Background ........................................ 108
  8.1.1 Multi-label Intrusions .......................... 110
8.2 Experiments and Discussion ........................ 110
  8.2.1 Experimental setup .............................. 110
  8.2.2 Simulation of Network Strings Examination: RCV1-V2
     Dataset ............................................. 111
  8.2.3 Internet Security Application: KDD’99 Datasets .... 114
8.3 Summary ............................................ 117

9 The Application of Dynamic Evolving Core Vector Machines to Solve
Face Membership Authentication Problem 119
  9.1 Background ........................................ 119
  9.2 Image Data Pre-processing .......................... 121
     9.2.1 LLE Eigenface ................................. 121
     9.2.2 Data Partition ................................. 124
  9.3 Experiment Results and Discussion ....................... 125
  9.4 Summary ............................................ 127

10 The Application of Meta Learning String Kernel MEBs System to
Solve Real World String Format Stream Data Modeling 128
  10.1 Experiment Setup .................................. 128
  10.2 Datasets and Data Pre-processing ....................... 129
     10.2.1 Spam Dataset ................................ 130
     10.2.2 Reuters-21578 Dataset ......................... 130
     10.2.3 Network Application Detection Data ............ 131
     10.2.4 e-News Categorisation Data ...................... 132
  10.3 Results ............................................ 132
     10.3.1 Meta Learning for Edit-Distance MEB Optimisation ... 132
     10.3.2 Meta Learning for Bag-of-Words MEB Optimisation ... 134
     10.3.3 Meta Learning for N-gram MEB Optimisation ....... 134
  10.4 Limitations ........................................ 135
  10.5 Summary ............................................ 137

11 Conclusion and future directions 138
  11.1 Summary of Achievements ............................ 138
DECLARATION

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgments), nor material which to a substantial extend has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

_Auckland, February, 2012_

Ye (Gary) Chen
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure 3.1</th>
<th>A graphical representation of an integrated global, local, and personalised multimodel system. For every single input vector, the outputs of the tree models are weighted (Kasabov, 2007a).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 3.2</td>
<td>An example of the linear separating hyperplanes in SVM. Note: the support vectors are encircled.</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>A SVM tree, where each node is a SVM (Pang &amp; Kasabov, 2004).</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>An example of two-class support vector machine decision tree (2-SVMT) for CNS tumor. Ellipses: Partitioning and SVM decision nodes, numbering indicates the sequence of decisions. Open circles: samples (patients) assigned to class 1, filled circles: class 2, corresponding to Table 1. Numbers inside circles indicates the number of patients in the end of each decision path. (Pang, Havukkala, &amp; Kasabov, 2006).</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>Deletion of the data using the hyperspheres (Katagiri &amp; Abe, 2006a).</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>Judging whether the data are inside of the hypercone or not (Katagiri &amp; Abe, 2006a).</td>
</tr>
</tbody>
</table>
Figure 3.7 Concept of sufficient spanning sets of the total scatter matrix (a), the between-class scatter matrix (b) and the projected matrix (c). The union set of the principal components $P_1$, $P_2$ or $Q_1$, $Q_2$ of the two data sets and the mean difference vector $\mu_1 - \mu_2$ can span the respective total or between-class scatter data space (a and b). The dimension for the component $m_{1i} - m_{2i}$ should not be removed (cross=incorrect) from the sufficient set of the between-class scatter data but retained in the set (circle=correct) (b). The projection and orthogonalisation of the original components $Q_{31}$, $Q_{32}$ yields the principal components of the projected data up to rotation (c). See the corresponding sections for detailed explanations ... 38

Figure 3.8 eTS+ as a neural network (Angelov, 2010) ... 40

Figure 3.9 An example of an EFuNN with a short term memory and feedback connections, adapted from Kasabov (2001) ... 42

Figure 3.10 Example of ECM Clustering algorithm. $x_i$: input vector (*), $C_{Gj}^k$: cluster center, $C_j^k$: cluster, $R_{uj}^k$: cluster radius (Kasabov & Song, 2002) ... 44

Figure 4.1 An example of exact MEB, Core set MEB, and Core set MEB expansion ... 54

Figure 4.2 MEB expansion ... 58

Figure 4.3 MEB contraction ... 60

Figure 5.1 The label hierarchy of network connection types adapted from KDD’99 dataset ... 70

Figure 5.2 Core set data approximation comparison: boundary on the MEB core set from the parent class A1 versus boundary on the proposed HCVM ($A1 \leq_h [A2, A3, A4]$) core set. 72

Figure 5.3 Synthetic data for HMC, where data points of different classes are represented as different colors ... 76

Figure 5.4 An example of using MEB for HMC, where * represents a MEB core set point. Note that this figure is best seen in color ... 77
Figure 6.1  Banana data set clustering (a) plot 200 banana shaped data points, (b) plot the cluster boundary in black lines formed by 70 classic MEB core vectors, (c) plot the new cluster data in the bend area of banana data and banana data cluster boundary formed by the classic MEB core vectors and 6 DE-CVM core vectors, (d) plot the new data surround the left hand side of the banana data and the DE-CVM core vectors.

Figure 7.1  String and Text Data: (a) Network traffic data produced by network applications. (b) Data from Reuters-21578 dataset. (c) Spam data which consists of ham and Spam E-mail messages (in every instance, the first character represents the class label followed by actual data).

Figure 7.2  String Data: Network traffic data produced by network applications (in every instance, the first character represents the class label of the network application, followed by actual network traffic data).

Figure 7.3  String meta-feature generation process

Figure 7.4  The procedure to employ meta learning for string classification

Figure 8.1  Cross validation tests of Algorithm 6 for parameter \( n \) determination on KDD’99 dataset.

Figure 8.2  (a) and (b) plot the Micro-Average F-measure and Macro-Average F-measure of different number of training dataset for both SVM and HCVM respectively.

Figure 8.3  ROC graphs with AUC results of 10 chunks training dataset with same size. The filled areas in (a) and (b) correspond to the AUC for SVM and HCVM respectively.

Figure 8.4  KDD99 data classification label replacement

Figure 8.5  Compare the classification accuracy with the winner of KDD’99 cup on each attack type.
Figure 8.6 (a) CPU time required by HCVM and SVM, (b) number of support vectors created by HCVM and SVM with different size of training dataset, respectively. (c) the error rate variation against the size of training dataset.

Figure 9.1 Comparison of membership LLE eigenfaces and PCA eigenfaces when $K = 10$ and $M = 20$.

Figure 9.2 Typical example of the binary LLE data partition.

Figure 9.3 A comparison of CVM and DE-CVM on classification accuracy under the condition of different membership group size.

Figure 9.4 A comparison of CVM and DE-CVM on number of core vectors under the condition of different membership group size.

Figure A.1 The NIDVS visualisation demonstration shows the global data flow in the left, and the panels and result tables are displayed on the right side of the demo interface.

Figure A.2 The NIDVS system design diagram.

Figure A.3 The network traffic visualisation of NIDVS, in which the green arrows represent normal data and red arrows indicate spam/intrusion.

Figure A.4 The NIDVS control panel.

Figure A.5 Detection results and accuracy.

Figure A.6 NID report including information about the source and target city of data package, the number of data packages passed by, and the number of spam/intrusion detected.

Figure A.7 Graph for real time intrusion detection report, where the $x$–axis represents the amount of data processed, and $y$–axis represents the detection accuracy in presents.

Figure B.1 The HCVM Prototype System.

Figure B.2 MEBs, core vectors and the corresponding boundaries visualisation.

Figure B.3 The control panel.

Figure B.4 MEB information list table.
| Table 1.1 | Stream processing vs Traditional processing | 2 |
| Table 4.1 | Summarisation of data sets characteristics | 61 |
| Table 4.2 | Summarisation of classifiers characteristics | 61 |
| Table 4.3 | Classification accuracy comparison for five methods applied on the six two-class datasets | 62 |
| Table 4.4 | Summarisation of classifiers’ parameters for six two-class datasets | 62 |
| Table 4.5 | Classification accuracy comparison for five methods applied on the six multi-class datasets | 63 |
| Table 4.6 | Summarisation of classifiers’ parameters for six multi-class datasets | 63 |
| Table 4.7 | Comparison of training CPU time (in seconds). OM indicates out of memory caused by size of data input that was too large. | 64 |
| Table 4.8 | Number of support vectors/core vectors comparison. OM indicates out of memory caused by too large size of data input. | 65 |
| Table 6.1 | NOTATION | 81 |
| Table 6.2 | Summarisation of data sets characteristics | 87 |
| Table 6.3 | Classification accuracy(acc) and number of support vectors(SVs)/core vectors(CVs) comparison on eight selected data sets | 88 |
| Table 7.1 | Token Frequency Table: Tokens and their frequencies for highlighted string in Figure 7.2 | 101 |
| Table 8.1 | Statistics of RCV1-V2 dataset | 112 |
| Table 8.2 | AUC, F-Measure and Macro F-Measure comparison | 113 |
| Table 8.3 | Compare the classification accuracy with (Bernhard 1999) who is the winner of KDD’99 cup | 116 |
| Table 9.1 | Classification results of the membership authentication | 125 |
| Table 10.1 | The string datasets used in training and testing the proposed algorithm | 129 |
| Table 10.2 | Data distribution-reuters-21578 dataset | 131 |
| Table 10.3 | Data distribution-application detection dataset | 131 |
| Table 10.4 | Data distribution e-Newsgroup dataset | 132 |
| Table 10.5 | Experimental results for Edit-Distance MEB optimisation: (the top 10 predicted parameter combinations using the proposed algorithm on each string dataset) | 133 |
| Table 10.6 | Experimental results for Bag-of-Words MEB optimisation: (the top 10 predicted parameter combinations using the proposed algorithm on each string dataset) | 135 |
| Table 10.7 | Experimental results for N-gram MEB optimisation: (the top 10 predicted parameter combinations using the proposed algorithm on each string dataset) | 136 |
| Table 10.8 | Root Mean Squared Error (RMSE) for string dernel MEB optimisation on each string dataset (for top 10 predicted parameter combinations) | 136 |
ACKNOWLEDGEMENTS

I have the pleasure to acknowledge here some of the many people who have inspired, supported, and educated me over the past three years. First and foremost, I am very grateful to my primary supervisor Prof. Nikola Kasabov who gave me the chance to start my PhD at Knowledge Engineering and Discovery Research Institute (KEDRI) which is a high-profile, well known and successful international research group.

I must especially thank the National Institution of Information and Communication Technology (NICT) who granted me a scholarship during the first two and a half years of my PhD study. The KEDRI PhD Scholarship supported me for the last but most important 6 months study.

I wish to express my sincere appreciation to Dr. Shaoning Pang, who used to be my primary supervisor, and who enlightened me at the beginning of my postgraduate study.

Many thanks to my secondary supervisor Dr. Russel Pears and to my third supervisor Dr. Tao Ban, whose enthusiasm, patience and intellectual spirits have been invaluable for my personal and professional growth.

I am very indebted to Joyce D’Mello, the manager and the soul of the KEDRI for all her kindness and the tremendous support she has offered me since I joined KEDRI.

I would also like to thank the past and present members of KEDRI at the Auckland University of Technology, who provided a vibrant and multicultural research environment. I am especially grateful to the members of NICT project group:

1. Lei Song who worked with me on visualisation systems;

2. Nuwan Amila Gunasekara who helped me on string kernels;

3. Kshitij Dhoble who developed active learning methods together with me and corrected my writings;
4. *Yiming Peng, Fan Liu, Alex Song, Chelsea Ji* and *Bassie Chen* who helped to finish the NICT project successfully.

I would also like to acknowledge a few people with whom I had many productive discussions during the course of my study: to *Dr. Raphael Hu* for the many conversations on technical and family issues; *Dr Stefan Schliebs* for his jokes which relaxed me during the breaks; *Dr Ammar Mohemmed, Linda Liang, Marin Karaivanov, Harya Widiputra, Haza Nuzly* and *Nuttapod Nuntalid* for many scientific and non-scientific discussions.

Last but not least, I am delighted to thank my *wife* Yu Yingying and my *parents* who supported me for almost 10 years since I first came to New Zealand. Without their supports and encouragement, I would not have been able to complete my study.
This thesis proposes and presents several novel methods to address some of the real world stream data modelling issues through the use of global and local modeling approaches. A set of real world stream data modelling issues such as dealing with large size and, high dimensionality data, skewed class distribution, different formats of data and visualisation problem are reviewed and their impact on various models are analysed.

The thesis has made nine major contributions to information science, that include four evolving modelling methods, three real world application systems that apply these methods and two stream data visualisation software prototypes. Four novel methods have been developed and published in the course of this study. They are: (1) Online Core Vector Machines (OCVM); (2) Hierarchical CVMs (HCVM) - a local modelling system based on hierarchical labelling data; (3) Dynamic Evolving CVMs (DE-CVM) - a kernel based dynamic evolving learning system; (4) Meta-Learning String Kernel CVM.

OCVM addresses the issue of one-pass, large size, high dimensionality stream data through a kernel-based online learning process. OCVM is proposed for large-scale classification by leveraging connections between learning and computational geometry. It imposes the constraint that only a single pass over the data is allowed. Standard support vector machines (SVM) training has $O(m^3)$ time and $O(m^2)$ space complexities, where $m$ is the training set size. It is thus computationally infeasible on very large data sets. Our proposed OCVM inherits the advantage of the Core Vector Machine (CVM) algorithm which can be used with non-linear kernels and has a time complexity that is \textit{linear} in $m$ and a space complexity that is \textit{independent} of $m$.

HCVM solves the skewed-class distribution problem for hierarchical stream data by identifying them through the sub-classes clustering process, creating child CVMs based on the hierarchical labels and applies supervised learning to update the core vectors. This puts strong emphasis on the unique problem
subspaces and allows easy to discriminate parent classes by local modelling on their child classes.

DE-CVM takes HCVM a step further by implementing an evolving clustering process. DE-CVM evolves through incremental, hybrid learning and accommodates new input stream data, including new features, new classes, etc. through local element tuning. New core vectors are created and updated while the system is operating. In contrast to HCVM, DE-CVM can work not only on hierarchical data but also on any numerical stream data.

Meta Learning String Kernel CVM is proposed to satisfy the string format stream data learning. Recently, string kernel based support vector machines have shown competitive performance in tasks such as text classification and protein homology detection. Meta Learning String Kernel CVM improves the effectiveness of traditional string kernels SVMs by learning the meta knowledge and adopting CVMs.

The novel stream learning methods outlined above have been applied to the following three real world data modelling problems:

1. Hierarchical network data intrusion detection;
2. Face Membership Authentication;
3. String data (i.e. Spam email, news and malicious software) classification.

These solutions constitute the main contribution of this research to the area of applied information science.

In addition to the above contributions, two stream data visualisation systems were developed: the network intrusion detection visualisation system (NIDVS) and the HCVM prototype system. These systems overcome the difficulty of monitoring stream data learning progress and also provide a better understanding of local modelling.

In summary, real world problems consist of many smaller problems. It was found beneficial to acknowledge the existence of these sub-problems and address them through the use of local models.

The core vectors extracted from the local models also brought about the availability of new knowledge for researchers and would allow more in-depth study of the sub-problems to be carried out in future research.
INTRODUCTION

1.1 BACKGROUND

Data mining is concerned with the process of computationally extracting hidden knowledge structures represented in models and patterns from large data repositories. It is an interdisciplinary field of study that has its roots in databases, statistics, machine learning, and data visualisation. Data mining has emerged as a direct outcome of the data explosion that resulted from the success in database and data warehousing technologies over the past two decades (Frawley, Piatetsky-Shapiro, & Matheus, 1992; Fayyad, 1998).

The conventional focus of data mining research was on mining resident data stored in large data repositories. The growth of technologies such as wireless sensor networks (Akyildiz, Su, Sankarasubramaniam, & Cayirci, 2002) have contributed to the emergence of data streams (Muthukrishnan, 2003). The distinctive characteristic of such data is that it is unbounded in terms of continuity of data generation. This form of data has been termed as data streams to express its flowing nature (Henzinger & Raghavan, 1998).

Table 1.1 shows the major differences between data stream processing and traditional data processing. The objective of this table is to clearly differentiate between traditional stored data processing and stream processing as a step towards focusing on the data mining aspects of data stream processing systems.
1.2 RESEARCH GOAL AND OBJECTIVES

The goal of this research is to develop novel information methods and systems for stream data modelling and specifically for network intrusion detection and string classification applications. In order to achieve the goal, this study will investigate this new and promising area, and build an on-line modelling environment using evolving systems for stream data classification.

1.2.1 Specific Research objectives

More specifically, the research includes the following objectives:

1. To critically analyse the problems related to stream data mining. Although plenty of computational intelligent models have been developed so far for stream data learning, there are few effective and integrated systems. There are a variety of issues that have not been resolved yet.

2. To develop a new on-line kernel based learning method with low computational cost and to analyse its performance under different scenarios.

3. To develop evolving clustering methods based on a local modelling framework. Local modelling breaks down the entire problem into many smaller

---

<table>
<thead>
<tr>
<th>Stream Processing</th>
<th>Traditional Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-time processing</td>
<td>Offline processing</td>
</tr>
<tr>
<td>Rapid data generation relative to the available computational resources</td>
<td>Normal or slow data generation relative to the available computational resources</td>
</tr>
<tr>
<td>Storage of data is not feasible</td>
<td>Storage of data is feasible</td>
</tr>
<tr>
<td>Approximate results are acceptable</td>
<td>Accurate results are required</td>
</tr>
<tr>
<td>Processing of samples of data is the usual task</td>
<td>Processing of every data item/record is the usual task</td>
</tr>
<tr>
<td>Linear and sublinear computational techniques are widely used</td>
<td>Techniques with high space and time complexity are used if necessary</td>
</tr>
</tbody>
</table>

Table 1.1: Stream processing vs Traditional processing
sub-problems, based on its position in the problem space. Thus, evolving clustering is a fundamental step to the creation of a local modelling system.

4. To develop a novel string classification method for string format stream data mining. One major task for stream data mining is to categorise texture. This study aims to develop a kernel based classification method for texture categorisation and to investigate its performance over network string data.

5. To develop dynamic visualisation systems for monitoring stream data learning. These will present the stream data mining progress and local modelling evaluation.

In summary, the ultimate objective of this research is to develop new methods and systems for evolving stream data mining that leads to improved classification performance and various capabilities. Such methods and systems will integrate novel machine learning and modelling techniques for:

- on-line adaption;
- fast learning;
- evolving clustering;
- classification;
- knowledge discovery and model validation.

1.3 THESIS STRUCTURE

The structure of the thesis follows the research objectives presented in the previous section and is outlined below.

CHAPTER 2 reviews current developments in the area of stream data mining by providing background information, and describing existing modelling techniques, along with learning algorithms and applications.
CHAPTER 3 presents an overview of a range of computational intelligent techniques that are relevant to this research. A brief description of batch and on-line learning techniques that have been used for global, local and personalised modelling is provided.

CHAPTER 4 proposes an on-line, novel kernel based learning algorithm on-line core vector machine (OCVM), for which computational complexity is independent from the size of training samples. An experimental comparison between the proposed and traditional support vector machine is undertaken.

CHAPTER 5 proposes a novel learning method hierarchical core vector machine (HCVM) for local modelling of hierarchical structure stream data.

CHAPTER 6 proposes a novel kernel based method dynamic evolving core vector machine (DE-CVM), which improves the original HCVM algorithm by adopting evolving clustering processing that evolves through local element tuning.

CHAPTER 7 proposes a novel string classification method, Meta Learning String Kernels CVM, that is capable of string format stream data learning.

CHAPTER 8 demonstrates the inherent suitability of using HCVM on hierarchical structure stream data (i.e. network data flow). The experimental results validate the effectiveness of HCVM by comparing with the winner of KDD’99 cup (KDD99, 1999).

CHAPTER 9 presents a case study on face membership authentication with DE-CVM. The results attest that local modelling is able to capture more valuable knowledge from sub-spaces than global modelling does.

CHAPTER 10 presents experiments with Meta Learning String Kernel CVM for string classification. The results demonstrate the capability of Meta Learning String Kernel CVM for string format stream data learning.
CHAPTER 11 conclusions are drawn and future directions for research are given.

APPENDIX A AND B describes two dynamic visualisation systems developed for monitoring stream data learning progress.
1.4 CONTRIBUTIONS

During my PhD study, I have published or delivered 3 conference paper, 3 technical reports, 1 journal article, 4 novel methods and 2 prototype systems.

1.4.1 Publications

The material presented in this thesis was partially published in a number of peer-reviewed international conference and journal articles:


1.4.2 Technical Reports

3 technical reports have been completed and delivered to National Institution of Information and Communication Technology (NICT), Japan:


1.4.3 Novel Methods

Total 4 novel methods are presented in this thesis:

• On-line Core Vector Machine (OCVM) (Chapter 4)

• Hierarchical Core Vector Machine (HCVM) (Chapter 5)

• Dynamic Evolving Core Vector Machine (DE-CVM) (Chapter 6)

• Meta Learning for String Categorization (Chapter 7)

1.4.4 Prototype Systems

We developed 2 prototype systems:

• Network Intrusion Detection Visualization System (Appendix A)

• HCVM Prototype System (Appendix B)
This chapter reviews the theoretical foundations of data stream analysis. Techniques for mining stream data are critically reviewed. Finally, research problems in the streaming mining field of study are outlined and discussed. These research issues should be addressed in order to realize robust systems that are capable of fulfilling the needs of data stream mining applications.

Intelligent data analysis has passed through a number of stages. Each stage addresses new research issues that have arisen. Statistical exploratory data analysis represents the first stage. The goal was to explore the available data in order to test a specific hypothesis. With the advances in computing power, the field of machine learning has emerged. The objective was to find computationally efficient solutions to data analysis problems. Along with the progress in machine learning research, new data analysis problems have been addressed. Due to the increase in database sizes, new algorithms have been proposed to deal with the scalability issue. Moreover, machine learning and statistical analysis techniques have been adopted and modified in order to address the problem of very large databases. Data mining is an interdisciplinary field of study and is used to extract models and patterns from large amounts of information stored in data repositories (Hand, 1999; Hand, Mannila, & Smyth, 2001; Himberg, Tikanmaki, Toivonen, Korpioaho, & Mannila, 2001).

Advances in networking and parallel computation have lead to the introduction of distributed and parallel data mining. The goal is to extract knowledge from different subsets of a dataset and integrate these generated knowledge structures in order to build a global model of the whole dataset. Client/server, mobile agent based and hybrid models have been proposed to address the com-
munication overhead issue. Different variations of algorithms have been developed in order to increase the accuracy of the generated global model. More details about distributed data mining could be found in (Park & Kargupta, 2002). Recently, the data generation rates in some data sources has become faster than ever before. This rapid generation of continuous streams of information has challenged the storage, computation and communication capabilities of computing systems. Systems, models and techniques have been proposed and developed over the past few years to address these challenges (Babcock, Babu, Datar, Motwani, & Widom, 2002).

The chapter is organised as follows. Section 2.1 presents the theoretical background of data stream analysis; mining data stream techniques and systems are reviewed in sections 2.2 and 2.3 respectively; finally section 2.4 summarizes this review chapter.

2.1 EXISTING TECHNIQUES

Research problems and challenges that have arisen in the area of mining data streams have been addressed by using well established statistical and computational approaches. We can categorise these solutions into data-based and task-based ones. In data-based solutions, the idea is to examine only a subset of the whole dataset or to transform the data either vertically or horizontally to an approximate, smaller size data representation. On the other hand, in task-based solutions, techniques from computational theory have been adopted to achieve time and space efficient solutions. In this section we review these theoretical foundations.

2.1.1 Data-based Techniques

Data-based techniques refer to summarising the whole dataset or choosing a subset of the incoming stream to be analysed. Sampling, load shedding and sketching techniques represent the former one, while synopsis data structures and aggregation represent the latter one. Here is an outline of the basics of
these techniques with pointers to their applications in the context of data stream analysis.

**Sampling**

Sampling refers to the process of probabilistic choice of a data item to be processed or not. Sampling is an old statistical technique that has been used for a long time. Boundaries of the error rate of the computation are given as a function of the sampling rate. As a fast Machine Learning technique, Hoeffding bound has been used to measure the sample size according to some derived loss functions (Domingos & Hulten, 2001).

The problem with using sampling in the context of data stream analysis is the unknown dataset size. Thus the treatment of data streams should follow a special type of analysis to find the error bounds. Another problem with sampling is that it would be important to check for anomalies for surveillance analysis as an application in mining data streams. Sampling may not be the right choice for such an application. Sampling also does not address the problem of fluctuating data rates. It would be worth investigating the relationship among the three parameters: data rate, sampling rate and error bounds.

**Load Shedding**

Load shedding refers to the process of dropping a sequence of data streams (Mayur, Babcock, Datar, & Motwani, 2003). Load shedding has been used successfully in querying data streams. It has the same problems as sampling. Load shedding is difficult to be used with mining algorithms because it drops chunks of data streams that could be used in the structuring of the generated models or it might represent a pattern of interest in time series analysis.

**Sketching**

Babcock et al. (2002) describe the process of randomly projecting a subset of features. It is the process of vertically sampling the incoming stream. Sketching has been applied in comparing different data streams and in aggregate queries. The major drawback of sketching is that it has low accuracy. It is
2.1 EXISTING TECHNIQUES

Principal Component Analysis (PCA) would be a better solution in streaming applications.

Synopsis Data Structures

Creating synopsis of data refers to the process of applying summarisation techniques that are capable of summarising the incoming stream for further analysis. Wavelet analysis, histograms, quantiles and frequency moments have been proposed as synopsis data structures. Since synopsis of data does not represent all the characteristics of the dataset, approximate answers are produced when using such data structures (Gilbert, Kotidis, Muthukrishnan, & Strauss, 2003).

Aggregation

Aggregation is the process of computing statistical measures such as means and variance that summarise the incoming stream. This aggregated data could be used by the mining algorithm. The problem with aggregation is that it does not perform well with highly fluctuating data distributions. Merging online aggregation with offline mining has been studied by Aggarwal, Han, Wang, and Yu (2003, 2004).

2.1.2 Task-based Techniques

Task-based techniques are methods that modify existing techniques or invent new ones in order to address the computational challenges of data stream processing. Approximation algorithms, sliding window and algorithm output granularity represent this category. The following subsections, examine each of these techniques and its application in the context of data stream analysis.

Approximation algorithms

Approximation algorithms have their roots in algorithm design (Muthukrishnan, 2003). It is concerned with design algorithms for computationally hard problems. These algorithms can result in an approximate solution with error bounds. The idea is that mining algorithms are considered hard computational problems given their features of continuance and speed and the generating environ-
ment that is featured by being resource constrained. Approximation algorithms have attracted researchers as a direct solution to data stream mining problems. However, the problem of data rates with regard with the available resources could not be solved using approximation algorithms. Other tools should be used along with these algorithms in order to adapt to the available resources. Approximation algorithms have been used by Cormode and Muthukrishnan (2003).

**Sliding Window**

The inspiration behind the technique of sliding window is that the user is more concerned with the analysis of most recent data streams. Thus a detailed analysis is done over the most recent data items and summarised versions of the old ones. This idea has been adopted in many techniques in the comprehensive data stream mining system MAIDS project (Dong et al., 2003).

**Algorithm Output Granularity**

The algorithm output granularity technique (AOG) (Bandyopadhyay, Holder, & Maulik, 2006) introduces the first resource-aware data analysis approach that can cope with fluctuating very high data rates according to the available memory and the processing speed represented in time constraints. The AOG performs the local data analysis on a resource constrained device that generates or receives streams of information. AOG has three main stages. Mining followed by adaptation to resources and data stream rates represent the first two stages. Merging the generated knowledge structures when running out of memory represents the last stage. AOG has been used in clustering, classification and frequency counting (Bandyopadhyay et al., 2006). Having discussed the different existing approaches to data stream analysis problems, the following section is devoted to stream mining techniques that use the above approaches in different ways.
2.1.3 Mining Techniques

Mining data streams has attracted the attention of the data mining community for the last three years. A number of algorithms have been proposed for extracting knowledge from streaming information. This section reviews clustering, classification, frequency counting and time series analysis techniques.

Clustering

Guha, Mishra, Motwani, and O’Callaghan (2000) have studied analytically clustering data streams using the k-median technique. The proposed algorithm makes a single pass over the data stream and uses small space. It requires $O(nk)$ time and $O(n\epsilon)$ space where $k$ is the number of centres, $n$ is the number of points and $\epsilon < 1$. They have proved that any k-median algorithm that achieves a constant factor approximation can not achieve a better run time than $O(nk)$. The algorithm starts by clustering a calculated size sample into $2k$, and then, at a second level, the algorithm clusters the above points for a number of samples into $2k$. This process is repeated until it clusters the $2k$ clusters into $k$ clusters.

Babcock, Datar, Motwani, and O’Callaghan (2003) have used an exponential histogram (EH) data structure to improve the algorithm proposed by Guha et al. (2000). They use the same method described above, however they address the problem of merging clusters when the two sets of cluster centres to be merged are far apart by maintaining the EH data structure. They have studied their proposed algorithm analytically.

Charikar, O’Callaghan, and Panigrahy (2003) have proposed another k-median algorithm that overcomes the problem of increasing approximation factors in the algorithm (Guha et al., 2000) with the increase in the number of levels used to result in the final solution of the divide and conquer algorithm. The algorithm has also been studied analytically.

Domingos and Hulten (2000) have proposed a general method for scaling up machine learning algorithms. They have termed this approach Very Fast Machine Learning VFML. This method depends on determining an upper bound for the learners loss as a function in a number of data items to be examined in
2.1 EXISTING TECHNIQUES

Each step of the algorithm. They have applied this method to K-means clustering VFKM and decision tree classification VFDT techniques. These algorithms have been implemented and evaluated using synthetic data sets as well as real web data streams. VFKM uses the Hoeffding bound to determine the number of examples needed in each step of k-means algorithm. The VFKM runs as a sequence of k-means executions where each run uses more examples than the previous one until a calculated statistical bound (Hoeffding bound) is satisfied.

Ordonez (2003) has proposed several improvements to k-means algorithm to cluster binary data streams. He has developed an incremental k-means algorithm. The experiments were conducted on real data sets as well as synthetic ones. He has demonstrated experimentally that the proposed algorithm outperforms the scalable k-means in the majority of cases. The proposed algorithm is a one pass algorithm in $O(Tkn)$ complexity, where $T$ is the average transaction size, $n$ is number of transactions and $k$ is number of centres. The use of binary data simplifies the manipulation of categorical data and eliminates the need for data normalisation. The main idea behind the proposed algorithm is that it updates the cluster centres and weights after examining a batch of transactions which equalizes the square root of the number of transactions rather than updating them one by one.

O’Callaghan, Mishra, Meyerson, Guha, and Motwani (2002) have proposed stream and local search algorithms for high quality data stream clustering. The stream algorithm starts by determining the size of the sample and then applies the local search algorithm if the sample size is larger than a pre-specified equation result. This process is repeated for each data chunk. Finally, the local search algorithm is applied to the cluster centers generated in the previous iterations.

Aggarwal, Han, Wang, and Yu (2003) have proposed a framework for clustering data steam called CluStream algorithm. The proposed technique divides the clustering process into two components. The online component stores summarised statistics about the data streams and the offline one performs clustering on the summarised data according to a number of user preferences such as the time frame and the number of clusters. A number of experiments on real datasets have been conducted to prove the accuracy and efficiency of
the proposed algorithm. Aggarwal et al. (2004) have recently proposed HPStream; a projected clustering for high dimensional data streams. HPStream has outperformed CluStream in recent results.

Keogh and Lin (2005) have shown empirically, that most highly cited clustering of time series data streams algorithms proposed so far in the literature come out with meaningless results in subsequence clustering. They have proposed a solution approach using k-motif to choose the sub-sequences that the algorithm can work on to produce meaningful results.

Bandyopadhyay, Holder, and Maulik (2006) have developed Lightweight Clustering LWC. It is an AOG-based algorithm. AOG has been discussed in section 2.1. The algorithm adjusts a threshold that represents the minimum distance measure between data items in different clusters. This adjustment is done regularly according to a pre-specified time frame. It is done according to the available resources by monitoring the input-output rate. This process is followed by merging clusters when the memory is full.

2.1.4 Classification

H. Wang, Fan, Yu, and Han (2003) have proposed a general framework for mining drifting data streams. They have observed that data stream mining algorithms proposed so far have not addressed the concept of drifting in the evolving data. The proposed technique uses weighted classifier ensembles to mine data streams. The expiration of old data in their model depends on the data distribution. They use synthetic and real life data streams to test their algorithm and compare between the single classifier and classifier ensembles. The proposed algorithm combines multiple classifiers weighted by their expected prediction accuracy. Also the selection of number of classifiers instead of using all is an option in the proposed framework without loosing accuracy in the classification process.

Ganti, Gehrke, and Ramakrishnan (2002) have developed analytically an algorithm for model maintenance using insertion and deletion of blocks of data records. This algorithm can be applied to any incremental data mining model. They have also described a generic framework for change detection between
two data sets in terms of the data mining results they induce. They formalise the above two techniques into two general algorithms: GEMM and FOCUS. The algorithms have been applied to decision tree models and the frequent itemset model. The GEMM algorithm accepts a class of models and an incremental model maintenance algorithm for the unrestricted window option, and outputs a model maintenance algorithm for both window-independent and window-dependent block selection sequence. The FOCUS framework uses the difference between data mining models as the deviation in data sets.

Domingos and Hulten (2000) have developed VFDT. It is a decision tree learning systems based on Hoeffding trees. It splits the tree using the current best attribute taking into consideration that the number of examined data items used satisfies a statistical measure which is Hoeffding bound. The algorithm also deactivates the least promising leaves and drops the non-potential attributes.

Papadimitriou, Brockwell, and Faloutsos (2003) have proposed Arbitrary Window Stream mOdeling Method (AWSOM) for interesting pattern discovery from sensors. They developed a one-pass algorithm to incrementally update the patterns. Their method requires only $O(\log N)$ memory where $N$ is the length of the sequence. They conducted experiments with real and synthetic data sets. They use wavelet coefficients as compact information representation and correlation structure detection, and then apply a linear regression model in the wavelet domain.

Aggarwal et al. have adopted the idea of microclusters introduced in Clus-stream in On-Demand classification (Aggarwal et al., 2004) and it shows a high accuracy. The technique uses clustering results to classify data using statistics of class distribution in each cluster.

Last (2002) has proposed an online classification system that can adapt to concept drift. The system rebuilds the classification model with the most recent examples. Using the error rate as a guide to concept drift, the frequency of model building and the window size are adjusted. The system uses info-fuzzy techniques for model building and information theory to calculate the window size.
Ding, Ding, and Perrizo (2002) have developed a decision tree based on Peano count tree data structure. It has been shown experimentally that it is a fast building algorithm that is suitable for streaming applications.

Bandyopadhyay et al. (2006) have developed Lightweight Classification LW-Class. It is a variation of LWC. It is also an AOG-based technique. The idea is to use K nearest neighbours with updating the frequency of class occurrence given the data stream features. In case of contradiction between the incoming stream and the stored summary of the cases, the frequency is reduced. In the case when the frequency is equal to zero, all the cases represented by this class are released from the memory.

2.1.5 Frequency Counting

Giannella, Yang, Zhang, Yan, and Yu (2008) have developed a frequent item sets mining algorithm over data stream. They have proposed the use of tilted windows to calculate the frequent patterns for the most recent transactions based on the fact that users are more interested in the most recent transactions. They use an incremental algorithm to maintain the FP-stream which is a tree data structure to represent the frequent itemsets. They conducted a number of experiments to prove the algorithm efficiency.

Manku and Motwani (2002) have proposed and implemented an approximate frequency count in data streams. The implemented algorithm uses all the previous historical data to calculate the frequent patterns incrementally.

Cormode and Muthukrishnan (2003) have developed an algorithm for counting frequent items. The algorithm uses group testing to find the hottest k items. The algorithm is used with the turnstile data stream model which allows addition as well as deletion of data items. An approximation randomized algorithm has been used to approximately count the most frequent items. It is worth mentioning that this data stream model is the hardest to analyse. Time series and cash register models are computationally easier. The former does not allow increments and decrements and the latter one allows only increments.

Bandyopadhyay et al. (2006) have developed one more AOG-based algorithm: Lightweight frequency counting (LWF). It has the ability to find an
approximate solution to the most frequent items in the incoming stream using adaptation and releasing the least frequent items regularly in order to count the more frequent ones.

2.1.6 Time Series Analysis

Indyk, Koudas, and Muthukrishnan (2000) have proposed approximate solutions with probabilistic error bounding to two problems in time series analysis: relaxed periods and average trends. The algorithms use dimensionality reduction sketching techniques. The process starts with computing the sketches over an arbitrarily chosen time window and creating the so called sketch pool. Using this pool of sketches, relaxed periods and average trends are computed. The algorithms have shown experimentally efficiency in running time and accuracy.

Perlman and Java (2002) have proposed a two phase approach to mine astronomical time series streams. The first phase clusters sliding window patterns of each time series. Using the created clusters, an association rule discovery technique is used to create affinity analysis results among the created clusters of time series.

Zhu and Shasha (2002) have proposed techniques to compute some statistical measures over time series data streams. The proposed techniques use the discrete Fourier transform. The system is called StatStream and is able to compute approximate error bounded correlations and inner products. The system works over an arbitrarily chosen sliding window.

Lin, Keogh, Lonardi, and Chiu (2003) have proposed the use of symbolic representation of time series data streams. This representation allows dimensionality/numerosity reduction. They have demonstrated the applicability of the proposed representation by applying it to clustering, classification, indexing and anomaly detection. The approach has two main stages. The first one is the transformation of time series data to Piecewise Aggregate Approximation followed by transforming the output to discrete string symbols in the second stage.
Y. Chen, Dong, Han, Wah, and Wang (2002) have proposed the application of so-called regression cubes for data streams. Due to the success of OLAP technology in the application of static stored data, it has been proposed to use multidimensional regression analysis to create a compact cube that could be used for answering aggregate queries over the incoming streams. This research has been extended to be adopted in an ongoing project Mining Alarming Incidents in Data Streams (MAIDS).

Himberg et al. (2001) have presented and analysed randomised variations of segmenting time series data streams generated by on-board mobile phone sensors. It has been proven in this study that Global Iterative Replacement provides approximately an optimal solution with high efficiency in running time.

Guralnik and Srivastava (1999) have developed a generic event detection approach of time series streams. They have developed techniques for batch and online incremental processing of time series data. The techniques have proven efficiency with real and synthetic data sets.

2.2 Systems

Several applications have stimulated the development of robust streaming analysis systems. They are discussed below.

Burl, Fowlkes, Roden, Stechert, and Mukhtar (1999) have developed Diamond Eye for NASA and JPL. The aim of this project was to enable remote computing systems as well as observing scientists to extract patterns from spatial objects in real-time image streams. The success of this project enables “a new era of exploration using highly autonomous spacecraft, rovers, and sensors?” (Burl et al., 1999). This project represents an early development in streaming analysis applications.

Kargupta et al. (2002) have developed the first ubiquitous data stream mining system: MobiMine. It is a client/server PDA-based distributed data stream mining application for stock market data. It should be pointed out that the mining component is located at the server side rather than the PDA. There are different interactions between the server and PDA until the results are finally
displayed on the PDA screen. The tendency to perform data mining at the server side has been changed with the increase of the computational power of small devices.

Tanner, Alshayeb, Criswell, Iyer, and Mcdowell (2008) have developed Environmen for On-Board Processing (EVE). The system mines data streams continuously generated from measurements of different on-board sensors in astronomical applications. Only interesting patterns are transferred to the ground stations for further analysis preserving the limited bandwidth. This system represents the typical case for astronomical applications. Huge amounts of data are generated and there is a need to analyse this streaming information in real time.

Srivastava (2003) have developed a NASA project for onboard detection of geophysical processes represented in snow, ice and clouds using kernel clustering methods. These techniques are used for data compression. The motivation of the project is to preserve the limited bandwidth needed to send image streams to the ground centers. The kernel methods have been chosen due to their low computational complexity in such resource-constrained environments.

### 2.3 Research Issues

Data stream mining is a stimulating field of study that has raised challenges and research issues to be addressed by the database and data mining communities. The following is a discussion of both addressed and open research issues (Dong et al., 2003; Bandyopadhyay et al., 2006; Golab & Özsu, 2003; Kargupta et al., 2002). The following is a brief discussion of previously addressed issues.

- Unbounded memory requirements due to the continuous flow of data streams: machine learning techniques represent the main source of data mining algorithms. Most of machine learning methods require data to be resident in memory while executing the analysis algorithm. Due to the huge amounts of data in the generated streams, it is absolutely crucial to
deign space efficient techniques that can have only one-pass learning of the incoming stream. (Chapter 4)

- Required result accuracy: designing space and time efficient techniques should be accompanied with acceptable result accuracy. Approximation algorithms as mentioned earlier can guarantee error bounds. Also sampling techniques adopt the same concept as it has been used in VFML (Domingos & Hulten, 2001). (Chapter 5)

- Modelling changes of mining results over time: in some cases, the user is not interested in mining data stream results, but how these results change over time. For example, if the number of clusters generated changes, it might represent some changes in the dynamics of the arriving stream. Dynamics of data streams using changes in the knowledge structures generated would benefit many temporal-based analysis applications. (Chapter 6)

- Visualisation of data mining results on small screens of mobile devices: visualisation of traditional data mining results on a desktop is still a research issue. Visualisation in small screens of a PDA for example is a real challenge. Imagine a businessman and data is being streamed and analysed on his PDA. Such results should be efficiently visualised in a way that enables him to take a quick decision. This issue has been addressed in (Kargupta et al., 2002). (Appendix A and B)

The above issues represent the grand challenges to the data mining community in this essential field. There is a real need inspired by the potential applications in astronomy and scientific laboratories as well as business applications to address the above research problems.

### 2.4 SUMMARY

The dissemination of the data stream phenomenon has necessitated the development of stream mining algorithms. The area has attracted the attention of the data mining community. The proposed techniques have their roots in statistics and theoretical computer science. Data-based and task-based techniques
are the two categories of data stream mining algorithms. Based on these two categories, a number of clustering, classification, frequency counting and time series analysis have been developed. Systems have been implemented to use these techniques in real applications. Mining data streams is still in its infancy. They are discussed in this thesis along with open issues in data stream mining.
Chapter 3

MODELING TECHNIQUES FOR COMPUTATIONAL INTELLIGENCE

This chapter gives an overview of different categories of computational intelligence models with their modelling methods in batch mode and online mode. Several research questions are formulated at the end of the chapter.

“Complex problems usually require a more complex intelligent system for their solution, consisting of several models. Some of these models can be evolving models.” Kasabov (2007b)

Kasabov (2007c) put predictive models into three different categories.

1. A Global Model is a single model that learns from the entire dataset. The developed model is then applied on future data.

2. A Local Model is a fixed mixture of models trained on the entire dataset. However, when it is applied to future data, only one or a subset of relevant models will contribute to the prediction.

3. A Personalised Model is an individualised model that is created dynamically for each prediction, using only relevant input vectors through transductive reasoning.

A graphical representation of an integrated multimodel system is depicted in Figure 3.1 For every single input vector, the outputs of the tree models are weighted. The weights can be adjusted and optimised for every new input vector in a similar way to the parameters of a personalised model (Kasabov, 2007a).

\[ y_i = w_{i,g} y_i^{(global)} + w_{i,l} y_i^{(local)} + w_{i,p} y_i^{(personalised)} \] (3.1)
The chapter is organised as follows. Section 3.1 presents the global model. Local and personalised models are reviewed in sections 3.2 and 3.3 respectively. Section 3.4 describes the differences between batch mode and on-line mode learning. Sections 3.5 and section 3.6 reviews batch and on-line mode methods. Finally section 3.7 summarises this review chapter and gives open questions.

3.1 GLOBAL MODEL

The global model is a single, fixed, reusable model, trained with the entire dataset and can be applied to future data.

Most of today’s predictive models are global (inductive) models, where the model learns from the training data and then is applied to future data. Lin-
ear Regression (Hastie, Tibshirani, & Friendman, 2001), Multi-layer Perceptron (MLP) (“Multilayer feedforward networks are universal approximators”, 1989; S. Yang, Ho, & Lee, 2006), Support Vector Machine (SVM) (Vapnik, 1998), Adaptive-Network-Based Fuzzy Inference System (ANFIS) (Jang, 1993; Jang & Sun, 1995; Jang, Sun, & Mizutani, 1997) and Echo State Network (ESN) are examples of global models. MLP and SVM machine learning algorithms were proposed many years ago and are still the two most widely used neural network models.

There are two limitations with this type of model:

First, if a new pattern emerges in the future, the existing model will not be able to handle it as the model has not been trained to recognise this pattern and a new model may need to be developed. This can be time consuming depending on the model and the complexity of the problem.

Second, as the model is developed based on all available data with the objective of minimising overall prediction error, it will be biased toward the majority of the data. A pattern without enough support will have little influence on the model.

This is similar to the issue with interpolation versus extrapolation. If the new pattern is similar to some existing pattern, then it is considered interpolation, where there is enough support of the prediction made for this pattern. However, if the new pattern is very different from any of the existing patterns, then it is considered extrapolation, where the prediction made for this new pattern is less meaningful and subject to greater uncertainty.

Recent research in the field of machine learning has focused on model ensembles that use a mixture of models to achieve better overall accuracy. Several studies have reported that an ensemble of models works better than a global model (Cevikalp & Polikar, 2008; Islam, Yao, Shahriar Nirjon, Islam, & Murase, 2008; H. C. Kim, Pang, Je, Kim, & Bang, 2002; Nguyen, Abbass, & McKay, 2008; Pang & Kasabov, 2004; Yao & Liu, 1998, 1996; Zhou & Jiang, 2003).

There are many strategies that are commonly used to create an ensemble: bagging (H. C. Kim et al., 2002), boosting (Islam et al., 2008) and clustering (Kasabov & Song, 2002) are well known strategies. Depending on the strategy used, the ensembles generally try to either generate different views of one
problem or break down the problem into smaller problems and tackle each problem independently. Sometimes both approaches are used.

3.2 LOCAL MODEL

The local model (Fontenla-Romero, Alonso-Betanzos, Castillo, Principe, & Guijarro-Berdias, 2002; Kasabov, 2001; Kasabov & Song, 2002; Lucks & Oki, 1999; Song & Kasabov, 2005; Yamada, Yamashita, Ishii, & Iwata, 2006) is a type of model ensemble that breaks down the problem into many smaller sub-problems, based on their position in the problem space. The sub-problems can be defined through a clustering process such as k-means, fuzzy c-means and hierarchical clustering that group similar input vectors based on their similarity (distance measure).

This type of model assumes that each cluster is a unique problem subspace and a sub-model should be developed for it. The quality of the cluster is, therefore, the foundation of this type of model.

The data clustering parameters often need to be adjusted, according to the sub-model’s requirements or the characteristic of the problem. Many models, such as linear regression, need the number of input vectors to be significantly greater than the number of variables and, therefore, the clusters must be large enough to support this type of sub-model. Hence, local models may require more training data than the global model to ensure that each sub-model is trained with a sufficient amount of input vectors.

In addition, the clustering process is strongly affected by the amount of noise in the data of irrelevant or redundant features, as it affects the distance measure used by most clustering methods.

3.3 PERSONALISED MODEL

Transductive reasoning (Vapnik, 1998; Kasabov & Pang, 2003; Kasabov, 2007b) was originally proposed by Vapnik in 1998 for the development of an individualised model through transductive reasoning for a given input vector without first developing a generalised model in the intermediate stage.
This approach has been widely used to solve various real life problems like text classification (“Learning with progressive transductive support vector machine”, 2003; Joachims, 1999), speech recognition (Joachims, 2003), image recognition (Li & Chua, 2003) and language translation (Ueffing, 2007).

The main difference here is that transductive reasoning focuses on finding a solution for each prediction instead of creating a generalised solution for the problem and then uses it for each prediction.

The model is created dynamically for each prediction, which utilises all available data and uses the most suitable parameters, features or model to make the prediction.

### 3.4 Batch vs Online Mode Learning

The on-line and batch modes are slightly different, although both will perform well for parabolic performance surfaces. One major difference is that the batch algorithm keeps the system weights constant while computing the error associated with each sample in the input. Since the on-line version is constantly updating its weights, its error calculation (and thus gradient estimation) uses different weights for each input sample. This means that the two algorithms visit different sets of points during adaptation. However, they both converge to the same minimum.

An online model can be: global (e.g. Incremental Support Vector Machine (SVM), Incremental Principal Component Analysis (PCA), Incremental Linear Discriminant Analysis (LDA)), local (e.g. Evolving Takagi-Sugeno (eTS+), Evolving Fuzzy Neural Networks (EFuNN), Dynamic Evolving Neural-Fuzzy Inference System (DENFIS)) or personalised (e.g. Weighted Nearest Neighbor (wkNN and wwKNN)).

A batch model can be global (e.g. SVM) or local but no adoption (e.g. SVM tree).
3.5 BATCH MODE METHODS

3.5.1 Support Vector Machines

Support vector machines (Vapnik, 1998) are a set of related supervised learning methods that are used for classification. In Support Vector machine, each instance in the training set contains one “target value” (class labels) and several “attributes” (features). The goal of SVM is to produce a model that predicts instances in the testing set given only their features.

\[
\begin{align*}
W \cdot X - b &= -1 \\
W \cdot X - b &= 1 \\
W \cdot X - b &= 0
\end{align*}
\]

Figure 3.2: An example of the linear separating hyperplanes in SVM. Note: the support vectors are encircled

SVM works on the principle that it tries to form the hyperplane between the data points given it separates these data points into two sets in higher dimension space by mapping these data points into the higher dimension using the feature vectors that are obtained using the attributes of the data. One set contains positive class labels and the other set contains negative class labels.

The training data is given as \( X = \{x_i, y_i\}, i = 1, \ldots, n, y_i \in \{-1, 1\}, x_i \in \mathbb{R}^m \), where \( x_i \) is an \( m \)-dimensional data vector, \( y_i \) is the corresponding class label.

In mapping the point from one dimension to a higher dimension in linear SVM, a dot product is used between them but there exist non-linear kernels
that use other functions to achieve this task. This is needed because at times the distribution of the data is such that we cannot find a linear separating hyperplane even in higher dimensions. The kernel functions that we explored are as follows:

1. Polynomial: \((x \cdot x' + 1)^d\);

2. Radial: \( \exp(-\gamma \|x - x'\|^2)\);

3. Sigmoid Kernel: \(\tanh(\kappa x \cdot x' + c)\).

Assume there exist some hyperplanes that separate positive (label ‘+1’) and negative (label ‘-1’) samples. The data points \(x_i\) falling on such a hyperplane should satisfy the following equation:

\[ w \cdot x_i + b = 0 \]  \hspace{1cm} (3.2)

where \(w\) is a normal vector perpendicular to the hyperplane, a parameter \(b\) specifies the perpendicular offset from the hyperplane to the origin, and \(\|w\|\) is an Euclidean normal vector of \(w\).

The shortest distances from the separating hyperplane to the closest positive and negative data points are denoted by \(d_+\) and \(d_-\), respectively. Let \(d_+\) and \(d_-\) be the “margin” of a separating hyperplane. Then, the given problem is simplified by using a SVM algorithm to find the separating hyperplane with the largest margin. If the training data are linearly separable, all the training data samples should satisfy the following constraints:

\[ x_i \cdot w + b \geq +1, \forall y_i = +1 \]  \hspace{1cm} (3.3)

\[ x_i \cdot w + b \leq -1, \forall y_i = -1 \]  \hspace{1cm} (3.4)

They can be further combined and written as:

\[ y_i(x_i \cdot w + b) - 1 \geq 0, \forall i \in \{1, 2, \ldots, n\} \]  \hspace{1cm} (3.5)

The data points satisfying the equality in Eq.3.3 will fall on the hyperplane \(H1: x_i \cdot w + b = +1\), with vector \(w\) and perpendicular distance from the origin
In the same way, the data points satisfying the equality in Eq. 3.4 will fall on the hyperplane 
$H_2 : x_i \cdot w + b = -1$, with vector $w$ and perpendicular distance from the origin 
$\frac{|1 - b|}{\|w\|}$. The margin can be calculated by 
$2/\|w\|$, as $d_+ = d_- = 1/\|w\|$. Thus two parallel hyperplanes $H_1$ and $H_2$ are 
constructed, and there are no data points lying between them. Consequently, 
the pair of hyperplanes giving the maximum margin through minimising $\|w\|^2$ 
will be found and subjected to Eq. 3.5. Finally, an optimal separation can be 
achieved by the hyperplane that has the greatest distance to the neighbouring 
data points of both classes, as is illustrated in Figure 3.2. The data points are 
referred as support vectors if they satisfy the equality in Eq. 3.3 or 3.4 and their 
removal would change the solution to the discovered hyperplane. In Figure 
3.2, support vectors are indicated by extra circles. Generally, the larger the 
margin, the lower the generalisation error of the classifier (Burges, 1998a).

For non-linear classification problems, a kernel function is introduced into 
SVM to find the maximum-margin hyperplane (Boser, Guyon, & Vapnik, 1992). 
The SVM based classifiers can be mathematically formulated by:

$$
g(x) = \text{sign} \left[ \sum_{i=1}^{n} a_i y_i \Phi(x, x_i) + b \right]$$

(3.6)

where $a_i$ is a positive real constant and $b$ is a real constant, $\Phi$ is a mapping 
function used for SVM kernel function construction (Suykens & Vandewalle, 
1999), which typically has the choices from linear, polynomial and radial ba-
sis function (RBF) function. The solution to a nonlinear optimisation prob-
lem with inequality constraints is given by the saddle point of the Lagrangian, 
which is computed by:

$$
\max_{\alpha_i, \nu_i} \min_{w, b, \xi_i} \mathcal{L}(w, b, \xi_i; \alpha_i, \nu_i)
$$

(3.7)

where $\mathcal{L}$ is the Lagrangian constructed by:

$$
\mathcal{L}(w, b, \xi_i; a_i, \nu_i) = J(w, \xi_i) - \sum_{i=1}^{n} a_i \{ y_i [w^T \varphi(x_i) + b] - 1 + \xi_i \} - \sum_{i=1}^{n} \nu_i \xi_i
$$

(3.8)
where \( a_i \geq 0, b_i \geq 0 (i = 1, \cdots, n) \) are Lagrange multipliers, \( J \) is the risk bound minimised by:

\[
\min_{w, \xi_i} J(w, \xi_i) = \frac{1}{2} w^T w + c \sum_{i=1}^{n} \xi_i
\]  

(3.9)

where the parameter \( \xi_i \) is introduced by:

\[
y_i[w^T \varphi(x_i) + b] \leq 1 - \xi_i, \quad i = 1, \cdots, n \quad \xi_i \leq 0
\]  

(3.10)

Although SVM has been extensively used for solving real world problems in different research areas, there are some issues that we have to consider if we would like to have a successful implementation. One main limitation of SVM methods lies in the choice of kernel for solving real world problems, which remains an open research question in computer science and engineering. Another concern of SVM implementation for real world problems is speed and size, especially during training stage. This issue might make the learning process for a very large dataset (a large number of support vectors) particularly difficult (Burges, 1998a). Additionally, SVM is difficult to adapt to new data and the knowledge discovered by it is very limited (Kasabov, 2007c).

3.5.2 SVM tree

The SVM tree is constructed by a divide-and-conquer approach using a binary class-specific clustering and SVM classification technique; see, for example, Figure 3.3 (Pang & Kasabov, 2004; Pang, Havukkala, & Kasabov, 2006).

Basically, we perform two procedures at each node in the above tree generation. First, the class-specific clustering performs a rough classification because it splits the data into two disjoint subsets based on the global features. Next, the SVM classifier performs a ‘fine’ classification based on training supported by the previous separation result.

Figure 3.3 is an example of the SVM tree which is derived from the above SVM tree construction. As mentioned, the SVM test starts at the root node 1. If the test \( T1(x) = +1 \) is observed, the test \( T2(x) \) is performed. If the condition
3.5 BATCH MODE METHODS

$T_1(x) = +1$ and $T_2(x) = -1$ is observed, then the input data $x$ are assigned to class $a$, and so forth.

SVM trees can evolve new nodes, new local SVM to accommodate new data from an input data stream. An example of an evolving SVM tree for CNS tumor is shown in Figure 3.4 (Pang et al., 2006). Most of class 1 patients were classified into one node of 26 patients. Class 2 patients were classified into two main nodes with 16 and 8 individuals, suggesting a potential difference in these two subsets of patients. The rest of individuals were classified into nodes of only a few patients each, so the question arises, whether these are especially hard to classify patients, misclassifications or maybe represent some other types of cancer. This exemplifies the potentially valuable additional information that the 2-SVMT can provide, compared to other algorithms.

Other cancer datasets produced similar trees, some simpler, some more complex. It appears some cancers are inherently more difficult to classify than others, and this is reflected in the complexity of the classification trees.
3.6 ON-LINE MODE METHODS

3.6.1 *Incremental Support Vector Machine*

Katagiri and Abe (2006a, 2006b) proposed an incremental training method which based on the assumption that candidates for support vectors exist near the separating hyperplane and are close to the surface of a region that includes training data of each class.

The incremental SVM method generates the minimum-volume hypersphere in the feature space that includes the training data of class $j$ ($j = 1, 2$) with radius $R_j$. Next, it defines a concentric hypersphere with radius $\rho R_j$, where $\rho (0 < \rho < 1)$ is the user-defined parameter. Next, it defines the hypercone whose vertex is at the centre of the hyperspheres and which opens in the
opposite direction of the separating hyperplane. The user-defined parameter \( \theta (-90 < \theta < 90) \) defines the angle between the separating hyperplane and the surface of the hypercone.

![Figure 3.5: Deletion of the data using the hyperspheres (Katagiri & Abe, 2006a).](image)

If the added data are in the shaded regions in Figure 3.5, they will be deleted. Figure 3.6 shows the progress of deleting such new data. If the distance \( r_j(x) \) between \( \phi(x) \) and the center of the hypersphere, \( a_j \), is smaller than \( \rho R_j \), where \( \phi(x) \) is the mapping function to the feature space:

\[
r_j(x) < \rho R_j \tag{3.11}
\]

the data is deleted. Otherwise, if the angle between \( \phi(x) - a_j \) and the separating hyperplane, \( \psi_j(x) \), is larger than \( \theta \):

\[
\psi_j(x) > \theta \tag{3.12}
\]

then \( \phi(x) \) exists inside of the hypercone and \( x \) is deleted.

But even if equation 3.11 or 3.11 is satisfied, if \( x \) satisfies

\[
y(x) D(x) \leq 1, \tag{3.13}
\]
Figure 3.6: Judging whether the data are inside of the hypercone or not (Katagiri & Abe, 2006a).

$x$ is a candidate for support vectors, where $D(x) = w^T \phi(x) + b$ is the decision function. In such a case, $x$ will not be deleted. In addition, the data will be saved that are support vectors for hyperspheres because the support vectors for hyperspheres are candidates for the support vectors for hyperspheres at the next training step.

The general procedure for incremental training is as follows:

1. Train the support vector machine using the initial data set $X_a$.

2. Add the additional data set $X_b$ to $X_a$: $X_a = X_a \cup X_b$.

3. If for $x \in X_a$, equation 3.13 is not satisfied and $x$ satisfies $r_j(x) < \rho R_j$, where $j$ is the class label for $x$ or $\psi_j(x) > \theta$, delete $x$ from $X_a$:

   $X_a = X_a - \{x\}$.

4. If for $x \in X_a$ equation 3.13 is satisfied, retrain the support vector machine.

5. Repeat (2), (3), and (4).

### 3.6.2 Incremental Principal Component Analysis

Ozawa, Pang, and Kasabov (2010) proposed a new scheme of incremental learning in which feature extraction and classifier learning are simultaneously carried out online. Incremental Principal Component Analysis (IPCA) (Hall,
Marshall, & Martin, 1998) is adopted as the feature extraction method, and k-nearest neighbour classifier with evolving clustering Method (ECM) (Kasabov & Song, 2002). The capability of this on-line learning system was verified as the classification accuracy was improved constantly even with a small set of training samples.

Assume that \( N \) training samples \( x^i \in \mathbb{R}^n (i = 1, \ldots, N) \) are initially provided to a system and eigenspace model \( \Omega = (\bar{x}, U_k, \delta_k, N) \) is obtained by applying Principal Component Analysis (PCA) to the training samples. In the eigenspace model \( \Omega \), \( \bar{x} \) is a mean vector of \( x^i \in \mathbb{R}^n (i = 1, \ldots, N) \), \( U_k \) is an \( n \times k \) matrix whose column vectors correspond to eigenvectors, and \( \delta_k = \text{diag}\{\lambda_1, \ldots, \lambda_k\} \) is a \( k \times k \) matrix whose diagonal elements are non-zero eigenvalues. Here, \( k \) is the number of eigen-axes spanning the eigenspace and the value of \( k \) is determined based on a certain criterion. After calculating \( \Omega \), the system holds the information on \( \Omega \) and all the training samples are thrown away.

Now assume that the \( (N + 1)^{th} \) training sample \( x^{N+1} = y \in \mathbb{R}^n \) is given. The addition of this new sample results in changes in the mean vector and the covariance matrix; therefore, the eigenspace model \( \Omega = (\bar{x}, U_k, \delta_k, N) \) should be updated. Let us define the new eigenspace model by \( \Omega' = (\bar{x}', U'_k, \delta'_k, N + 1) \). Note that the eigenspace dimensions might be increased from \( k \) to \( k + 1 \); thus, \( k' \) in \( \Omega' \) is either \( k \) or \( k + 1 \). Intuitively, if \( y \) includes almost all energy in the current eigenspace spanned by the eigenvector \( U'_k \), there is no need to increase its dimensions. However, if \( y \) includes a certain energy in the complementary eigenspace, the dimensional augmentation is inevitable; otherwise, crucial information on the new sample \( y \) might be lost. Regardless of the necessity in eigenspace augmentation, the eigen axes should be rotated to adapt to variation in the data distribution.

To build a classifier under a dynamic enviroment, Ozawa et al. (2010) propose a nearnest-neighbor classifier whose prototypes are evolved by the ECM (Kasabov & Song, 2002)
3.6.3 *Incremental Linear Discriminant Analysis*

Linear Discriminant Analysis (LDA) finds the linear projections of data that best separate two or more classes under the assumption that the classes have equal covariance Gaussian structure (Fukunaga, 1990). LDA is an effective and widely employed technique for dimension reduction and feature extraction. It is often beneficial to learn the LDA basis from large training sets, which may not be available initially. This motivates techniques for incrementally updating the discriminant components when more data becomes available.

As noted by Fukunaga (1990), there are equivalent variants of Fisher’s criterion to find the projection matrix $U$ to maximise class separability of the data set:

$$
\max_{\arg \frac{U^T S_B U}{U^T S_W U}} = \max_{\arg \frac{U^T S_T U}{U^T S_W U}} = \max_{\arg \frac{U^T S_B U}{U^T S_T U}},
$$

(3.14)

where

$$
S_B = \sum_{i=1}^{C} n_i (m_i - \mu)(m_i - \mu)^T
$$

(3.15)

is the between-class scatter matrix,

$$
S_W = \sum_{i=1}^{C} \sum_{x \in C_i} (x - m_i)(x - m_i)^T
$$

(3.16)

is the within-class scatter matrix,

$$
S_T = \sum_{\forall x} (x - \mu)(x - \mu)^T = S_B + S_W
$$

(3.17)

is the total scatter matrix; $C$ the total number of classes; $n_i$ the sample number of class $i$; $m_i$ is the mean of class $i$, and $\mu$ is the global mean.

T. K. Kim, Wong, Stenger, Kittler, and Cipolla (2007) use the third criterion in equation 3.14 and separately update the principal components as the minimal sufficient spanning sets of $S_B$ and $S_T$. The scatter matrix approximation with a small number of principal components (corresponding to significant
eigenvalues) allows an efficient update of the discriminant components. The $S_T$ matrix rather than $S_W$ is used to avoid losing discriminatory data during the update. If we only kept track of the significant principal components of $S_B$ and $S_W$, any discriminatory information contained in the null space of $S_W$ would be lost (note that any component in the null space maximises the LDA criterion). However, as $S_T = S_B + S_W$ and both $S_B$ and $S_W$ are positive semi-definite, vectors in the null space of $S_T$ are also in the null space of $S_B$, and are thus being ignored in the update. The two steps of the algorithm are: (1) Update the total scatter matrix $S_T$, (2) Update the between-class scatter matrix $S_B$.

Given two sets of data represented by eigenspace models

$$\{\mu_i, M_i, P_i, \Lambda_i\}_{i=1,2}$$  \hspace{1cm} (3.18)

where $\mu_i$ is the mean, $M_i$ the number of samples, $P_i$ the matrix of eigenvectors and $\Lambda_i$ the eigenvalue matrix of the i-th data set, the combined eigenspace model $\{\mu_3, M_3, P_3, \Lambda_3\}$ is computed.

Figure 3.7: Concept of sufficient spanning sets of the total scatter matrix (a), the between-class scatter matrix (b) and the projected matrix (c). The union set of the principal components $P_1$, $P_2$ or $Q_1$, $Q_2$ of the two data sets and the mean difference vector $\mu_1 - \mu_2$ can span the respective total or between-class scatter data space (a and b). The dimension for the component $m_{1i} - m_{2i}$ should not be removed (cross=incorrect) from the sufficient set of the between-class scatter data but retained in the set (circle=correct) (b). The projection and orthogonalisation of the original components $Q_{31}$. $Q_{32}$ yields the principal components of the projected data up to rotation (c). See the corresponding sections for detailed explanations.
As visualised in Figure 3.7a, the union of the two principal components and the mean difference vector can span all data points of the combined set in the three dimensional space. The principal components of the combined set are found by rotating this sufficient spanning set. Note that this use of the sufficient spanning set is only possible in the case of merging generative models where the scatter matrix of the union set is represented as the sum of the scatter matrices of the two sets explicitly as

\[ S_{T,3} = S_{T,1} + S_{T,2} + M_1 M_2 / M_3 \cdot (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T, \]  

where \( \{S_{T,i}\}_i = 1, 2 \) are the scatter matrices of the first two sets. The method can therefore not be used to directly merge the discriminant components of LDA models.

The between-class scatter matrix \( S_{B,i} \) can be written as

\[ S_{B,i} = \sum_{j=1}^{C_i} n_{ij} (m_{ij} - \mu_i)(m_{ij} - \mu_i)^T = \sum_{j=1}^{C_i} n_{ij} m_{ij} m_{ij}^T - M_i \mu_i \mu_i^T. \]  

The combined between-class scatter matrix can further be written as the original between-class scatter matrices and an auxiliary matrix \( A \) as

\[ S_{B,3} = S_{B,1} + S_{B,2} + A + M_1 M_2 / M_3 \cdot (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T, \]  

where

\[ A = \sum_{k \in s} \frac{-n_{1k} n_{2k}}{n_{1k} + n_{2k}} (m_{2k} - m_{1k})(m_{2k} - m_{1k})^T. \]  

The set \( s = \{k|k = 1, 2, \ldots, c\} \) contains the indices of the common classes of both data sets.

The incremental LDA allows highly efficient learning to adapt to new data sets. A solution closely agreeing with the batch LDA result can be obtained with far lower complexity in both time and space. The incremental LDA algorithm can also be incorporated into a classic semi-supervised learning framework and applied to many other problems in which LDA-like discriminant components are required.
3.6.4 *Evolving Takagi-Sugeno model*

Angelov and Buswell (2003) proposed the evolving Takagi-Sugeno (eTS) fuzzy system; after that, Angelov (2010) presented an enhanced version of eTS algorithm which is called eTS+. This algorithm has been tested on time-series prediction and a data stream from a real engine.

Evolving Takagi-Sugeno (TS) fuzzy systems (Angelov, 2004) can be of multi-input-multi-output (MIMO) type. As seen from Figure 3.8, eTS+ is represented as a six-layer neuro-fuzzy system. The first aim of data space partitioning can be achieved by forming clusters around focal points that have high density.

The aim of the eTS+ clustering method differs from typical clustering approaches as the clusters associated with antecedent parts of fuzzy systems significant overlap. In this case, (Angelov, 2010) adopted centre of gravity (CoG) aggregation (Yager & Filev, 1994) which weighs the average outputs of individual fuzzy rules to produce the overall output. By doing this, the fuzzy system is able to cope with uncertainties and has a cooperative nature.
Once clusters are generated, eTS+ can monitor the quality of the clusters online. Five quality elements are estimated in total: support, age, utility, zone of influence and local density.

Once the structure of eTS+ is defined and established, (Angelov, 2004) proposed weighted least square method (fwRLS) for local or global optimisation. The test results show that local optimum, usually provides a lower error rate than globally optimum.

3.6.5 Evolving Fuzzy Neural Networks (EFuNN)

Evolving Fuzzy Neural Networks (EFuNN) (Kasabov, 2002) is a connectionist model with neuro-fuzzy inference systems for implementing ECOS. EFuNNs are fuzzy neural network structures that evolve based on Evolving Connectionist Systems (ECOS) principles. Fuzzy neural networks are connectionist structures that can be interpreted by a set of fuzzy rules and a fuzzy inference system (Jang, 1993). EFuNN has a five-layer structure in which all nodes represent membership functions (MF) and can be modified during learning. Figure 3.9 illustrates an example of an EFuNN with a short term memory and feedback connections.

The input layer is the first layer that contains input variables. The second layer is a fuzzy input layer where each input variable is represented by a group of neurons. These neurons denote the fuzzy quantisation of the input variable, e.g. three neurons can be used to represent “best”, “good” and “bad” fuzzy values of a variable. Different MFs can be attached to the neurons, such as triangular or Gaussian MF. This layer aims to assign the input variables into membership degrees to which they belong to the corresponding MF. Within this layer, new neurons are created, when the corresponding variable value of a given input vector does not belong to any of the existing MFs. An optional short-term memory layer can be introduced through feedback connections from the rule node layer.

The rule (case) layer is the third layer in EFuNN which contains rule nodes that evolve through supervised or unsupervised learning. The rule nodes represent prototypes of the associations between input and output data. Each rule
node $r$ is defined by two vectors of connection weights: $\omega_1(r)$ and $\omega_2(r)$. The former is adjusted by an unsupervised learning model based on the similarity measurement within a local problem space, while the latter is adjusted by a supervised learning model based on the estimation of output error. The neurons in the fourth layer represent the fuzzy quantisation of the output variables. Finally, the fifth layer gives the value of the output variables.

3.6.6 Dynamic Evolving Neural-Fuzzy Inference System

A Dynamic Evolving Neural-Fuzzy Inference System (DENFIS) (Kasabov & Song, 2002) is a fuzzy inference system that is capable of on-line and off-line learning through on-line clustering. This paper had over 380 citations on Google Scholar at the time of writing.

DENFIS starts by clustering the data and creates a fuzzy inference system that is based on the clusters. A maximum distance-based clustering algorithm, Evolving Clustering Method, is used to cluster the input data. Once the clusters

Figure 3.9: An example of an EFuNN with a short term memory and feedback connections, adapted from Kasabov (2001)
are derived, a Takagi-Sugeno fuzzy rule is created for each cluster. These rules are then optimised through the back-propagation method. For each prediction, \( m \) most activated rules are dynamically chosen to derive the final output. New rule sets can be inserted into or extracted from the model. The DENFIS algorithm is described in Algorithm 1

\begin{algorithm}
\begin{algorithmic}
\State \textbf{Input:} Input vectors \( x_i \in X \)
\State \textbf{Output:} \( n \) clusters’ centers \( C_{c_j} \) and corresponding radius \( R_{u_j} \) where \( j = 1, 2, \ldots, n \)
\State 1: Create the initial cluster \( C_1 \) and set the position of the first training data as a cluster center \( C_{c_1} \) with the cluster radius \( R_{u_1} = 0 \).
\State 2: \textbf{while} do
\State 3: \textbf{if} \( \forall x_i \in C = \{C_1, C_2, \ldots, C_n\} \) \textbf{then}
\State 4: \hspace{1em} Terminates the algorithm.
\State 5: \textbf{else}
\State 6: \hspace{1em} Calculate the distance between the current training input vector \( x_i \) and the cluster center \( C_{c_j} \). \( D_{ij} = \|x_i - C_{c_j}\| \), \( j = 1, 2, \ldots, n \).
\State 7: \hspace{1em} \textbf{end if}
\State 8: \textbf{if} \( \exists \|x_i - C_{c_m}\| \leq R_{u_m} \) \textbf{then}
\State 9: \hspace{1em} \( x_i \) is assumed to belong to cluster \( C_m \) and no new cluster is created and no existing cluster is updated.
\State 10: \hspace{1em} Go back to step 3.
\State 11: \textbf{end if}
\State 12: \hspace{1em} Find a cluster \( C_a \) from all existing cluster centers with \( S_{ij} = D_{ij} + R_{u_j} \), \( j = 1, 2, \ldots, n \), and select the cluster center \( C_{c_a} \) with the smallest \( S_{ia} = D_{ia} + R_{u_a} = \min \{S_{ij} \}, j = 1, 2, \ldots, n \).
\State 13: \hspace{1em} \textbf{if} \( S_{ia} > 2 \times D_{thr} \) \textbf{then}
\State 14: \hspace{1em} \textbf{end if}
\State 15: \hspace{1em} \textbf{end if}
\State 16: \hspace{1em} \textbf{end if}
\State 17: \hspace{1em} \textbf{if} \( S_{ia} \leq 2 \times D_{thr} \) \textbf{then}
\State 18: \hspace{1em} The cluster \( C_a \) is updated by moving \( C_{c_a} \) and enlarging the cluster radius \( R_{u_a} = S_{ia}/2 \) and the new cluster center \( C_{c_a} \) is set as follows: \( C_{c_{a_{\text{new}}}} = x_i - (C_{c_a} - x_i) \times \frac{S_{ia}/2}{D_{ia}} \).
\State 19: \hspace{1em} \textbf{end if}
\State 20: \hspace{1em} \textbf{end while}
\end{algorithmic}
\end{algorithm}

\textbf{Algorithm 1:} Evolving Clustering Method (ECM): a fast one-pass algorithm
$Dthr$ is the distance threshold of the cluster, which defines the maximum radius of the cluster. $S_{ia}$ defines inverse the level how much the $x_i$ belongs to cluster centers $C_a$. The smaller the $S_{ia}$, the more $x_i$ belongs to cluster $C_a$. Note that $x_i$ can belong to multiple clusters as there may be overlapping of the cluster radius. Euclidean distance is used as the distance measuring method in ECM.

$\begin{align*}
Dthr \quad & \text{distance threshold of the cluster} \\
S_{ia} \quad & \text{defines the level how much } x_i \text{ belongs to cluster centers } C_a \\
\end{align*}$

Figure 3.10 shows the ECM clustering process step by step.

(a) The initial cluster is created for the first input vector $x_1$.  

Figure 3.10: Example of ECM Clustering algorithm. $x_i$: input vector (*), $C^k$: cluster center, $C^k_j$: cluster, $R^k_j$: cluster radius (Kasabov & Song, 2002)
(b) $x_2$: update cluster $C_1^0 \rightarrow C_1^1$

$x_3$: create a new cluster $C_2^0$

$x_4$: belongs to $C_1^1$, no action required.

(c) $x_5$: update cluster $C_1^1 \rightarrow C_1^2$

$x_6$: belongs $C_2^1$, no action required

$x_7$: update cluster $C_2^0 \rightarrow C_2^1$

$x_8$: create a new cluster $C_3^0$

(d) $x_9$: update cluster $C_2^2 \rightarrow C_1^3$

ECM processes input vectors in a one-input-vector-at-a-time manner, and therefore the order of the input vectors being processed affects the final output. This is evident in the way the first cluster is created for the first input vector. This design was necessary since ECM is an on-line clustering method where data is made available one input vector at a time. However, this does not prove to be a significant problem in practice as the cluster centre may be slightly different based on the order of input vectors being processed, when inspected closely by visualising the input vectors in each cluster, the input vectors were very similar, as ECM originally intended.

The consequence of the Takagi-Sugeno fuzzy rule is created and updated by a (weighted) least-square estimator. The linear function is expressed as follows:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_q x_q$$

(3.23)

The coefficient $\beta$ is obtained through the following formula

$$\beta = [b_1 \ b_2 \ \ldots \ b_q]^T$$

$$b = (A^T A)^{-1} A^T y,$$

(3.24)

or for the weighted version of the LSE

$$b = (A^T W A)^{-1} A^T W y$$

(3.25)
where

\[
A = \begin{pmatrix}
1 & x_{11} & x_{12} & \cdots & x_{1q} \\
1 & x_{21} & x_{22} & \cdots & x_{2q} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_{p1} & x_{p2} & \cdots & x_{pq}
\end{pmatrix}
\]

and

\[
y = [y_1 \ y_2 \ \cdots \ y_p]^T.
\]  
(3.26)

Assume \( W \) is a diagonal matrix:

\[
W = \begin{pmatrix}
w_1 & 0 & \cdots & 0 \\
0 & w_2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & \cdots & \cdots & w_p
\end{pmatrix}
\]  
(3.27)

equation 3.24 and 3.25 can be rewritten as

\[
Weighted - LSE = \begin{cases}
    P = (A^T A)^{-1} \\
    b = PA^T y
\end{cases}
\]  
(3.28)

\[
LSE = \begin{cases}
    P_w = (A^T W A)^{-1} \\
    b_w = P_w A^T W y
\end{cases}
\]  
(3.29)

In the DENFIS on-line mode, the weighted recursive LSE is used with the following equation:

\[
WRRLSE = \begin{cases}
    b_{k+1} = b_k + w_{k+1} P_{k+1} a_{k+1} (y_{k+1} - a_{k+1}^T b_k) \\
    P_{k+1} = \frac{1}{\lambda}(P_k - \frac{w_{k+1} P_k a_{k+1} a_{k+1}^T P_k}{\lambda + a_{k+1}^T P_k a_{k+1}})
\end{cases}
\]  
(3.30)

The forgetting factor \( \lambda \) is set between 0.8 and 1. The DENFIS on-line model learning procedure is explained below:
1. Perform ECM clustering on the initial set of data $n_0$ to obtain $M$ clusters

2. For every cluster $C_i$, find $p_i$ data points are closest to $C_i, i = 1, 2, \ldots, M$;

3. Create a fuzzy rule for each cluster. The antecedent of the fuzzy rule is the cluster center. The consequent function is created using equation 3.28 or 3.29. The distance between $p_i$ and the cluster center is used to create the weight matrix.

4. The size of $p_i$ is a model training parameter. It defines the number of data points used to derive the consequent function of the fuzzy rules.

As new input vector enters the system, new fuzzy rules may be created and some rules updated. A new fuzzy rule is created if a new cluster is found in ECM. If no new clusters are created, one or more fuzzy rules are updated by using equation 3.30.

For each input vector, the DENFIS on-line model dynamically creates a Takagi-Sugeno fuzzy inference system using $m$ activated rules. $m$ is a model training parameter that should be adjusted based on the characteristic of the problem. The rules are chosen based on the position of the input vector. Since the rules are updated constantly, two input vectors with the same values at different time points may have different inferences as the fuzzy rule may have been updated before the second input vector entered the system.

3.6.7 Weighted Nearest Neighbor: WKNN & WWKNN

In a weighted distance KNN algorithm (WKNN), the output $y_i$ is calculated not only based on the output values (e.g. class label) $y_j$, but is also dependent on the weight $w_j$ measured by the distance between the nearest neighbours and the new data sample $x_i$:

$$y_i = \frac{\sum_{j=1}^{K_i} w_j \cdot y_j}{\sum_{j=1}^{K_i} w_j}$$

(3.31)

where:
• $y_i$ is the predicted output for the new vector $x_i$;

• $y_j$ is the class label of each sample in the neighborhood of $x_i$.

• $K_i$ is the number of K nearest samples to $x_i$;

• $w_j$ is the weight value calculated based on the distance from the new input vector $x_j$ to its K nearest neighbours.

The weight $w_j$ can be calculated as follows:

$$w_j = \frac{\max(d) - (d_j - \min(d))}{\max(d)}, \quad j = 1, \cdots, K$$

(3.32)

where:

• the value of weights $w_j$ ranges from $\frac{\min(d)}{\max(d)}$ to 1;

• $d = [d_1, d_2, \cdots, d_K]$ denotes the distance vector between the new input data $d_i$ and its K nearest neighbouring samples;

• $\max(d)$ and $\min(d)$ are the maximum and minimum values for vector $d$.

The distance vector $d$ is computed as:

$$d_j = \sqrt{\sum_{l=1}^{m} (x_{i,l} - x_{j,l})^2}, \quad j = 1, \cdots, K$$

(3.33)

where $m$ is the number of variables (features) representing the new input vector $x_i$ within the problem space; $x_{i,l}$ and $x_{j,l}$ are the $l$th variable values corresponding to the data vector $x_i$ and $x_j$, respectively.

The output from a WKNN classifier for the new input vector $x_i$ is a “personalised probability” that indicates the probability of vector $x_i$ belonging to a given class. For a two-class classification problem, a WKNN classifier requires a threshold $\theta$ to determine the class label of $x_i$, i.e., if the output (personalised probability) is less than the threshold $\theta$, then $x_i$ is classified into the group with “small” class label, otherwise into the group with “big” class label.

Weighted distance and weighted variables K-nearest neighbors (WWKNN) is a personalised modeling algorithm introduced by Kasabov (2007c). The
main idea behind WWKNN algorithm is: the $K$ nearest neighbor vectors are weighted based on their distance to the new data vector $x_i$, and also the contribution of each variable is weighted according to their importance within the local area where the new vector belongs (Kasabov, 2007c). In WWKNN, the assumption is made that the different variables have different importance to classifying samples into different classes when the variables are ranked in terms of their discriminative power of class samples over the whole $m$-dimensional space. Therefore, it will be more likely that the variables have different ranking scores if the discriminative power of the same variables is measured for a sub-space (localised space) of the entire problem space. The calculation of Euclidean distance $d_j$ between a new vector $x_i$ and a neighbor $x_j$ is mathematically formulated by:

$$d_{j} = \sqrt{\sum_{l=1}^{K} c_{i,l}(x_{i,l} - x_{j,l})^2}, \quad j = 1, \ldots, K$$

(3.34)

where: $c_{i,l}$ is the coefficient weighting $x_l$ in relation with its neighborhood of $x_i$, and $K$ is the number of the nearest neighbors. The coefficient $c_{i,l}$ can be calculated by a SNR function that ranks variables across all vectors in the neighbourhood set $D_{nbr}(x_i)$:

$$c_{i,l} = \frac{\bar{x}_{l}^{class1} - \bar{x}_{l}^{class2}}{\sigma_{l}^{class1} + \sigma_{l}^{class2}}$$

(3.35)

where $\bar{x}_{l}^{classi}, i = \{1, 2\}$ and $\sigma_{l}^{classi}, i = \{1, 2\}$ are the mean value and standard deviation of the $l^{th}$ feature belonging to class $i$ across the neighborhood $D_{nbr}(x_i)$ of $x_j$, respectively.

Comparing to a conventional KNN algorithm, the contribution of WWKNN lies in the new distance measurement: all variables are weighted according to their importance as discriminating factors in the neighborhood area (personalised sub-space), which might provide more precise information for classification or prediction of the new data vector.
This chapter reviews methods and techniques that are used or are highly related to the research in this PhD study. Each of the work carried out in this PhD study are either improvements made on previous studies reviewed in this chapter or integrate existing methods in a new way to achieve better results.

As a result of the reviews presented in chapter 2 and 3, several research questions related to stream data mining can be formulated as follows:

1. Can we (successfully or accurately) train/test large amount of data at real-time? (Chapter 4, page 51)

2. Can we extract more knowledge from hierarchically labelled data that is often seen in stream datasets? (Chapter 5, page 67)

3. Can the local knowledge discovered from stream data help us in achieving better results? (Chapter 6, page 80)

4. How can we produce more accurate on string classification? (Chapter 7, page 95)

5. How to monitor the process of dynamic stream data learning? (Appendix A and B, page 143-154)

These question will be answered in the rest of the thesis.

In the next chapter, a novel method is proposed for on-line kernel based stream data learning which related to research question 1.
ON-LINE CORE VECTOR MACHINES - A NOVEL ONE-PASS, FAST KERNEL BASED LEARNING METHOD

This chapter presents a one-pass, fast kernel based learning method called On-line Core Vector Machine (OCVM). It is based on the minimum enclosing ball (MEB) of streaming data. We show that the MEB updates for the streaming case can be easily adapted to learn the CVM weight vector in a way similar to using on-line stochastic gradient updates.

The rest of chapter is structured as follows: Section 4.1 discusses the reason for using kernel methods for stream data mining. Section 4.2 reviews the batch mode MEB and CVM; Section 4.3 presents the novel OCVM method based on CVM. The evaluation of OCVM on 12 benchmark datasets is presented in Section 4.4. Section 4.5 gives the conclusion.

4.1 WHY USE ON-LINE KERNEL METHODS FOR STREAM DATA MINING (E.G. NETWORK INTRUSION DETECTION)?
- ANSWER FOR QUESTION 1

The existing machine learning methods for Network Intrusion Detection (NID) fall into two categories: unsupervised learning and supervised learning. Unsupervised learning methods, such as clustering, are useful for NIDS, as malicious activities could be clustered, and better distinguished from non-malicious activities. Some successful stories have been reported in literature, Frank and Mda-c (1994) demonstrated that clustering is an effective way to find hidden patterns in data that humans might otherwise miss; Eskin, Arnold, Prerau, Port-
noy, and Stolfo (2002) detected network connection logs outliers which represent anomalies in the network traffic using fixed-width and k-nearest neighbour; and Marin, Ragsdale, and Sirdu (2001) detected network intrusions by learning vector quantization (LVQ) and locating the Bayes Optimal boundary between classes.

Unlike unsupervised learning capturing threats through outliers detection, supervised learning detects network intrusion by using classification methods, categorising network traffics into normal and abnormal groups. Diversity of classification methods have been proposed for NID applications. Frank and Mda-c (1994) used recurrent neural networks, and some other neural networks, such as Kohonen, Hopfield for intrusion detection. Dickerson, Juslin, Koukousoula, and Dickerson (2001) classified intrusions based on the combination of various statistical metrics and fuzzy logic rules. Apart from that, it is worth noting that SVM, because of its promising classification performance, is popularly used for NIDS. W.-H. Chen, Hsu, and Shen (2005) used a single polynomial kernel SVM for classifying network intrusions, which outperformed artificial neural networks (ANNs) method on both computational time and classification accuracy. Also, Mukkamala and Sung (2003) used multiple SVM’s for identifying normal traffic against four types of malicious activity, which demonstrated the super discriminability of SVM on intrusion detection.

In spite of the current success of SVM in NID applications, it is too slow for NIDS due to the high-speed computing requirement from the Internet. Thus, we need on-line kernel based methods because on-line learning is concerned with learning data as the system operates (usually in real time) and the data might exist only for a short time (Kasabov & Song, 2002).

4.2 Core Vector Machines

Standard SVM training has $O(m^3)$ time and $O(m^2)$ space complexities, where $m$ is the training set size. It is thus computationally infeasible on very large data sets. By observing that practical SVM implementations only approximate the optimal solution by an iterative strategy, Tsang, Kwok, and Cheung (2005) scaled up kernel methods by exploiting such “approximateness”’. They found
that many kernel methods can be equivalently formulated as MEB problems in computational geometry. Then, by adopting an efficient approximate MEB algorithm, Tsang obtained provably approximately optimal solutions with the idea of core sets. The CVM algorithm can be used with non-linear kernels and has a time complexity that is linear in $m$ and a space complexity that is independent of $m$.

### 4.2.1 Minimum Enclosing Ball

Given a set of points $S = \{x_1, \ldots, x_m\}, x_i \in \mathbb{R}^d$, the minimum enclosing ball of $S$ is the smallest ball that contains all the points in $S$ (denoted by $\text{MEB}(S)$). Welzl (1991) proposed $(1 + \epsilon)$-approximation MEB which can be efficiently obtained based on those called Core Set. The core set is a subset of given dataset that contains the instances located in the outer area.

Let $B_S(c, r)$ be an exact MEB of the data set $S$ with center $c$ and radius $r$, and $B_Q(\tilde{c}, \tilde{r})$ be another exact MEB with center $\tilde{c}$ and radius $\tilde{r}$. Note that, different from $B_S$, MEB $B_Q$ is constructed on the Core Set of $S : Q, Q \subset S$. Given an $\epsilon > 0$, a ball $B_Q(\tilde{c}, (1 + \epsilon)\tilde{r})$ is a $(1 + \epsilon)$-approximation of $B_S(c, r)$, if $S \supset B_Q(\tilde{c}, (1 + \epsilon)\tilde{r})$ and $\tilde{r} \leq r$.

Formally, subset $Q$ is judged as the core set of $S$, if an expansion by a factor $(1 + \epsilon)$ of its MEB contains $S$ (i.e. $S \subset B_Q(\tilde{c}, (1 + \epsilon)\tilde{r})$). Fig. 4.1 gives an example of exact MEB, Core set MEB, and Core set MEB expansion, where the dotted-line circle identifies the exact MEB of the entire dataset $B_S$, and the inner solid-line circle gives the exact MEB of Core set $B_Q$ (denoted in square). $B_Q$ does not cover the whole data points, but its $(1 + \epsilon)$ expansion (the outside circle) does.

### 4.2.2 Kernel MEB: Core Vector Machine

By adopting the above MEB algorithm to enhance a support vector machine (SVM), Tsang developed the kernel MEB method called core vector machine.

Let $\varphi$ be the feature map using kernel $\kappa$, and provides a set of $\varphi$-mapped points $S_\varphi = \{\varphi(x_1), \ldots, \varphi(x_n)\}$, the MEB of $S_\varphi$ is such a smallest ball $B(\varphi, r^*)$
Input: Set of points $X \in \mathbb{R}^d$; parameter $\epsilon = 2^{-m}$; subset $Q_0 \subset X$

Output: A batch mode MEB learning model $\Omega$

1: for $i \leftarrow 1$ to $m$ do
2: $Q \leftarrow Q_{i-1}$
3: Compute MEB center $c \leftarrow \sum_{i=1}^{m} \alpha_i \varphi(x_i)$ where $x_i \in Q$
4: radius $r \leftarrow -\sqrt{\alpha' \text{diag}(K) - \alpha' K \alpha}$
5: includeAll $\leftarrow$ true, $p \leftarrow (1 + \epsilon) \ast r$
6: for $j \leftarrow 1$ to $|X|$ do
7: if $p < ||x_j - c||$ then
8: includeAll $\leftarrow$ false
9: $p \leftarrow ||x_j - c||$
10: $q \leftarrow x_j$
11: end if
12: end for
13: if includeAll == true then
14: Return $\Omega \leftarrow \{c, r, Q\}$
15: end if
16: $Q_i \leftarrow Q_{i-1} \cup \{q\}$
17: end for
18: Return $\Omega \leftarrow \{c, r, Q\}$

Algorithm 2: Original MEB algorithm

that encloses all data points of $S_\varphi$ and has the ball center $c^*$ and radius $r^*$ determined by,

$$(c^*, r^*) = \arg\min_{c, r} r^2 : ||c - \varphi(x_i)||^2 \leq r^2 \forall i.$$ (4.1)
By the corresponding Wolfe dual,

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i k(x_i, x_i) - \sum_{i,j=1}^{m} \alpha_i \alpha_j k(x_i, x_j), \alpha \geq 0, i = 1, \ldots, m, \sum_{i=1}^{m} \alpha_i = 1, \quad (4.2)$$

where $\alpha = [\alpha_1, \ldots, \alpha_m]$ are the Lagrange multipliers and $K_{m \times m} = [k(x_i, x_j)]$ is the kernel matrix, the MEB centre $c$ and radius $r$ can be found from the optimal $\alpha$ as (Tsang et al., 2005)

$$c = \sum_{i=1}^{m} \alpha_i \varphi(x_i), \quad r = -\sqrt{\alpha' \text{diag}(K) - \alpha' K \alpha}. \quad (4.3)$$

In a kernel MEB, a core set instance is called a core vector, and the constructed supervised learning model constructed by core vectors, is called a core vector machine.

### 4.2.3 Time and Space Complexities for CVM

Existing decomposition algorithms cannot guarantee the number of iterations and consequently the overall time complexity (Chang & Lin, 2001). This section shows how this can be obtained for CVM. In the following, we assume that a plain Quadratic Programming (QP) implementation, which takes $O(m^3)$ time and $O(m^2)$ space for $m$ patterns, is used for the QP sub-problem. The time and space complexities obtained below can be further improved if more efficient QP solvers were used. Moreover, each kernel evaluation is assumed to take constant time.

Consider first the case where probabilistic speedup is not used. As proved in (Bădoiu, Har-Peled, & Indyk, 2002), CVM converges in $2/\epsilon$ iterations at the most. In other words, the total number of iterations, and consequently the size of the final core set, are $\tau = O(1/\epsilon)$. In practice, it has often been observed that the size of the core set is much smaller than this worst-case theoretical upper bound (Kumar, Mitchell, & Yildirim, 2003). As only one core vector is added at each iteration, $|S_t| = t + 2$. Initialisation takes $O(m)$ time while distance computations take $O((t + 2)^2 + tm) = O(t^2 + tm)$ time. Finding the MEB takes $O((t + 2)^3) = O(t^3)$ time, and the other operations take constant
time. Hence, the $t^{th}$ iteration takes a total of $O(tm + t^3)$ time. The overall time for $\tau = O(1/\epsilon)$ iterations is

$$T = \sum_{t=1}^{\tau} O(tm - t^3) = O(\tau^2m + \tau^4) = O\left(\frac{m}{\epsilon^2} + \frac{1}{\epsilon^4}\right).$$

(4.4)

which is linear in $m$ for a fixed $\epsilon$.

Next, we consider its space complexity. As the $m$ training patterns may be stored outside the core memory, the $O(m)$ space required will be ignored in the following. Since only the core vectors are involved in the QP, the space complexity for the $t^{th}$ iteration is $O(|S_t|^2)$. As $\tau = O(1/\epsilon)$, the space complexity for the whole procedure is $O(1/\epsilon^2)$, which is independent of $m$ for a fixed $\epsilon$.

On the other hand, when probabilistic speedup is used, initialization only takes $O(1)$ time while distance computations take $O((t + 2)^2) = O(t^2)$ time. Time for the other operations remains the same. Hence, the $t^{th}$ iteration takes $O(t^3)$ time. As probabilistic speed-up may not find the furthest point in each iteration, $\tau$ may be greater than $2/\epsilon$ though it can still be bounded by $O(1/\epsilon^2)$ (Bădoiu et al., 2002). Hence, the whole procedure takes

$$T = \sum_{t=1}^{\tau} O(t^3) = O(\tau^4) = O\left(\frac{1}{\epsilon^8}\right).$$

(4.5)

For a fixed $\epsilon$, it is thus independent of $m$. The space complexity, which depends only on the number of iterations $\tau$, becomes $O(1/\epsilon^4)$.

When $\epsilon$ decreases, the CVM solution becomes closer to the exact optimal solution, but at the expense of higher time and space complexities. Such a tradeoff between efficiency and approximation quality is typical of all approximation schemes. Moreover, be cautioned that the $O$-notation is used for studying the asymptotic efficiency of algorithms. As we are interested in handling very large data sets, an algorithm that is asymptotically more efficient (in time and space) will be the best choice. However, on smaller problems, this may be outperformed by algorithms that are not as efficient asymptotically.
4.3 THE PROPOSED ON-LINE CORE VECTOR MACHINES

This section introduces the proposed on-line CVM method for classification tasks.

4.3.1 Motivations for using CVM for Incremental Learning

We found that the CVM updates for the streaming case can be easily adapted to learn the SVM weight vector in a way similar to using online stochastic gradient updates. D. Wang, Zhang, Zhang, and Qiao (2010) addressed this issue by proposing an online CVM classifier with adaptive minimum-enclosing-ball (MEB) adjustment. This method performs polylogarithmic computation at each example, and requires very small and constant storage. However, the computational cost of Wang’s algorithm is same as original CVM. To reduce not only the constant storage requirement but also the learning time, we proposed our on-line CVM learning algorithm that perform each updating using geometry techniques. As the updating processing, which is the most important part of incremental learning, has been changed to a much faster algorithm, our OCVM performs more efficiently. Experimental results show that, even in such restrictive settings, the algorithm can learn efficiently in just one pass and demonstrates accuracies comparable to other methods.

4.3.2 On-line Mode CVM Learning

Let $x'$ be the input point causing an update to the MEB and $B'$ be the resulting ball after the update. From figure 4.2, it is easy to verify that the new centre $c'$ lies on the line joining the old center $c$ and the new point $x'$. The radius $r'$ and the centre $c'$ of the resulting MEB can be defined by simple update equations as follows:

\begin{align}
    r' &= r + \delta, \\
    \delta &= ||c' - c||.
\end{align}
Figure 4.2: MEB expansion

Hence $2\delta$, which is the closest distance of the new point $x'$ from the old ball $B$, can be defined as:

$$2\delta = (||x' - c|| - r)$$

(4.8)

Using these, we can define a closed-form analytical update equation for the new ball $B'$:

$$c' = c + \frac{\delta}{||x' - c||}(x' - c).$$

(4.9)

Thus, the updated MEB learning model $\Omega'$ is obtained by conducting the following algorithm.

**Kernelised On-line CVM**

In order to extend the usability of OCVM from linear kernels to non-linear kernels, we modify Algorithm 3. Instead of storing the weight vector $c = \sum_{x \in Q} x$, it stores Lagrange coefficients $c \leftarrow \sum_{i=1}^{m} \alpha_i \varphi(x_i)$ as the center of MEB. The distance from an incoming data sample $x'_i$ to the centre is calculated as

$$||x'_i - c||^2 = \sum_{x_j, x_l \in Q} \alpha_j \alpha_l k(x_j, x_l) + k(x_i, x_l) - 2 \sum_{x_j \in Q} \alpha_j k(x_j, x_j) + \xi^2.$$  (4.10)
4.3 THE PROPOSED ON-LINE CORE VECTOR MACHINES

**Input:** Original CVM learning model $\Omega = \{c, r, Q\}$, incoming data matrix $X'$, MEB parameter $\epsilon$, and slack parameter $\xi$

**Output:** A batch mode MEB learning model $\Omega'$

1: Initial $c' \leftarrow c$, $r' \leftarrow r$, and $Q' \leftarrow Q$

2: for $i \leftarrow 1$ to $|X'|$ do

3: Compute distance from $x'_i$ to $c$
   \[ d = \|x'_i - c\| + \sqrt{1/\xi} \]

4: Compute MEB center $c$ and radius $r$ on subset $Q$ using equation 4.3

5: if $(1 + \epsilon) \ast r < d$ then

6: $\delta \leftarrow \frac{\|x'_i - c\| - r}{r}$

7: $c' \leftarrow c + \left(\frac{1}{2} - \frac{r}{2\|x'_i - c\|}\right)(x'_i - c)$

8: $r' \leftarrow r + \left(\frac{\|x'_i - c\| - r}{2}\right)$

9: $Q' \leftarrow Q \cup x'_i$

end if

11: end for

12: Return $\Omega' \leftarrow \{c', r', Q'\}$

**Algorithm 3:** Incremental mode On-line CVM algorithm

Once the new sample $x'_i$ updates the MEB then $Q' = Q \cup \varphi(x'_i)$, the Lagrange coefficient for the new core vector is $\alpha_i = \frac{1}{2} - \frac{r}{2\|x'_i - c\|}$.

**Contraction of On-line CVM**

During the expanding of MEB, spare area is inevitably presented because the size of the MEB depends on the most significant outlier. In order to preserve the efficiency of MEB knowledge encryption, a contraction is conducted once an outlier is found. The approach of MEB contraction is similar to the expansion approach.

Given $q \in \tilde{Q}$ where $\tilde{Q} \subseteq Q - Q_{i-1}$, and $\forall \tilde{q} \in \tilde{Q}$ where $\tilde{q} \neq q$, $\exists ||\tilde{q} - c_{i-1}|| < ||q - c_{i-1}||$. From equation 4.7 and 4.8, the relationship between $\delta$ and $\delta'$ is defined as $\delta' = 2\delta$. From figure 4.3, radius $r_{i-1}$ for the original MEB $B_{i-1}$ and radius $r_i$ for the updated MEB $B_i$ are:

\[
\begin{align*}
  r_{i-1} &= r_i + \delta + ||q - c_{i-1}|| + \delta', \\
  r_i &= ||q - c_{i-1}|| + \delta.
\end{align*}
\]
From equation 4.7 and 4.8, the relationship between $\delta$ and $\delta'$ is defined as $\delta' = 2\delta$. Thus, equation 4.11 is transformed as:

\[
\begin{align*}
    r_{i-1} &= ||q - c_{i-1}|| + \delta + \delta + ||q - c_{i-1}|| + \delta', \\
    r_{i-1} &= 2||q - c_{i-1}|| + 3\delta, \\
    \delta &= \frac{r_{i-1} - 2||q - c_{i-1}||}{3}.
\end{align*}
\] (4.13) \hspace{1cm} (4.14) \hspace{1cm} (4.15)

Once we have the value for $\delta$ we can easily calculate $r_i$ and updated MEB center $c_i$ as:

\[
\begin{align*}
    r_i &= \frac{r_{i-1} + ||q - c_{i-1}||}{3}, \\
    c_i &= c_{i-1} + \frac{\delta}{||q - c_{i-1}||}(q - c_{i-1}).
\end{align*}
\] (4.16) \hspace{1cm} (4.17)

4.4 EXPERIMENTS AND DISCUSSIONS

To evaluate the performance of on-line CVM we compared the classification accuracy and the computational cost with two conventional classification methods on six two-class and six multi-class datasets. Table 4.1 and table 4.2 summarise the characteristics of 12 datasets and five classification methods respectively.
4.4 EXPERIMENTS AND DISCUSSIONS

<table>
<thead>
<tr>
<th>Type</th>
<th>Dataset</th>
<th>No. of Features</th>
<th>No. of Classes</th>
<th>Complexity</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>two-class</td>
<td>synthetic binary</td>
<td>2</td>
<td>2</td>
<td>0.0743</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>liver-disorder</td>
<td>6</td>
<td>2</td>
<td>0.5432</td>
<td>345</td>
</tr>
<tr>
<td></td>
<td>breast-cancer</td>
<td>10</td>
<td>2</td>
<td>0.1743</td>
<td>683</td>
</tr>
<tr>
<td></td>
<td>heart</td>
<td>13</td>
<td>2</td>
<td>0.7632</td>
<td>270</td>
</tr>
<tr>
<td></td>
<td>ionosphere</td>
<td>34</td>
<td>2</td>
<td>0.5163</td>
<td>351</td>
</tr>
<tr>
<td></td>
<td>web Spam</td>
<td>57</td>
<td>2</td>
<td>5.5188</td>
<td>4601</td>
</tr>
<tr>
<td>multi-class</td>
<td>synthetic multi-class</td>
<td>2</td>
<td>3</td>
<td>0.0743</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>iris</td>
<td>4</td>
<td>3</td>
<td>0.0636</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>wine</td>
<td>13</td>
<td>3</td>
<td>0.3007</td>
<td>178</td>
</tr>
<tr>
<td></td>
<td>vehicle</td>
<td>18</td>
<td>4</td>
<td>2.7264</td>
<td>846</td>
</tr>
<tr>
<td></td>
<td>vowel</td>
<td>10</td>
<td>11</td>
<td>5.1335</td>
<td>990</td>
</tr>
<tr>
<td></td>
<td>KDD99(KDD99, 1999)</td>
<td>42</td>
<td>5</td>
<td>0.5163</td>
<td>100000</td>
</tr>
</tbody>
</table>

Table 4.1: *Summarisation of data sets characteristics*

<table>
<thead>
<tr>
<th>Notation</th>
<th>Descriptions</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN</td>
<td>k-nearest neighbor</td>
<td>$k$</td>
</tr>
<tr>
<td>Bayes</td>
<td>Naive Bayes</td>
<td>None</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi layer perception</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>SVM</td>
<td>Support vector machine</td>
<td>Kernel Method ($KM$)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kernel Parameter ($KP$)</td>
</tr>
<tr>
<td>OCVM</td>
<td>Hierarchical minimum enclosing ball</td>
<td>Kernel Method ($KM$)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kernel Parameter ($KP$)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Epsilon ($\epsilon$)</td>
</tr>
</tbody>
</table>

Table 4.2: *Summarisation of classifiers characteristics*

**Accuracy of Binary Classification**

For each two-class dataset, we conducted $K$-folds cross validation which randomly partitioned original data into $K$ subsamples. Of the $K$ subsamples, a single subsample was retained as the validation data for testing the model, and the remaining $K - 1$ subsamples are used as training data. The cross-validation process was repeated $K$ times, with each of the $K$ subsamples used exactly once as the validation data. Then we can obtain the mean value and standard deviation (std) value of the $K$ results from the folds. In our case, we set $K = 10$ as 10-fold cross-validation is commonly used.

Table 6.3 shows the classification accuracy of the two-class datasets using different classifiers. The accuracy is represented by the mean value and the std value of the 10-fold cross validation. To make sure that the results for the conventional methods are not biased by an inappropriate choice of parameters, we optimised all parameters (see table 4.4) by running cross validation.
4.4 EXPERIMENTS AND DISCUSSIONS

On the two-class datasets, MLP and SVM achieved the two highest accuracies among the conventional methods. Compared to the conventional methods, OCVM obtained higher accuracies on three out of six datasets (i.e. breast-cancer, ionosphere and web Spam). It can be found from table 4.1, these three datasets have greater number of features than the other datasets. That indicates the proposed OCVM is effective for classifying higher dimensional datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>KNN</th>
<th>Bayes</th>
<th>MLP</th>
<th>SVM</th>
<th>OCVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>liver-disorder</td>
<td>64.92% ± 6.55%</td>
<td>63.17% ± 6.31%</td>
<td>71.28% ± 7.71%</td>
<td>64.92% ± 1.06%</td>
<td>59.13% ± 4.91%</td>
</tr>
<tr>
<td>breast-cancer</td>
<td>96.63% ± 1.71%</td>
<td>95.97% ± 2.44%</td>
<td>96.65% ± 2.38%</td>
<td>97.07% ± 3.07%</td>
<td>97.07% ± 2.82%</td>
</tr>
<tr>
<td>heart</td>
<td>82.50% ± 8.20%</td>
<td>75.65% ± 8.25%</td>
<td>83.33% ± 5.25%</td>
<td>83.19% ± 6.06%</td>
<td>82.67% ± 6.33%</td>
</tr>
<tr>
<td>ionosphere</td>
<td>64.11% ± 1.09%</td>
<td>65.32% ± 1.74%</td>
<td>89.16% ± 5.35%</td>
<td>92.06% ± 4.46%</td>
<td>93.73% ± 4.57%</td>
</tr>
<tr>
<td>web Spam</td>
<td>93.32% ± 0.78%</td>
<td>84.13% ± 1.97%</td>
<td>94.62% ± 0.12%</td>
<td>94.60% ± 0.10%</td>
<td>94.64% ± 0.06%</td>
</tr>
</tbody>
</table>

Table 4.3: Classification accuracy comparison for five methods applied on the six two-class datasets

<table>
<thead>
<tr>
<th>classifier</th>
<th>parameters</th>
<th>liver-disorder</th>
<th>breast-cancer</th>
<th>heart</th>
<th>ionosphere</th>
<th>web Spam</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN</td>
<td>k</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>MLP</td>
<td>α</td>
<td>100</td>
<td>0.5</td>
<td>10</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>SVM</td>
<td>KM</td>
<td>RBF</td>
<td>RBF</td>
<td>RBF</td>
<td>RBF</td>
<td>RBF</td>
</tr>
<tr>
<td></td>
<td>KP</td>
<td>0.001</td>
<td>0.5</td>
<td>0.01</td>
<td>0.01</td>
<td>0.001</td>
</tr>
<tr>
<td>OCVM</td>
<td>KM</td>
<td>RBF</td>
<td>RBF</td>
<td>RBF</td>
<td>RBF</td>
<td>RBF</td>
</tr>
<tr>
<td></td>
<td>Kp(γ)</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>ε</td>
<td>1e-6</td>
<td>1e-2</td>
<td>1e-3</td>
<td>1e-6</td>
<td>1e-3</td>
</tr>
</tbody>
</table>

Table 4.4: Summarisation of classifiers’ parameters for six two-class datasets

**Accuracy of Multi-class Classification**

Similarly to the experiments with two-class datasets, we conduct 10-fold cross validation with optimised parameters (see table 4.6) on multi-class dataset. The classification accuracy was represented by the 10-folds accuracies mean value and std value in shown in table 4.5.
On the multi-class datasets, except Naive Bayes all selected conventional methods perform similar. Compared to the conventional methods, OCVM which showed the 3 highest accuracies, performs better specially on KDD99 which has a much larger number of samples.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>KNN</th>
<th>Bayes</th>
<th>MLP</th>
<th>SVM</th>
<th>OCVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syn m-class</td>
<td>94.53% ± 1.14%</td>
<td>88.57% ± 4.29%</td>
<td>93.86% ± 2.68%</td>
<td><strong>94.97% ± 1.99%</strong></td>
<td>90.95% ± 3.67%</td>
</tr>
<tr>
<td>iris</td>
<td>94.67% ± 6.13%</td>
<td>90.35% ± 7.11%</td>
<td>95.97% ± 4.03%</td>
<td>96.00% ± 5.62%</td>
<td><strong>96.66% ± 4.24%</strong></td>
</tr>
<tr>
<td>wine</td>
<td><strong>97.16% ± 3.00%</strong></td>
<td>71.31% ± 8.53%</td>
<td>93.91% ± 8.01%</td>
<td>74.85% ± 9.15%</td>
<td>76.40% ± 8.22%</td>
</tr>
<tr>
<td>vehicle</td>
<td>71.04% ± 3.01%</td>
<td>66.24% ± 5.57%</td>
<td><strong>77.78 ± 3.96%</strong></td>
<td>71.61% ± 3.88%</td>
<td>71.61% ± 3.88%</td>
</tr>
<tr>
<td>vowel</td>
<td>96.89% ± 1.00%</td>
<td>91.37% ± 3.25%</td>
<td>98.08% ± 1.21%</td>
<td>97.47% ± 1.73%</td>
<td><strong>98.56% ± 1.34%</strong></td>
</tr>
<tr>
<td>KDD99</td>
<td>89.32% ± 1.12%</td>
<td>75.21% ± 5.32%</td>
<td>90.53% ± 2.83%</td>
<td>90.42% ± 2.18%</td>
<td><strong>94.95% ± 1.47%</strong></td>
</tr>
</tbody>
</table>

Table 4.5: Classification accuracy comparison for five methods applied on the six multi-class datasets

<table>
<thead>
<tr>
<th>classifier</th>
<th>parameters</th>
<th>iris</th>
<th>wine</th>
<th>vehicle</th>
<th>vowel</th>
<th>KDD99</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN</td>
<td>$k$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>MLP</td>
<td>$\alpha$</td>
<td>0.001</td>
<td>100</td>
<td>0.5</td>
<td>0.2</td>
<td>5</td>
</tr>
<tr>
<td>SVM</td>
<td>$K_M$</td>
<td>RBF</td>
<td>RBF</td>
<td>RBF</td>
<td>RBF</td>
<td>RBF</td>
</tr>
<tr>
<td></td>
<td>$K_P(\gamma)$</td>
<td>0.01</td>
<td>0.001</td>
<td>0.001</td>
<td>0.5</td>
<td>0.05</td>
</tr>
<tr>
<td>OCVM</td>
<td>$K_M$</td>
<td>RBF</td>
<td>RBF</td>
<td>RBF</td>
<td>RBF</td>
<td>RBF</td>
</tr>
<tr>
<td></td>
<td>$K_P(\gamma)$</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>$\epsilon$</td>
<td>$1e-6$</td>
<td>$1e-2$</td>
<td>$1e-6$</td>
<td>$1e-3$</td>
<td>$1e-4$</td>
</tr>
</tbody>
</table>

Table 4.6: Summarisation of classifiers’ parameters for six multi-class datasets

**Computational Costs and Speed Efficiency**

We measured the computation cost by analysing the training time and number of the support vectors (for SVM and OCVM). For each dataset, we train the model based on 10% samples up to 100% step by 10%.

Results are shown in Table 4.7. As it can be seen, the CPU time cost by SVM shot up with the growth of the training size, but it is the shortest while the size of training set is smaller than 300. OCVM is much faster and produces far
Table 4.7: Comparison of training CPU time (in seconds). OM indicates out of memory caused by size of data input that was too large.

fewer support vectors (which implies faster testing) on large data sets. In particular, for web Spam data 5000 samples can be processed in less than 1 second, which is 750 times faster than SVM. As we developed the evaluation environment based on MATLAB, the largest memory is restricted. Thus, 10,000 is the maximum number of samples that can be used by SVM and training takes 200 times more CPU time than OCVM. The CPU time cost by MLP is not obviously greater than OCVM on a small training set. However, it is easy to see that only OCVM keeps consistent speed once the size of training data is over 5000.

The comparison result for number of support vectors, which is shown in Table 4.8, is similar to the CPU time comparison. For relatively small training sets, with fewer than 200 samples, the SVM produces less support vectors. However, the number of support vectors obtained using SVM hoicks.
4.5 SUMMARY

Within the streaming framework for learning, we have presented an efficient, on-line CVM learning algorithm using a streaming algorithm for the MEB problem. OCVM, incrementally learning the stream data by extending the corresponding MEB towards a new sample which located outside of the existing MEB and this sample will be saved by updating the core vectors. In addition, to preserve the efficiency of MEB knowledge encryption, a contraction is conducted once an outlier is found. The experiments on batch mark datasets show the on-line CVM obtained similar or higher classification accuracy than the selected bench learning methods by taking significantly less computational time.

In the next chapter, a novel method for hierarchical structure stream data modelling is presented.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classifier</th>
<th>SVM</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
<th>80%</th>
<th>90%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>breast-cancer</td>
<td>SVM</td>
<td>34</td>
<td>68</td>
<td>102</td>
<td>136</td>
<td>170</td>
<td>204</td>
<td>238</td>
<td>272</td>
<td>306</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OCVM</td>
<td>215</td>
<td>178</td>
<td>102</td>
<td>133</td>
<td>223</td>
<td>196</td>
<td>166</td>
<td>136</td>
<td>118</td>
<td>176</td>
<td></td>
</tr>
<tr>
<td>liver-disorder</td>
<td>SVM</td>
<td>17</td>
<td>34</td>
<td>51</td>
<td>68</td>
<td>85</td>
<td>102</td>
<td>119</td>
<td>136</td>
<td>153</td>
<td>170</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OCVM</td>
<td>171</td>
<td>171</td>
<td>171</td>
<td>170</td>
<td>170</td>
<td>171</td>
<td>170</td>
<td>170</td>
<td>170</td>
<td>171</td>
<td></td>
</tr>
<tr>
<td>heart</td>
<td>SVM</td>
<td>13</td>
<td>27</td>
<td>40</td>
<td>54</td>
<td>67</td>
<td>81</td>
<td>94</td>
<td>108</td>
<td>121</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OCVM</td>
<td>99</td>
<td>103</td>
<td>83</td>
<td>103</td>
<td>100</td>
<td>106</td>
<td>97</td>
<td>116</td>
<td>78</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>web Spam</td>
<td>SVM</td>
<td>192</td>
<td>284</td>
<td>576</td>
<td>768</td>
<td>960</td>
<td>1152</td>
<td>1344</td>
<td>1536</td>
<td>1728</td>
<td>1920</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OCVM</td>
<td>294</td>
<td>294</td>
<td>294</td>
<td>294</td>
<td>294</td>
<td>294</td>
<td>294</td>
<td>294</td>
<td>294</td>
<td>294</td>
<td></td>
</tr>
<tr>
<td>ionosphere</td>
<td>SVM</td>
<td>17</td>
<td>35</td>
<td>52</td>
<td>70</td>
<td>87</td>
<td>105</td>
<td>122</td>
<td>140</td>
<td>157</td>
<td>175</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OCVM</td>
<td>140</td>
<td>137</td>
<td>139</td>
<td>105</td>
<td>122</td>
<td>123</td>
<td>135</td>
<td>133</td>
<td>130</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>iris</td>
<td>SVM</td>
<td>7</td>
<td>15</td>
<td>22</td>
<td>30</td>
<td>37</td>
<td>45</td>
<td>52</td>
<td>60</td>
<td>67</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OCVM</td>
<td>67</td>
<td>70</td>
<td>70</td>
<td>64</td>
<td>67</td>
<td>69</td>
<td>64</td>
<td>67</td>
<td>70</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>wine</td>
<td>SVM</td>
<td>8</td>
<td>17</td>
<td>25</td>
<td>34</td>
<td>42</td>
<td>51</td>
<td>59</td>
<td>68</td>
<td>76</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OCVM</td>
<td>87</td>
<td>86</td>
<td>85</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>vehicle</td>
<td>SVM</td>
<td>42</td>
<td>84</td>
<td>126</td>
<td>168</td>
<td>210</td>
<td>252</td>
<td>294</td>
<td>336</td>
<td>378</td>
<td>420</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OCVM</td>
<td>71</td>
<td>72</td>
<td>72</td>
<td>70</td>
<td>72</td>
<td>70</td>
<td>73</td>
<td>73</td>
<td>72</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>vowel</td>
<td>SVM</td>
<td>49</td>
<td>99</td>
<td>148</td>
<td>198</td>
<td>247</td>
<td>297</td>
<td>346</td>
<td>396</td>
<td>445</td>
<td>495</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OCVM</td>
<td>111</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>112</td>
<td>110</td>
<td>112</td>
<td>111</td>
<td>111</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>KDD99</td>
<td>SVM</td>
<td>2500</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td>OM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OCVM</td>
<td>122</td>
<td>134</td>
<td>134</td>
<td>142</td>
<td>145</td>
<td>145</td>
<td>146</td>
<td>146</td>
<td>147</td>
<td>148</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.8: Number of support vectors/core vectors comparison. OM indicates out of memory caused by too large size of data input.
This chapter presents a high-speed hierarchical multi-label classification algorithm, called the hierarchical core vector machine (HCVM), for labelling hierarchical multi-label structure stream data.

We model the multi-label hierarchy into a data Hyper-Sphere constructed by numbers of minimum enclosing balls/core vector machines (MEB/CVM). The MEBs/CVMs are separating, encompassing and overlapping with each other formed as a tree structure, representing the multi-label hierarchy encoded. Provided an unlabelled sample, the HCVM seeks a MEB/CVM enclosing the sample, and multi-label the sample according to the MEB’s position in the hierarchy. The proposed HCVM has been examined on a Gaussian synthetic data.

This chapter is structured as follows: Section 5.1 describes related researches on machine learning method for network intrusion detection and multi-label classification methods. Section 5.2 presents the proposed HCVM learning. In Section 5.3, we cover experimentation and algorithm evaluation. Lastly, in Section 5.4 we draw our conclusion and state future directions.

5.1 WHY HIERARCHICAL LEARNING? - ANSWER FOR QUESTION 2

Previous supervised learning methods group any network traffic into several major clusters, disregarding the complexity fact that one major category communication includes often several subcategories, and so on for those subcate-
5.1 Why Hierarchical Learning? - Answer for Question 2

In general, network traffic poses, by its nature, a complex multi-label, rather than a single-label status.

5.1.1 Review of Multi-label Classification Methods

The existing multi-label classification methods can be grouped into two main categories: a) problem transformation methods and b) algorithm adaptation methods.

The problem transformation methods were defined in (Tsoumakas & Katakis, 2007) as those methods that transform a multi-label classification problem into a multiple single-label’s classification problem (Boutell, 2004; Diplaris, 2005). Given a dataset $X$ with a set of class labels $L$ for multi-label classification training, a common problem transformation method is to train $|L|$ binary classifiers $H_l : X \rightarrow \{l, \neg l\}$, one for each individual label $l$ in $L$. Thus, the original dataset is transformed into $|L|$ data sets, for the $l^{th}$ dataset $D_l$ the same instance of the original dataset whose instances are labelled as $l$ or $\neg l$, and a binary classifier is applied for the classification on the dataset. In this way, when an unlabeled instance $x$ is provided for classification, a set of labels are produced simultaneously by $|L|$ classifiers,

$$H(x) = \bigcup_{l \in L} \{l\} : H_l(x) = l.$$ (5.1)

Algorithm adaptation methods extend the existing learning algorithms such as AdaBoost to handle multi label data directly. Adaboost.MH and Adaboost.MR (Schapire, 2000) are two extensions of AdaBoost (Freund, 1997) for multi-label classification. Both extensions are used on weak classifiers represented in the form $H : X \times L \rightarrow R$. Given an unlabelled instance $x$, in AdaBoost.MH, $x$ is labelled as $l, l \in L$ if the weak classifier for $l$ is positive. In AdaBoost.MR, the output of the weak classifiers is considered for ranking each of the labels in $L$ and $x$ is labelled as $l$ which is the top rank class label.
5.2 THE PROPOSED HCVM LEARNING METHOD

This section introduces the proposed HCVM method for hierarchical multi-label classification in the context of network security stream data.

5.2.1 MEB/CVM and Motivations of CVM for HMC

The minimum enclosing ball computes the ball(s) of minimum radius enclosing a given set of points. Traditional algorithms for finding exact MEBs/CVMs developed by Megiddo (Megiddo, 1983) and Welzl (Welzl, 1991) do not scale well with higher dimensional datasets. Recently, approximation algorithms for finding MEBs/CVMs have been given by Badoiu (Bădoiu et al., 2002) and Kumar (Kumar et al., 2003). Badoiu indicated that a \((1 + \epsilon) - approximation\) MEB can be efficiently obtained by using a subset of the input dataset, called the core set. Additionally, Badoiu (Bădoiu et al., 2002) found that the size of the MEB core set is independent of both dataset dimensionality \(d\) and the size of the dataset. Kumar et al. (Kumar et al., 2003) developed methods for computing core sets and approximate the smallest enclosing hyper-spheres in high dimension spaces. In this way, the MEB can be implemented in applications with large numbers of numerical attributes.

As mentioned above, a straightforward method is to transfer a HMC problem to a number of single class classification problems using a certain transformation method (Boutell, 2004; Diplaris, 2005), so that a HMC problem can be managed as a typical multi-class classification. The disadvantage of this method is, the original label hierarchy information (i.e. the partial order of hierarchical multi-labels) of the data is changed or lost completely, which eventually leads to an unsatisfied multi-label classification.

Alternatively, we consider here the HMC problem from the viewpoint of MEB (Bădoiu et al., 2002; Kumar et al., 2003), for each single class of the HMC problem, wherever it is located in the label hierarchy, it can be approximated by an MEB. One class is represented as one MEB, meanwhile the MEB may overlap with MEBs/CVMs from other classes, or encompass MEBs/CVMs from its child classes. In this way, the HMC problem is addressed by
a set of related MEBs/CVMs, in which the relationship of MEB represents the label hierarchy, so that the original hierarchy information of data get reserved in MEB modelling.

5.2.2 HMC Problem Transformation

Hierarchical multi-label classification (HMC) is an extension of binary classification where an instance can be labeled with multiple classes that are organized in a hierarchy (Hendrik, Leander, Jan, & Amanda, 2006).

In the example of network intrusion detection, Fig. 5.1 describes a typical HMC adapted from the KDD’99 dataset, where different network connection types are divided into 4 major categories and 21 subcategories. The relationship between these subcategory connection types and major connection category types are structured as a Markov Tree. The major connection types are set as the branch of the tree, and those subcategory types as the leaves of the tree. In this way, the problem of Internet intrusion detection is interpolated into the learning of hierarchical multi-label classification. Here, the definition of HMC task is briefed as follows (Hendrik et al., 2006):

Given: (1) data set \( S = \{x_1, \ldots, x_m\} \), where each \( x_i \in \mathbb{R}^d \), (2) class label set \( C = \{c_1, \ldots, c_n\} \), and (3) a class hierarchy \((C, \leq_h)\), where \( \leq_h \) is a partial order representing the parent class relationship (\( \forall c_1, c_2 \in C : c_1 \leq_h c_2 \), if and only if \( c_1 \) is a superclass of \( c_2 \)).

Find: a function \( f : x \rightarrow C_i \) where \( x \in S \) and \( C_i \subseteq C \), \( C_i \) includes an leaf class \( c \) and its parent classes \( c' \) such that \( c \in C_i \Rightarrow \forall c' \leq_h c : c' \in C_i \).

In HMC, \( f \) is able to classify an unlabelled message as normal, or as a subcategory attack type together with all its parent category attack types.

5.2.3 Hierarchical Minimum Enclosing Ball

For HMC, the above kernel MEB is extended to the approximation of hierarchical minimum enclosing balls (HCVM) as follows:
Given dataset \( S = (X \times Y) \) where \( X \in \mathbb{R}^d \), and class label set \( Y \) is in such a hierarchy as \( Y = \{Y_i|y \leq_h \forall y'\} \) whose \( y, y' \in Y_i \), and \( y' \neq y \). To transform the HMC problem to a HCVM problem, we seek the root class label set \( C_1 \subseteq Y \) where \( \forall y^l \in C_1 : \neg \exists y'^l \leq_h y^l : y'^l \in Y, y'^l \notin C_1 \). For each root class label \( y^l \in C_1 \), we create an MEB, \( B \), and mark it with (1) its hierarchical level which is 1, (2) the corresponding class label \( y^l \), and (3) its parent class. (we use 0 to represent the parent class of a root class). Thus, for each class \( y^l \) in the root class label set \( C_1 \), we obtain the corresponding MEB \( B_1^{y^l,0} \). Similarly, we address the next level class set \( C_2 \subseteq Y \) where \( \forall y'^l \in C_2 : \exists p'^l \leq_h y'^l : p'^l \in C_{2-1} \), and create the MEB \( B_2^{y^l,p^l} \) for each class label \( y'^l \in C_2 \). This process is continued until no child classes can be found.

As a result, we have a multi-label hierarchy modelled as an HCVM structured as a set of MEBs/CVMs ordered with its class label, level in the hierarchy and its parent class,

\[
\{Y_i|y \leq_h \forall y'\} \rightarrow \{B_1^{y^l,0}, B_2^{y^l,p^l}, ..., B_i^{y^l,p^l}\}. \tag{5.2}
\]
While we transfer a multi-label hierarchy to an HCVM, the knowledge of the relationship between classes at the same level naturally appears. This is because that label represents the class as a signature, but HCVM represents the class in a data distribution space, it captures not only the parenthood relationship (i.e. parent class MEB encompasses the child class MEB) but also the relationship between classes at the same level of the hierarchical structure (i.e. same level classes MEBs/CVMs may separating or overlapping).

For decision making, we expect that \( f(B) \) approximates the ground truth multi-label \( f(x) \) better than \( f(X,Y) \) because each MEB in HCVM approximates accurately its associated class of \( Y \), preserving perfectly the multi-label hierarchy in \( Y \). Thus, the HCVM eventually can be learned by an aggregation of MEBs/CVMs as,

\[
f(x) = f(\{B^y_l\}) = \bigvee_{i=1}^{m} \bigvee_{l=1}^{t_i} f_{B_{i,l}} \tag{5.3}
\]

where \( f_{B_{i,l}} \) represents an elementary MEB model on class \( y^l_i \). In Eq. (5.3), \( \bigvee \) denotes the union between models, in which \( \bigvee \) represents a natural parent relationship. As modelling \( f_{B_{i,l}} \), class \( y^l \) is addressed as a binary problem with class \( y^l \) as the positive and the remaining classes as the negative, and a set of MEBs/CVMs are computed according to Eq. (4.1)- Eq. (4.3) to enclose only the positive instances. Then, \( f_{B_{i,l}} \) is formed by assigning the corresponding MEB core sets as core vectors.

Given an instance \( x \) enclosed in one \( i^{th} \)-layer MEB, due to the parent relationship, it must also be enclosed in the \( (i - 1)^{th}, (i - 2)^{th}, \ldots, 1^{st}, 0 \) layer ancestor MEB; and \( \bigvee \) represents a relationship at the same layer of HCVM. \( l \) is often fixed by distances calculation. To compute the distance between \( x \) and the center \( c_t \) of \( B_t \), \( x \) is mapped into the feature space by a kernel function \( k(x,x) \). By Eq. (4.3), the distance between \( x \) and \( c_t \) is calculated as:

\[
\|c_t - \varphi(x)\|^2 = \sum_{z_i, z_j \in S_t} \alpha_i \alpha_j k(z_i, z_j) - 2 \sum_{z_i \in S_t} \alpha_i k(z_i, x) + k(x,x). \tag{5.4}
\]

Fig. 5.2 gives an example of HCVM data approximation over a synthetic dataset with 2 layers, \( A_1 \leq h [A_2, A_3, A_4] \). To approximate the data distribution of the dataset, an individual MEB can be used to enclose data in class \( A_1 \).
Figure 5.2: Core set data approximation comparison: boundary on the MEB core set from the parent class A1 versus boundary on the proposed HCVM ($A_1 \leq_h [A_2, A_3, A_4]$) core set.

As shown in Fig. 5.2, the resulting boundary (identified as the dashed curve) summarizes the knowledge of MEB (represented as the solid line cycle) and the MEB core set (identified as squared data points). Based on the fact that if an instance is in a parent class, then it must belong to one of its child classes, the proposed HCVM method approximates the dataset by combining core sets from $A_1$ MEB and MEBs/CVMs of $A_1$’s 3 child classes $A_2$, $A_3$ and $A_4$, and has the boundary for data approximation plotted in Fig. 5.2 right.

It clearly shows that the resulting boundary enclosed almost 2 times the space than the obtained boundary did. However, no data point exists in the extra space. Thus, the HCVM method gives a much smaller enclosing space while discarding the non-essential areas from the MEB for the parent class only.
According to the theories discussed above, we have developed the following 3 algorithms for hierarchical multi-label classification. Given dataset $S = (X \times Y)$ where $X \in \mathbb{R}^d$, and class label set $Y$ is in such a hierarchy as $Y = \{Y_i|y \leq_h y' \forall y_i\}$ whose $y, y' \in Y_i$, and $y' \neq y$. Algorithm 4 computes the MEB and core set for $S$. Algorithm 5 constructs HCVM where Algorithm 4 is used for modelling kernel MEBs/CVMs on subclasses $S_i = (X_i, y_i) \subset S$. Note that the constructed HCVM $B$ is an $m$ level hierarchical MEB, $B = \{B_{y_1}^{y}, p_1, B_{y_2}^{y}, p_2, \ldots, B_{y_t}^{y}, p_t\}$ where $y^i, p^i$ denote the corresponding class label and the parent-class label respectively.

**Input:** Set of points $S \in \mathbb{R}^d$; parameter $\epsilon = 2^{-m}$; subset $Q_0 \subset S$

**Output:** A $(1 + \epsilon)$-approximation MEB(S); $O(1/\epsilon)$-size core set $Q$

1. for $i \leftarrow 1$ to $m$ do
2. loop
3. $Q \leftarrow Q_{i-1}$
4. Compute $B_{c,r} = MEB(Q)$
5. if $S \subset B_{c,(1+\epsilon)r}$ then
6. Return $B_{c,r}, Q$
7. else
8. $p \leftarrow$ point of $S$ maximizing distance, $\|cp\|$, from $c$
9. end if
10. $Q \leftarrow Q_{i-1} \cup \{p\}$
11. end loop
12. $Q_i \leftarrow Q$
13. end for

**Algorithm 4:** MEB algorithm

Algorithm 6 tests the constructed HCVM for the multi-label classification of a new instance $x$. In order to check whether $x$ is located in the range of the nearest MEB’s radius an iterative checking process is run until one nearest MEB is found at every level of the HCVM. Associating together as the set of located MEBs/CVMs, it comes up with a complete tree path from the root to the terminal level of the HCVM. Along the tree path, the instance $x$ is branched...
5.2 THE PROPOSED HCVM LEARNING METHOD

**Input:** Set of points \( S = (X \times Y) \) where \( Y = \{Y_i | y \leq_h y' \} \) whose \( y, y' \in Y_i \), and \( y' \neq y, \epsilon \).

**Output:** HCVM \( B \); the \( O(1/\epsilon) \)-size core set \( Q \).

1: Initial \( i \leftarrow 1 \).
2: Find \( C_i \subseteq Y \) where \( \forall y \in C_i : \neg \exists y' \leq_h y : y' \in Y, y' \notin C_i \).
3: while \( C_i \neq \phi \) do
4: \( t_i \leftarrow |C_i| \).
5: for \( l \leftarrow 1 \) to \( t_i \) do
6: \( S = \{\varphi(x)\}, x = (X, y^l) \) where \( y^l \in C_i \) and \( Q = 0 \).
7: \([\text{MEB}(b), \text{CoreSet}(q)] \leftarrow \text{Run Algorithm 4 with input } S, Q, \text{ and } \epsilon \).
8: if \( i=1 \) then
9: \( p^l \leftarrow 0 \).
10: else
11: \( p^l \leftarrow p \) where \( p \leq_h y^l \).
12: end if
13: \( B_{i}^{p^l,y^l} \leftarrow b, Q_{i}^{p^l,y^l} \leftarrow q \).
14: \( i = i + 1 \).
15: end for
16: Find \( C_i \subseteq Y \) where \( \forall y \in C_i : \exists p \leq_h y : p \in C_{i-1} \).
17: end while

**Algorithm 5:** Hierarchical core vector machines training algorithm

from the root progressively to the terminal level of HCVM, meanwhile multi-labeling \( x \) with the associated labels of MEBs/CVMs at every level.

5.2.5 Time and Space Complexities for HCVM

Note that CVM has the computational complexity \( O(\sqrt{nd^2(n+d)\log(n/\epsilon)}) \) (Kumar et al., 2003). Using CVM for HMC problem, the expected computational complexity is \( O(\sqrt{nd^2(n+d)\log(n/\epsilon)} + \sum_{i=1}^{m} \sqrt{n_i d^2(n_i + d)\log(n_i/\epsilon)}) \), where \( m \) is the number of child classes. This obviously is less expensive than the quadratic programming (QP) methods, such as SVM, whose training time complexity is \( O(n^3) \). In this sense, a HMC problem can be modelled by MEB more efficiently, especially for problems involving large size dataset. CVM core set estimates the boundary of a given class in \( O(1/\epsilon) \) (Kumar et al., 2003), thus MEB in the nature is independent on dataset size \( n \) and dimension \( d \), which indicates that MEB is deterministically robust to any class-
**Input:** Instance $x$; HCVM $B$

**Output:** labels of $x$ $Label$

1: Transfer $x$ to kernel $k(x, x)$ by putting it in feature map $\varphi$.
2: $Label \leftarrow 0$.
3: $index = \left\{ 1, 1, \ldots, 1 \right\}$.
4: $i \leftarrow 1$.
5: **while** $i \leq m$ **do**
6:  **for** $j \leftarrow index(i)$ **to** $n$ **do**
7:  Find MEB $B^y_i \varphi \in B$ where $p^l = Label(i)$, $B^y_i \varphi$'s center is $j^{th}$ nearest to $k(x, x)$.
8:  **if** $k(x, x)$ inside of boundary formed by core set of $B_k$ **then**
9:    $Label \leftarrow Label \cup l$.
10:   **Break.**
11:  **else if** $j == n$ **then**
12:    $Label \leftarrow Label - Label(i)$.
13:    $i \leftarrow i - 1$.
14: **end if**
15: **end for**
16: **if** $Label$ is empty **then**
17:    Return $Label \leftarrow 0$.
18: **end if**
19: $index(i) = index(i) + 1$.
20: $i \leftarrow i + 1$.
21: **end while**

**Algorithm 6:** Hierarchical core vector machines testing algorithm

Imbalanced and/or high dimensional datasets. In HCVM, maximum $m$ MEB-s/CVMs are used for HMC problem solving where $m$ represents the number of child classes. Thus, the $O((m + 1)/\epsilon)$ computational complexity of HCVM enable solving of any HMC problem more efficiently, meanwhile immune to the class-imbalanced and/or high dimensional difficulty.

### 5.3 Validity Examination on Synthetic Dataset

For testing the capability of the proposed HCVM for multi-label, in particular for hierarchical multi-label classification, we examined the developed HCVM on a synthetic dataset.
The synthetic dataset for HCVM examination is a mixture of several 2D Gaussian labelled with a 2-level multi-label hierarchy, \( \mathcal{H} = \{ \text{class 1: \{class(1,1), class(1,2), class(1,3) \}; class 2: \{class(2,1), class(2,2), class(2,3) \}} \}, \) in which the distribution of class 1 data is shown in Fig 5.3.

5.3.1 Results and Discussion

Using MEB, class 1 can be approximated by constructing a single independent MEB as Fig. 5.4a, and the obtained core set, as support vectors for building class 1 descriptive plane, is presented in Fig. 5.4c. Alternatively using HCVM, the MEB can be constructed instead as Fig. 5.4b for both the parent and the child classes as class 1 in fact has 3 child classes. Thus, a 2-level HMC problem is modelled by HCVM since a set of correlated MEBs/CVMs (i.e a hierarchical MEB). Fig. 5.4d gives accordingly the obtained core sets from 4 correlated MEBs/CVMs. As seen, it is apparently that the core set in Fig. 5.4d approximates class 1 data more accurate than the core set in Fig. 5.4c. This demonstrates that the developed HCVM derives important additional discriminant information from the 2-level label hierarchy.

Figure 5.3: Synthetic data for HMC, where data points of different classes are represented as different colors.
5.4 SUMMARY

In this chapter, we studied HMC problem aiming to multiply labelling network intrusions for Internet security. The HMC differs from typical classification in two aspects: (1) an instance belongs often to more than one class simultaneously; (2) every instance from any class belongs automatically to its ancestor classes, as classes in HMC are in hierarchical tree structure.

By aggregating MEBs/CVMs, we proposed a novel HCVM model in which MEBs/CVMs separate, encompass and overlap with each other representing an organized multi-label tree structure. Moreover, we have modified the original MEB algorithm using the Gaussian kernel method, enabling MEBs/CVMs to be applicable for kernel computing in a higher dimensional space.

The results also shows that, HCVM is a local modelling method which learns the local information from sub-classes. And local modelling beat global modelling by obtaining a much more accurate class boundary.

Figure 5.4: An example of using MEB for HMC, where * represents a MEB core set point. Note that this figure is best seen in color.
In general, the proposed HCVM method has the following desirable properties. First, HCVM models a difficult hierarchical multi-label problem as a simple MEB association analysis. Second, MEB excludes the existing sparseness of data, which enables HCVM to approximate data more accurately. However, the proposed method inhibits a limitation that the classification performance of the terminal classes might be inferior to that of non-terminal classes since the boundaries for the terminal classes in a multi-label hierarchy are not optimized.

The next chapter introduces a on-line local modelling method which is capable of not only hierarchical structured stream datasets but normal datasets.
Chapter 6

DYNAMIC EVOLVING CORE VECTOR MACHINES - A FAST CONNECTIONIST-BASED KERNEL LEARNING SYSTEMS

This chapter introduces a new type of kernel-based learning algorithm, denoted as dynamic evolving core vector machines (DE-CVM). This method is developed for adaptive on-line stream data learning. Similar to DENFIS (Kasabov & Song, 2002), DE-CVM evolve through incremental, hybrid (supervised/unsupervised) learning, and accommodate new input data, including new features, new classes, etc., through local element tuning.

This chapter is structured as follows: Section 6.1 explains why DE-CVM is essential for stream data learning. Section 6.2 gives the motivations for the proposed DE-CVM. The algorithm is presented in section 6.3. Section 6.4 covers experimentation and algorithm evaluation. Lastly, in section 6.5 we draw our conclusion.

6.1 WHY EVOLVING SYSTEMS ARE ESSENTIAL FOR STREAM DATA MINING?
- ANSWER FOR QUESTIONS 3

Kasabov and Song (2002) maintained that, the complexity and dynamics of real-world problems, especially in engineering and manufacturing, require sophisticated methods and tools for building on-line, adaptive intelligent systems (ISs). Such systems should be able to grow as they operate, to update their knowledge and refine the model through interaction with the environment.
Stream data mining focuses on real-world applications, such as network intrusion detection, texture categorisation and image recognition. These applications need machine learning methods that have the following characteristics: fast learning, on-line incremental adaptive learning, open structure organisation, memorising information, active interaction, knowledge acquisition and self-improvement, and spatial and temporal learning (Kasabov & Song, 2002).

We adopt evolving connectionist systems (ECOSs) (Kasabov, 1998) for stream data mining because ECOSs evolve their structure and functionality from a continuous input data stream in an adaptive, life-long, modular way. Another important reason is that ECOSs employ local learning as they create connectionist-based modules and connect them, if that is required according to the input data distribution and the system’s performance at a certain time moment.

6.2 MOTIVATION OF DE-CVM

The minimum enclosing ball (MEB) problem is to compute a ball of minimum radius enclosing a given set of objects (points, balls, etc.) in $\mathbb{R}^d$. It has been widely implemented for clustering applications, such as spatial hierarchies (Hubbard, 1996), and support vector clustering (Ben-Hur, Horn, Siegelmann, & Vapnik, 2002); classification applications, such as area gap tolerant classifiers (Burges, 1998b), and core vector machine (CVM) (Tsang et al., 2005); as well as approximation applications, such as fast farthest neighbour query approximation (Goel, Indyk, & Varadarajan, 2001), and 1-cylinder problem approximation (Bădoiu et al., 2002).

Classic CVM for classification computes a $(1 + \varepsilon)$-approximation (Bădoiu et al., 2002) for a minimum radius ball learning, and extracts those data points located at the outer area of a CVM for classification modelling. The set of those extracted data points characterise the entire given dataset, and thus are called core vector set or core set. For classification modelling, CVM can be used to approximate each class data distribution, so that one class can be distinguished from another by core set computing.
However, in practice such classic MEB has the following difficulties: (1) CVM often encloses sparseness together with data. To enclose an isolated outlier point, a huge MEB is required, which makes the MEB include actually more sparseness than the data occupation; (2) MEB is keen on enclosing data, and thus disables the detection of any outliers despite the outliers producing the sparseness of MEB.

To mitigate the above problems, we propose a novel dynamic evolving CVM (DE-CVM) approach to learning the core set in a group manner. DE-CVM sets MEB/CVM in different data distribution area, reducing the sparseness in CVM by decomposing data space based on data distribution density, discriminating core vectors on class interaction hyperplanes, and enabling outlier detection to decrease the effects of noise.

To have a better and clear presentation of the proposed one-pass MEB, Table 6.1 presents the symbols that will be used in this chapter.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>data matrix</td>
</tr>
<tr>
<td>$X_i$</td>
<td>data matrix for $i$-th class</td>
</tr>
<tr>
<td>$n$</td>
<td>number of training data instances</td>
</tr>
<tr>
<td>$n_i$</td>
<td>number of instances for $i$-th class</td>
</tr>
<tr>
<td>$l$</td>
<td>number of classes</td>
</tr>
<tr>
<td>$B_X$</td>
<td>minimum enclosing ball/balls (MEB) based on data set $X$</td>
</tr>
<tr>
<td>$c_{i,j}$</td>
<td>$i$-th class $j$-th MEB center/centers</td>
</tr>
<tr>
<td>$r_{i,j}$</td>
<td>$i$-th class $j$-th MEB radius</td>
</tr>
<tr>
<td>$k_i$</td>
<td>total number of MEBs for $i$-th class</td>
</tr>
<tr>
<td>$Q_{i,j}$</td>
<td>$i$-th class $j$-th MEB core set</td>
</tr>
<tr>
<td>$\Omega_{MEB}$</td>
<td>classic MEB learning model</td>
</tr>
<tr>
<td>$\Omega_{cMEBs}$</td>
<td>cross-MEBs learning model</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>MEB approximation value</td>
</tr>
</tbody>
</table>

Table 6.1: NOTATION
In terms of complex data distribution, a large MEB/CVM follows the sparseness of the ball, thus it is often unable to approximate a data distribution accurately. A single MEB/CVM for one class data is likely to enclose wrongly almost all data points from another class, especially when data is zonally distributed. As a solution, a number of smaller MEBs/CVMs are able to drill into the details of any data distribution, apparently allowing a more accurate approximation.

Motivated by this, the above kernel MEB is renovated for group manner MEB/CVM computing (DE-CVM). Instead of addressing a whole class data $X_i$ with one MEB/CVM $B_Q$, DE-CVM models class data using with a set of MEBs $B_Q = \bigcup_{j=1}^{k} B_{Q_{i,j}}$, where $k$ is the number of MEBs/CVMs. Consider that MEB/CVM learning is an iterative learning process, we represent an individual kernel MEB/CVM here as $B_{Q_{i,j}} = \{c_{i,j}, (1+\varepsilon)r_{i,j}, \varphi(Q_{i,j})\}$ with $j$ as the index of MEB/CVM, and $u$ as the iteration number of MEB/CVM updating. In this way, given $X = \bigcup_{i=1}^{d} X_i$ as the training dataset, the proposed DE-CVM learning is described below.

Similar to CVM, we initialise one MEB/CVM (i.e. $B_{Q_{i,k}}$; $k = 1$) on one class data $X_i$ at very beginning. Here, we initialise the core set as $Q_{i,1} = \{\varphi(x_a), \varphi(x_b)\}$, where $x_a$ is the furthest data point to a random $x \in X_i$ and $x_b$ is calculated as:

$$x_b = x_a + \frac{x_a - \underset{x \in X}{\text{arg max}} ||x_a - x||}{\lambda}, \quad \lambda > 1.$$ (6.1)

Then, we obtain $B_{Q_{i,1}} = \{c_{i,1}, (1+\varepsilon)r_{i,1}, Q_{i,1}\}$ using CVM optimisation function.

**Theorem 6.3.1.** The first initialised MEB/CVM’s radius $r_{i,1} \approx \frac{\Delta}{2\lambda}$, where $\Delta$ denotes the diameter of $X_i$.

**Proof.** Science $Q_{i,1} = \varphi(x_a, x_b)$, $x_a$ is the furthest point to $x_1$. Clearly, the distance from $x_a$ to its furthest data point approximates to $\delta$. Then, $x_b$ locates about $\Delta/\lambda$ away from $x_a$. According to the definition, $r_{i,1}$ is the radius of $B_{Q_{i,1}}$, we have $r_{i,1} \approx \frac{\Delta}{2\lambda}$. \hfill \square
6.3 THE PROPOSED DE-CVM LEARNING

If there is \( x \in X_i \) not contained in any MEB \( B_{Q_{i,j}^{u-1}} \), \((j = \{1, 2, \ldots, k\})\). It results in one of the following 3 DE-CVM updating cases.

In the first case, if MEB \( B_{Q_{i,j}^u} \) over the existed core set \( Q_{i,j}^{u-1} \) with \( x \) (i.e. \( Q_{i,j}^u = Q_{i,j}^{u-1} \cup \varphi(x) \)) has radius \( r_{i,j}^u \), which is less or equal to the upper bound value as \( 1 + \frac{\varepsilon^2 x n}{16} \), \((\eta > 1)\) times of the existed radius \( r_{i,j}^{u-1} \). The existing MEB \( B_{Q_{i,j}^{u-1}} \) is updated by replacing with \( B_{Q_{i,j}^u} \). Note that, \( c_{i,j}^{u-1} \) is the closest MEB centre to \( x \).

**Theorem 6.3.2.** Given an upper bound of radius increment as \( r_{i,j}^u \leq (1 + \frac{\varepsilon^2 x n}{16}) r_{i,j}^{u-1} \), the DE-CVM expansion is between \( \frac{32}{\varepsilon^2} \) to \( \frac{128 \lambda \delta}{\varepsilon^2} \) times.

**Proof.** Since \( r_{i,j}^0 \geq \frac{\Delta}{2\lambda} \), and each step we increase the radius by at least \( (\Delta/4)\varepsilon/16 = \Delta\varepsilon/64 \), it follows that we cannot encounter this case more than \( 64/\varepsilon \) times, as \( \delta \) is an upper bound of the radius of the minimum enclosing ball of \( X_i \). \( \square \)

In the second case, if radius \( r_{i,j}^u \) is greater than the upper bound value \( (1 + \frac{\varepsilon^2 x n}{16}) r_{i,j}^{u-1} \), a new fragmentary core set \( Q_{i,j}^0 = \{\varphi(x)\} \) is created as a completed core set which has at least 2 vectors.

In the third case, if the distance from \( x \) to the closest fragmentary core set \( Q_{i,j}^0 \) is less than \( \frac{\Delta}{X} \), we add \( \varphi(x) \) into the fragmentary core set \( Q_{i,j}^0 \) as \( Q_{i,j}^0 = \{Q_{i,j}^0 \cup \varphi(x)\} \), in this way, the fragmentary core set \( Q_{i,j}^0 \) becomes completed. In addition, a new MEB \( B_{Q_{i,j}^0} = \{c_{i,j}^0, (1+\varepsilon)r_{i,j}^0, Q_{i,j}^0\}, (k = k+1) \) is created using CVM optimisation function (Y. Chen, Pang, Kasabov, Ban, & Kadobayashi, 2009). The threshold \( \frac{\Delta}{X} \) is also considered as the outliers threshold, as if a single data point away from the rest data farther than the threshold, this data point is treated as an outlier by DE-CVM.

**Theorem 6.3.3.** The total number of DE-CVM \( k \) is equal to approximately \( \lambda/2 \).

**Proof.** Science we guarantee that the radius of each firstborn DE-CVM \( r_{i,j}^0 \) \((j = \{1, 2, \ldots, k\})\) is less than \( \frac{1}{X} \) diameter of the given data \( X_i \), the total number of the DE-CVM \( k \) approximates to \( \lambda/2 \). \( \square \)

The DE-CVM updating is terminated once \( X_i \subset \bigcup_{j=1}^kB_{Q_{i,j}^k} \). For the overall dataset \( X = \bigcup_{i=1}^I X_i \), let \( k_i \) be the number of MEBs/CVMs of \( i-th \) class, then we factorise the core sets by abandoning these core vectors contained in just
one MEB/CVM. As a result, we have the DE-CVM model \( \Omega_{DE-CVM} \) as the set of MEBs/CVMs that constructed by the above 3 DE-CVM updating cases.

\[
\Omega_{DE-CVM} = \left( \bigcup_{j=1}^{k_1} c_{1,j} \cup \bigcup_{j=1}^{k_1} (1 + \varepsilon) r_{1,j} \cup \bigcup_{j=1}^{k_1} Q_{1,j} \right) \cup \left( \bigcup_{j=1}^{k_2} c_{2,j} \cup \bigcup_{j=1}^{k_2} (1 + \varepsilon) r_{2,j} \cup \bigcup_{j=1}^{k_2} Q_{2,j} \right) \cup \cdots \cup \left( \bigcup_{j=1}^{k_l} c_{l,j} \cup \bigcup_{j=1}^{k_l} (1 + \varepsilon) r_{l,j} \cup \bigcup_{j=1}^{k_l} Q_{l,j} \right),
\]

and summarise the computation of DE-CVM learning as Algorithm 7.

6.4 EXPERIMENTS AND DISCUSSIONS

In this section, we describe two experiments where we used DE-CVM for banana data verification and for benchmark UCI data classification.

6.4.1 Class Factorisation Ability Test

To highlight the class factorisation ability of the proposed DE-CVM, we cluster an artificial two-dimensional banana dataset (see Fig. 6.1a) containing 200 data points in a banana shaped distribution using both DE-CVM and classic CVM. Fig. 6.1b illustrates the cluster boundary formed by 70 classic CVM core vectors over the banana data set, where star symbols represent the core vectors and a solid line represents the boundaries. From 6.1b, it can be seen that all the MEB/CVM core vectors are positioned around the outer area of data distribution. As a result, data points located around the inner curvature of the banana distribution are not encompassed by the cluster boundary.

To overcome the difficulty of forming the inner curvature area of the banana data cluster, we employ the proposed DE-CVM method by inputting a new cluster dataset which contains 80 data points located close to the inner curvature area of the banana dataset (see Fig. 6.1c, the new cluster is represented by squares). Then we perform DE-CVM on both datasets by configuring the banana dataset as a positive class while the new cluster is configured as negative class. Fig. 6.1c illustrates the six DE-CVM core vectors refining the cluster boundary calculated by the classic CVM, thereby accurately depicting the in-
Input: Set of points $X \in \mathbb{R}^d$, parameter $\varepsilon$, $\lambda$, and $\eta$ 

Output: A DE-CVM learning model $\Omega_{DE-CVM}$ 

1: for each $X_i \subset X$ do 
2:   Initialise $k \leftarrow 1$, $T = \emptyset$, and $\Delta$ which is the diameter of $x$ 
3:   Initialise $Q_{i,j}^0$ by equation 6.1 
4:   Compute the initial MEB $B_{i,1}^0$ and its radius $r_{i,1}^0$ and center $c_{i,1}^0$ using equation ?? on $Q_{i,1}^0$ 
5:   for each $x \in X_i$ do 
6:     if $x \not\in \bigcup_{j=1}^k B_{i,j}$ then 
7:       Find $t \leftarrow \arg \min_{t \in T} ||x - t||$ 
8:       if $||x - t|| < \frac{\Delta}{2\lambda}$ then 
9:         Remove $t$ from $T$ 
10:        $k \leftarrow k + 1$ 
11:       $Q_{i,k}^0 = \{\varphi(t), \varphi(x)\}$ 
12:       Computer a new MEB $B_{i,k}^0$ and its radius $r_{i,k}^0$ and center $c_{i,k}^0$ using optimisation function on $Q_{i,k}^0$ 
13:     else 
14:       Find $c_{i,j}^u \leftarrow \arg \min_{j \in \{1:k\}} ||x - c_{i,j}^u||$ 
15:       $Q_{i,j}^{u+1} \leftarrow \{Q_{i,j}^u, \varphi(x)\}$ 
16:       Update MEB $B_{i,j}^{u+1}$ and its radius $r_{i,j}^{u+1}$ and center $c_{i,j}^{u+1}$ using equation ?? on $Q_{i,j}^{u+1}$ 
17:       if $r_{i,j}^{u+1} > r_{i,j}^u \times (1 + \varepsilon^2 \times \eta / 16)$ then 
18:         $T \leftarrow T \cup x$ 
19:       Undo this update 
20:     end if 
21:   end if 
22: end for 
23: $\Omega_{DE-CVM} \leftarrow \Omega_{DE-CVM} \cup \{\bigcup_{j=1}^k c_{i,j}, \bigcup_{j=1}^k (1 + \varepsilon) r_{i,j}, \bigcup_{j=1}^k Q_{i,j}\}$ 
24: end for 
25: Return $\Omega_{DE-CVM}$ 

Algorithm 7: DE-CVM Algorithm

ner curvature of banana dataset. Furthermore, we increase the new cluster data by adding another 50 data points that surround the left hand side of the banana dataset, then implement DE-CVM on these two datasets. Fig. 6.1d illustrates an approximate hyperplane, which is formed by 22 DE-CVM core vectors, in order to discriminate the banana data from the new cluster data. From Fig. 6.1d, it can be inferred that DE-CVM is able to factorize the class character-
istics of the banana dataset by creating a more accurate boundary using fewer core vectors than MEB/CVM. Similarly, by adding new cluster data around the banana cluster, using the same process, we are able to reduce the core vector size due to class factorization.

![Figure 6.1: Banana data set clustering (a) plot 200 banana shaped data points, (b) plot the cluster boundary in black lines formed by 70 classic MEB core vectors, (c) plot the new cluster data in the bend area of banana data and banana data cluster boundary formed by the classic MEB core vectors and 6 DE-CVM core vectors, (d) plot the new data surround the left hand side of the banana data and the DE-CVM core vectors.](image)

6.4.2 Classification Accuracy Comparison

To evaluate the classification performance of DE-CVM, we compared DE-CVM with five other classification methods (i.e. KNN, Bayes, MLP, SVM and
CVM) on four two-class and four multi-class benchmark datasets which are summarised in Table 6.2.

<table>
<thead>
<tr>
<th>Type</th>
<th>Dataset</th>
<th>No. of Features</th>
<th>No. of Classes</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>two-class</td>
<td>heart</td>
<td>13</td>
<td>2</td>
<td>270</td>
</tr>
<tr>
<td></td>
<td>liver-disorder</td>
<td>6</td>
<td>2</td>
<td>345</td>
</tr>
<tr>
<td></td>
<td>ionosphere</td>
<td>34</td>
<td>2</td>
<td>351</td>
</tr>
<tr>
<td></td>
<td>breast-cancer</td>
<td>10</td>
<td>2</td>
<td>683</td>
</tr>
<tr>
<td>multi-class</td>
<td>iris</td>
<td>4</td>
<td>3</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>wine</td>
<td>13</td>
<td>3</td>
<td>178</td>
</tr>
<tr>
<td></td>
<td>vehicle</td>
<td>18</td>
<td>4</td>
<td>846</td>
</tr>
<tr>
<td></td>
<td>vowel</td>
<td>10</td>
<td>11</td>
<td>990</td>
</tr>
</tbody>
</table>

Table 6.2: Summarisation of data sets characteristics

For each dataset, we conducted \( K \)-fold cross-validation which randomly partitioned the original data into \( K \) subsamples. Of the \( K \) subsamples, a single subsample was retained as the validation data for testing the model, and the remaining \( K - 1 \) subsamples are used as training data. The cross-validation process was repeated \( K \) times, with each of the \( K \) subsamples used exactly once as validation data. Then we obtained the mean value of the \( K \) results from the folds. In our case, we set \( K = 10 \) as 10-fold cross-validation which is commonly used.

We also compared the class factorization ability of DE-CVM, CVM, and SVM as these three kernel-based methods extract support/core vectors which are the discriminative data points for each class. We consider a method to have strong class factorization ability, if it is able to learn most discriminative class characteristics. In other words, this is a method that is capable of extracting fewest support/core vectors and giving highest classification accuracy along with strongest class factorization ability.

Table 6.3 presents the comparison results of the classification accuracies and the number of support/core vectors among six classifiers. The three columns next to the name of datasets record classification accuracies for three conventional methods, such as KNN, Bayes, and MLP where KNN and MLP achieve
the highest accuracy on one dataset only. The remaining columns in Table 6.3 summarised the classification accuracies and number of support/core vectors extracted by those three kernel-based classification methods mentioned above. It can be observed that the three selected kernel-based methods achieve higher classification accuracy than those three conventional methods in general. DE-CVM undoubtedly shows that it has the strongest class factorisation ability among these three kernel base methods by achieving the highest classification accuracy but extracting fewest core vectors on five out of eight datasets. The results indicate that DE-CVM can be considered the perfect method for most of the selected datasets.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>KNN</th>
<th>Bayes</th>
<th>MLP</th>
<th>SVM</th>
<th>CVM</th>
<th>De-CVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
<td>acc</td>
<td>acc</td>
<td>acc</td>
<td>acc</td>
<td>acc</td>
<td>acc</td>
</tr>
<tr>
<td>heart</td>
<td>82.59%</td>
<td>75.65%</td>
<td>83.33%</td>
<td>83.19%</td>
<td>135</td>
<td>84.11%</td>
</tr>
<tr>
<td>liver-disorder</td>
<td>64.92%</td>
<td>63.17%</td>
<td>71.28%</td>
<td>64.92%</td>
<td>170</td>
<td>65.13%</td>
</tr>
<tr>
<td>ionosphere</td>
<td>64.11%</td>
<td>65.32%</td>
<td>89.16%</td>
<td>92.06%</td>
<td>135</td>
<td>92.72%</td>
</tr>
<tr>
<td>breast-cancer</td>
<td>96.63%</td>
<td>95.97%</td>
<td>96.65%</td>
<td>97.07%</td>
<td>340</td>
<td>97.07%</td>
</tr>
<tr>
<td>iris</td>
<td>94.67%</td>
<td>90.35%</td>
<td>95.97%</td>
<td>96.00%</td>
<td>73</td>
<td>96.66%</td>
</tr>
<tr>
<td>wine</td>
<td>97.16%</td>
<td>71.31%</td>
<td>93.91%</td>
<td>74.85%</td>
<td>85</td>
<td>83.12%</td>
</tr>
<tr>
<td>vehicle</td>
<td>71.04%</td>
<td>66.24%</td>
<td>77.78%</td>
<td>71.61%</td>
<td>420</td>
<td>78.13%</td>
</tr>
<tr>
<td>vowel</td>
<td>96.86%</td>
<td>91.37%</td>
<td>98.08%</td>
<td>97.47%</td>
<td>495</td>
<td>98.56%</td>
</tr>
</tbody>
</table>

Table 6.3: Classification accuracy(acc) and number of support vectors(SVs)/core vectors(CVs) comparison on eight selected data sets

6.5 SUMMARY

In this chapter, a novel DE-CVM is proposed that learns the characteristics of a class (i.e. core vectors) in a group manner. DE-CVM factorizes class characteristics by reducing the sparseness area, discriminating core vectors on class interaction hyperplanes, and enabling outliers detection. DE-CVM is evaluated by conducting a comparison of classification accuracy and computational cost on eight data sets with five conventional classification methods including original CVM. DE-CVM obtains the highest classification accuracy for five out of eight datasets. Compared to CVM, DE-CVM wins on all the data sets.
except ionosphere by utilising fewer core vectors hence rendering it computationally more efficient than CVM.

In the next chapter, we will introduce a novel string kernels classification method for string format stream data mining.
In this chapter, we investigate string format stream data modeling techniques. String classification is becoming a major area of stream data learning. This is because of the explosive growth of Internet users. Network intrusions such as SPAM emails, malicious software which formed as sequences of string data are also increasing (Shawe-Taylor & Cristianini, 2004).

Recently, string kernels-based support vector machines have shown competitive performance in tasks such as text classification and protein homology detection (Lodhi, Saunders, Shawe-Taylor, Cristianini, & Watkins, 2002). We proposed two novel string kernels learning methods: Meta Learning String Kernel SVMs and String Kernel MEBs to improve the effectiveness of the traditional string kernels SVMs.

Section 7.1 gives a brief review of string kernels and string kernels SVMs. Meta Learning String Kernel SVMs and String Kernel MEBs will be introduced in Section 7.2 and 7.3, respectively. Section 7.4 gives the conclusion for this chapter.

### 7.1 STRING KERNEL SVMs

Due to SVMs success in numerical pattern recognition with the help of kernel functionality, research focused on using SVMs with string kernels for string classification tasks. As Shawe-Taylor and Cristianini (2004) and Cortes, Haffner, and Mohri (2004) explain, apart from popularly used string kernels like Bag-of-Words and Edit Distance, one can also use kernels like n-gram...
or subsequence, which sum up substrings or subsequence in a document, as string kernels for a string classification task. Results from previous experiments (Shawe-Taylor & Cristianini, 2004; Cortes et al., 2004) have shown that SVM with string kernel functionality is able to recognise string patterns much more efficiently than other methods due to its ability to handle high dimensional data like string data without a decrease in performance decrease. This has made SVMs with string kernel functionality an ideal solution for DNA prediction, document classification, language recognition, image recognition and network anomaly detection scenarios. Even though string kernels are a subset of the kernel domain, they have their own properties and computational complexities associated with them. The following subsections explain about some widely used string kernels.

7.1.1 String Kernel methods

Further below we review the definition of mismatch kernels (Leslie, Eskin, Weston, & Noble, 2002) and present three new families of string kernels: Bag-of-Words kernel, Gap-Weighted Subsequences kernel, n-gram kernel and Levenshtein (or edit) distance kernel. In each case, the kernel is defined via an explicit feature map from the space of all finite sequences from an alphabet \( S \) to a vector space indexed by the set of \( k \)-length subsequences from \( S \). These models have been used in the computational biology literature in other contexts, in particular for sequence pattern discovery in DNA and protein sequences (Sagot, 1998) and probabilistic models for sequence evolution (Henikoff, 1992; Altschul, Gish, Miller, Myers, & Lipman, 1990).

Bag-of-Words Kernel

In document categorisation, a collection of documents is called a ‘corpus’, which consists of a set of predefined terms and is identified as a dictionary. A term or synonymously a word in the dictionary is any sequence of letters separated by punctuation or spaces. On the other hand, a bag, is defined as a set that allows repeated items. This definition of bag helps one to view a document as a bag of terms or bag of words (BOW). This allows a document
to be presented as a vector where each dimension is associated with a term in the dictionary. This representation \((\phi)\) is given as:

\[
\phi(d) = (tf(t_1, d), tf(t_2, d), ..., tf(t_n, d)) \in \mathbb{R}^n,
\]

here \(tf(t_i, d)\) is the frequency of the \(t_i^{th}\) term in document \(d\). Also, \(n\) is the space dimensionality and the size of the dictionary (Shawe-Taylor & Cristianini, 2004). Now, one can define a function \(k\) in this document space to compare the similarity between two documents \(d_1\) and \(d_2\):

\[
k_{bow}(d_1, d_2) = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f(t_i, t_j)^j, \tag{7.1}
\]

where, \(d_1\) has \(n_1\) terms and \(d_2\) has \(n_2\) terms. Also, \(f\) is defined as:

\[
f(t_x, t_y) = \begin{cases} 
\lambda_B^2 & \text{if } t_x = t_y, \lambda_B \in (0, 1) \\
0 & \text{otherwise} 
\end{cases} \tag{7.2}
\]

where \(t_x\) and \(t_y\) are two terms.

**N-gram Kernel**

In n-gram kernel, a string \(s\) is defined from alphabet \(\Sigma\) of \(|\Sigma|\) symbols, and is presented in a feature space \(F\), where each dimension is a string (Shawe-Taylor & Cristianini, 2004; Lodhi et al., 2002). Also, \(\Sigma^*\) represents the set of all strings and \(\Sigma^n\) represents the string set of length \(n\). Furthermore, \(ss'\) represents the concatenation of strings \(s\) and \(s'\). Now, the substrings: \(u, v_1, v_2\) of string \(s\), are defined such that:

\[
s = v_1uv_2,
\]

where, if \(v_1 = \varepsilon\) (\(\varepsilon\) is the empty string of 0 length) then, \(u\) is called to be the prefix of \(s\) and if \(v_2 = \varepsilon\), then \(u\) is called to be the suffix of \(s\). Now, a feature map \(\varphi\) is defined in feature space \(F\), with below embedding,

\[
\phi_u^n(s) = |\{(v_1, v_2) : s = v_1uv_2\}|, u \in \Sigma^n. \tag{7.3}
\]
The associated kernel is defined as:

$$\kappa_{ngram}(s, t) = \langle \phi^n(s), \phi^n(t) \rangle = \sum_{u \in \Sigma^n} \phi^n_u(s) \phi^n_u(t).$$ (7.4)

Also, the computational complexity of $n$-gram kernel is written as $O(n|s||t|)$ (Shawe-Taylor & Cristianini, 2004).

**Gap-Weighted Subsequences Kernel**

Let $s, t \in \Sigma^*$ be two strings from a finite alphabet $\Sigma$. Denote the lengths of the strings by $m = |s|$, $n = |t|$ and assume without loss of generality that $n \leq m$. Given a gap penalty $\lambda$, and an integer $p$, the gap-weighted subsequences (GWS) kernel is

$$\kappa_{GWS}(s, t) = \sum_{u \in \Sigma^p} \phi^p_u(s) \phi^p_u(t),$$ (7.5)

where $\phi^p_u(s) = \sum_{i:u=s(i)} \lambda^{l(i)}, u \in \Sigma^p$ is the embedding to the feature space of subsequences of length $p$.

**Levenshtein (or edit) distance kernel**

The levenshtein (or edit) distance associates to the difference between two strings. The difference refers to, number of insertions, substitutions and deletions required to transform string $s$ to string $t$. Assume that string $s$ is of length $n$ and string $t$ is of length $m$. For the string $s$ let $s(i)$ be its $i^{th}$ character. Also for two characters $a$ and $b$, we define $r$ as follows:

$$r(a, b) = \begin{cases} 
0 & \text{if } a=b \\
1 & \text{otherwise}
\end{cases}$$ (7.6)

An $(n+1)(m+1)$ array $d$, furnishes the levenshtein distance $L(s, t)$ between $s$ and $t$ via its $(n+1), (m+1)^{th}$ item. The calculation of $d(i, j)$ is done in a recursive manner by initially setting up $d(i, 0)$ and $d(0, j)$ as

$$d(i, j) = \min(d(i-1, j)+1, d(i, j-1)+1, d(i-1, j-1)+r(s(i), t(j))).$$ (7.7)
In this work, we used $e^{\lambda d(i,j)}$, rather than $d(i,j)$ for improved results.

7.1.2 Parameter Optimisation for String Kernels

String kernels also have different parameters like $\lambda_L$ in edit-distance kernel, $\lambda_B$ in bag-of-words kernel, sub-string size in n-gram kernel and subsequence size in fixed length subsequence kernel, which derives different kernel matrices. Hence, the parameter optimisation problem applies to string kernels as well. Adding to the complexity, the computation cost for strings can be quite expensive compared to numeric data, where most of the string kernels require an internal function to map strings to numbers (i.e. function $r$ in edit-distance and function $f$ in bag-of-words kernel that is explained in section 7.1.1). Also, if one observes the experimental results mentioned in (Rieck & Laskov, 2007), (Lodhi et al., 2002), (Sharma, Girolami, & Sventek, 2007) and (S. Sonnenburg, 2006) even the same string kernel requires different parameter combinations on different string datasets to yield good classification accuracies. This brings about the point that string dataset characteristics also need to be included in a string kernel SVM optimisation method.

7.2 A NOVEL METHOD OF META LEARNING FOR STRING CATEGORISATION

7.2.1 Meta Learning for String Categorisation - Answer for Question 4

This section discusses the motivation to use meta learning for string classification. It defines and explains meta learning, then elucidates the process of meta learning and its applications, followed by an explanation on meta-features. Then this chapter discusses the application of meta learning for text categorisation using a set of text meta-features, which are calculated from text data. After that it clearly demarcates the difference between text and string data, clarifying their ability to use some of the text meta-features for string classification at the end.
Meta Learning

As Giraud-Carrier, Vilalta, and Brazdil (2004) explain, meta learning is the process of acquiring and exploiting meta-knowledge through re-learning from meta-features. Re-learning, which is to maintain the learning algorithm unchanged or to modify it, helps the learning system to profit from repetitive use of similar tasks. It can be applied on a single learning system to optimise parameters, or on a set of algorithms to select the best algorithm for a given classification task (Vilalta, Giraud-Carrier, Brazdil, & Soares, 2004; Furdík, Paralič, & Tutoky, 2008; Brazdil, Soares, & Da Costa, 2003; Giraud-Carrier et al., 2004). The process of re-learning requires a set of domain specific characteristics so called ‘meta-features’ to evaluate the performance of an algorithm or algorithms (Brazdil et al., 2003). In practice, meta learning is used to select the best algorithm for a text classification (Lam & Lai, 2001; Furdík et al., 2008), predict optimum parameters for kernels in SVMs (Soares, Brazdil, & Kuba, 2004), and to optimise neural networks (Kord et al., 2010).

Apart from parameter optimisation, meta learning on a single learning system is used to evolve the architecture of the learning system via experience, such as evolving a decision tree using past experience (Brazdil, Giraud-Carrier, Soares, & Vilalta, 2008) or to evolve a neural network considering past topology parameters (Kord et al., 2010). On the other hand, meta learning on a set of algorithms is used in situations such as algorithm ranking (Brazdil et al., 2003) and algorithm identification in text categorisation (Furdík et al., 2008). Sound meta features that effectively describe domain characteristics are required in both these systems (single learning and systems with multiple algorithms) (Brazdil et al., 2008; Giraud-Carrier et al., 2004).

Generally, there are three types of meta-features. Firstly, simple statistical and information-theoretic meta-features are calculated from the dataset, such as number of classes, number of features, degree of correlation between features, and average class entropy (Brazdil et al., 2008). Secondly, there are model based meta-features: which describe certain characteristics of the learning system, such as maximum number of nodes per feature in a decision tree, kernel width of the gaussian kernel, or maximum depth of a decision tree.
Thirdly, *landmark meta-features* describe the performance (i.e. accuracy, mean squared error) of a learning algorithm (Kord et al., 2010; Brazdil et al., 2008).

**Review of Meta Learning for Text Categorisation**

Lam and Lai (2001) explain nine text meta-features for text categorisation, later expanded by Furdič et al. (2008) in their work. Both studies use the text meta-features to build a meta model, which selects the best algorithm for a given document category in document categorisation.

**Text Meta-features**

The text meta-features elucidated by Lam and Lai (2001) and Furdič et al. (2008) for document categorisation are:

1. **TrainingInstancesPerCategory**: Number of positive training instances per category.
2. **TestingInstancesPerCategory**: Number of positive testing instances per category.
3. **AvgDocLenPerCategory**: The average document length of a category. The document length refers to the number of index terms in a document. The average is taken across all the positive documents within a category.
4. **AvgTermValPerCategory**: The average term weight of a document within a category. The average index term weight is taken for a single document and the average is then computed for all the documents in a category.
5. **AvgMaxTermValPerCategory**: The average maximum term weight of a document within a category. The maximum index term weights for individual documents are summed and the average is taken for a category.
6. **AvgMinTermValPerCategory**: The average minimum term weight of a document within a category. The minimum index term weights for individual documents are summed and average is taken for a category.
7. **AvgTermThrePerCategory**: The average number of terms above a term weight threshold for a given category. The ‘term weight threshold’ is set
7.2 A NOVEL METHOD OF META LEARNING FOR STRING CATEGORISATION

The number of index terms above the term weight threshold are summed for category, and the average is computed for all instances in the category.

8. **AvgTopInfoGainPerCategory**: The average information gain of the top \( n \) index terms of a category. The information gain of each individual index term is computed for each category and ranked. The average is taken across top \( n \) index terms with the highest information gain within a category.

9. **NumInfoGainThresPerCategory**: The number of index terms in a category, where the information gain value exceeds a globally specified threshold.

The above text meta-features explained by Lam and Lai (2001) and Furdik et al. (2008) extracts statistical and information-theoretic information from the dataset, and are later used to train a meta model, which identifies the most suitable algorithm for a given document category.

**Text Data versus String Data**

As the main focus of our research is to optimise string kernel SVMs, this section attempts to clearly define the difference between text and string data. According to Singhal (2001); De-Bie and Cristianini (2004); Shawe-Taylor and Cristianini (2004), a text dataset is a collection of *words*, where word or synonymously term is any sequence of letters separated by punctuation or spaces. On the other hand, a *string* is a finite sequence of symbols from an alphabet (Shawe-Taylor & Cristianini, 2004). This means even the word demarcation symbols like space and punctuation can be in a string.

If one is given a string and a text dataset, it is easy to categorise them both into their respective data types, considering their *term* separability (De-Bie & Cristianini, 2004). This can be illustrated more clearly using Figure 7.1 where there are three types of data: network data (Figure 7.1a), Reuters-21578 news data (Figure 7.1b), and spam data which consists of spam and non-spam emails (Figure 7.1c). While it is difficult to separate *terms* in both network data and spam data, *terms* are easily separable in Reuters-21578 data. This helps one
7.2 A NOVEL METHOD OF META LEARNING FOR STRING CATEGORIZATION

(a) Network Data

(b) Reuters-21578 Data

(c) Spam Data

Figure 7.1: String and Text Data: (a) Network traffic data produced by network applications. (b) Data from Reuters-21578 dataset. (c) Spam data which consists of ham and Spam E-mail messages (in every instance, the first character represents the class label followed by actual data)
to categorise both network data and spam as string data and Reuters-21578 as text data.

With this difference between string data and text data in mind, the next subsections explain our motivation to use text meta-features explained for string classification.

**Motivation for using Meta Learning for String Categorisation**

All text meta features described before (apart from \textit{TrainingInstancesPerCategory} and \textit{TestingInstancesPerCategory}) are computed using terms and their frequencies. In this way it is possible to represent a string dataset as a collection of terms and their frequencies, which helps to derive some string meta-features that are computed by terms and their frequencies. Now these string meta-features help to employ meta learning on string classification.

**7.2.2 Meta Learning for String Kernel SVM Optimisation**

**String Meta-features**

In order to use the meta-features as discussed in section 7.2.1 for string classification, the string dataset needs to be presented as terms and term frequencies. We accomplish this in a string dataset by using splitting characters: "+" : ":(){}[]\"" to split a string into set of terms or synonymously \textit{tokens}. This approach is referred as ‘tokenisation’ in the literature (Shawe-Taylor & Cristianini, 2004). To explain tokenisation, consider the highlighted string in Figure 7.2, which is from a network application that uses \textit{http} protocol. Using specified splitting characters, one can split the string into tokens: “\textit{akt} = 0;”, “\textit{r}”, “\textit{nvarbb};\textit{flash} = 0;”, “\textit{r}”, “\textit{ni}f”, “\textit{navigator}”, and “\textit{appV}”. Now a \textit{token frequency table} is generated, as shown in Table 7.1. This token-frequency information is used to compute the string meta-features explained in this section.

The main difference between the text meta features discussed in section 7.2.1 and string meta-features explained here, is that the \textit{text meta-features} are calculated for a text category in a text dataset, but, the \textit{string meta-features} are calculated for the entire string dataset. Out of nine text meta-features discussed in the section 7.2.1, seven are considered in deriving these string meta-features.
7.2 A NOVEL METHOD OF META LEARNING FOR STRING CATEGORIZATION

Figure 7.2: String Data: Network traffic data produced by network applications (in every instance, the first character represents the class label of the network application, followed by actual network traffic data)

Assume a string dataset that has \( n \) number of instances. The seven string meta-features are:

1. **AvgInstanceLen**: The average instance length of the dataset. The instance length refers to number of tokens in an instance. The average is taken across all the instances. If \( i^{th} \) instance has \( N_i \) tokens, then the average instance length for that dataset is \( \frac{\sum_{i=1}^{n} N_i}{n} \).

2. **AvgTokenVal**: The average token weight of an instance across a string dataset. Initially, the token weight is calculated for each token and the average is computed for single instance. Then, the average token weights for each instance are summed and the average is computed for all the instances.

If there are \( m \) unique tokens in \( i^{th} \) instance, the average token weight for a string dataset is written as:

\[
\text{Average token weight of the string dataset} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} TW(j, i)}{mn},
\]
7.2 A NOVEL METHOD OF META LEARNING FOR STRING CATEGORISATION

<table>
<thead>
<tr>
<th>Token</th>
<th>Token Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>akt = 0;</td>
<td>1</td>
</tr>
<tr>
<td>r</td>
<td>2</td>
</tr>
<tr>
<td>nvarbb, sflash = 0;</td>
<td>1</td>
</tr>
<tr>
<td>nif</td>
<td>1</td>
</tr>
<tr>
<td>navigator</td>
<td>1</td>
</tr>
<tr>
<td>appV</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 7.1: Token Frequency Table: Tokens and their frequencies for highlighted string in Figure 7.2

(7.8)

where $TW(j, i)$ is the token weight of $j^{th}$ token in $i^{th}$ instance. According to the interpretation given by Hersh (2008) of the term weight, the $TW(j, i)$ can be written as:

$$TW(j, i) = TF(j, i) \times IDF(j),$$

(7.9)

where $IDF(j)$ is the inverse document frequency of $j^{th}$ token, and $TF(j, i)$ is the frequency of $j^{th}$ token in instance $i$. Furthermore, according to (Hersh, 2008), the $IDF(j)$ is computed as:

$$IDF(j) = \log \frac{n}{TF(j)} + 1,$$

(7.10)

where $TF(j)$ is the frequency of the $j^{th}$ token in the dataset. Now, considering (7.9) and (7.10), equation (7.8) is rewritten as:

$$\text{Average token weight of the string dataset} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} TF(j) \left(\log \frac{n}{TF(j)} + 1\right)}{mn}$$

(7.11)

3. **AvgMaxTokenVal**: The average maximum token weight of an instance across a string dataset. Maximum token weights of an instance are summed and the average is taken across all instances.
4. **AvgMinTokenVal**: The average minimum token weight of an instance across a given string dataset. Minimum token weights of an instance are summed and the average is taken across all instances.

5. **AvgTokenThre**: The average number of tokens above a token weight threshold for a given string dataset. The token weight threshold is set globally. The number of tokens where the token weight is above the threshold are summed and the average is taken across all instances.

6. **AvgTopInfoGain**: The average information gain of the top \( r \) tokens in the string dataset. The information gain of each individual token is computed for each instance and ranked. Then, the average is taken across top \( r \) terms with highest information gain.

7. **NumInfoGainThres**: The average number of tokens in an instance where the information gain value exceeds a globally specified threshold.

**Meta Learning for String Classification**

The mentioned string meta-features help to employ meta learning on a string classification. The principle of using meta learning for string classification is discussed in section 7.2.2. A novel string kernel SVM optimisation method is elucidated in section 7.2.2.

![Figure 7.3: String meta-feature generation process](image)
The procedure to employ meta learning for string classification

**Meta Learning for String Classification: Principle**

Consider a string dataset $D$, which is represented as a vector in *token-frequency space* $\Omega$, where each dimension in $\Omega$ is associated with one *token*. Now, the dataset $D$ is represented via function $\omega$ in this new $\Omega$ token-frequency space:

$$\omega(D) = (TF(t_1, D), TF(t_2, D), ..., TF(t_N, D)) \in \Omega,$$

(7.12)

where $TF(t_j, D)$ is the token frequency of $j^{th}$ token in the string dataset $D$, and $N$ is the number of unique tokens in the dataset. Now one can derive a function $f_p : \Omega \rightarrow \mathbb{R}$:

$$f_p(D) = f'_{p,D}$$

(7.13)

where $f'_{p,D}$ represents the value for the $p^{th}$ string-meta feature for $D$. For the string dataset $D$, there are $p'$ finite meta-features, where all string meta-features $f_p(D)$ ($p = 1, 2, 3, ..., p'$) are well defined. This string meta-feature generation process is shown in Figure 7.3.

Using the above discussed string meta-features, Figure 7.4 explains the principle of meta learning for string classification. Assume there is a string dataset pool $L$ with $l'$ datasets, where, each string dataset $D_l$ ($D_l \in L, l = 1, ..., l'$) is again subdivided into unique $D_{lTR}$ (training) and $D_{lTS}$ (testing) datasets, which creates training ($L_{TR}$) and testing ($L_{TS}$) dataset pools. The string meta-feature $f'_{p,D_{lTR}}$ is computed for dataset $D_{lTR}$. Also, for $D_{lTR}$, the machine
learning algorithm $LA$ with parameter combination $c$, generates $Y_{D_{ITR},c}$ classification accuracy. These computed string meta-features ($f'_{p,D_{ITR}}$), parameter combinations ($c$) and accuracy information ($Y_{D_{ITR},c}$) generate a meta model via regression, which is able to predict the classification accuracy for a new string dataset, given the computed string meta-features and the parameter combination. Hence, for a new string dataset $D_{ITS}$, the meta model predicts the accuracy $Y_{D_{ITS},c'}$ for parameter combination $c'$ by computing string-meta features $f'_{p,D_{ITS}}$.

**Meta Learning for String Kernel SVM Optimisation: Algorithm**

According to the principle introduced in this section the built meta model is able to predict the string classification accuracy for a machine learning algorithm on a novel string dataset, using computed string meta-features. This section explains the procedure to use this principle (meta learning for string classification) to optimise string kernel SVMs, which is shown in Algorithm 8.

Algorithm 8 explains the procedure of using meta learning to optimise string kernel $SK$ with SVM. The proposed algorithm uses training string dataset pool $L_{TR}$, testing string dataset pool $L_{TS}$, training parameter pool $C$ and testing parameter pool $C'$ as inputs. Also, SVM with string kernel $SK$ is set as the learning algorithm ($LA$). Initially, a meta model is built using meta features calculated for each dataset in $L_{TR}$ with accuracy information obtained for each parameter combination in $C$ via n-fold cross validation. Then, for each new string dataset in $L_{TS}$, the built meta model predicts the classification accuracy for each combination in $C'$. The combination ($c_l$) which yields the highest accuracy is presented as the optimum parameter combination for dataset $D_{ITS}$.

7.3 A NOVEL FAST STRING CLASSIFICATION METHOD

7.3.1 Motivation

Despite of the high performance of meta learning string kernels SVMs on classification accuracy, the computational cost of the learning process is huge. Alternatively, we developed another novel method string kernels $MEB$ which
input:
\[ L_{TR} = \text{Training String Dataset Pool} \]
\[ L_{TS} = \text{Testing String Dataset Pool} \]
\[ C = \text{Parameter Combination Pool for Training (} c \in C) \]
\[ C' = \text{Parameter Combination Pool for Testing (} c' \in C') \]
\[ LA = \text{SVM with String Kernel } SK \]

output: Parameter combination \( \hat{c} \) which yields the best accuracy for string dataset \( D_{lTS} \)

for \( l \leftarrow 1 \) to \( l' \) do
  Pick \( D_{lTR} \) from \( L_{TR} \)
  for \( p \leftarrow 1 \) to \( p' \) do
    Compute \( f'_{p,D_{lTR}} \)
  end
  repeat
    Pick a parameter combination \( c \) from \( C \)
    Do 10-fold cross validation on \( D_{lTR} \), using \( LA \) with parameter combination \( c \) which yields \( Y_{D_{lTR},c} \) accuracy
    until no more parameter combinations in \( C \)
  end

Build a regression model (meta model) using \( f'_{p,D_{lTR}}, c, \) and \( Y_{D_{lTR},c} \)

for \( l \leftarrow 1 \) to \( l' \) do
  Pick \( D_{lTS} \) from \( L_{TS} \)
  for \( p \leftarrow 1 \) to \( p' \) do
    Compute \( f'_{p,D_{lTS}} \)
  end
  repeat
    Pick a parameter combination \( c' \) from \( C' \)
    Predict accuracy \( Y_{D_{lTS},c'} \) for \( LA \) with parameter combination \( c' \)
    using build meta model
    if \( Y_{D_{lTS},c'} \) is maximum then
      \( \hat{c}_l = c' \)
    end
  until no more parameter combinations in \( C \)
end

Algorithm 8: The proposed meta learning algorithm for string kernel SVM optimisation
extends original MEB/CVM algorithm with string kernels methods. As we discussed in chapter 4 the computational complexity of MEB is much smaller than SVMs, the string kernel learning time for string kernels MEB is shorter.

7.3.2 String Kernels MEB Algorithm Description

Given a data matrix \( X = \{x_1, x_2, \ldots, x_l\} \), MEB over \( X \) is modelled as the smallest hypersphere \( B_X = \{c, r, X\} \), where \( c, r \) are the centre and radius of \( B_X \), respectively. The \( B_X \) is calculated by solving the optimisation problem:

\[
\begin{align*}
\min_{c, r} & \quad r^2 \\
\text{subject to } & \quad ||c - \phi(x_i)||^2 = (\phi(x_i) - c)'(\phi(x_i) - c) \leq r^2 \\
& \quad i = 1, 2, \ldots, l
\end{align*}
\]  

(7.14)

which can be solved by introducing a Lagrange multiplier \( \alpha_i \geq 0 \) for each constraint

\[
L(c, r, \alpha) = r^2 + \sum_{i=1}^l \alpha_i \left[ ||\phi(x_i)||^2 - r^2 \right] = \ldots = \sum_{i=1}^l \alpha_i \kappa(x_i, x_i) - \sum_{i,j=1}^l \alpha_i \alpha_j \kappa(x_i, x_j)
\]  

(7.15)

Given a set of string \( S = s_1, s_2, \ldots, s_l \) while is defined from alphabet \( \Sigma \) of \( |\Sigma| \) symbols. MEB \( B_S \) is able to be obtained by transferring string kernels (e.g. bag-of-words) of \( S \) into equation 7.15 as:

\[
L(c, r, \alpha) = \sum_{i=1}^l \alpha_i \kappa_{bow}(s_i, s_i) - \sum_{i,j=1}^l \alpha_i \alpha_j \kappa_{bow}(s_i, s_j).
\]  

(7.16)

The optimised Lagrange parameter \( \alpha^* \) is calculated by solving the problem as:

\[
\begin{align*}
\min_{\alpha} & \quad \sum_{i=1}^l \alpha_i \kappa_{bow}(s_i, s_i) - \sum_{i,j=1}^l \alpha_i \alpha_j \kappa_{bow}(s_i, s_j) \\
\text{subject to } & \quad \sum_{i=1}^l \alpha_i = 1, \text{ and } \alpha_i \geq 0, \ i = 1, 2, \ldots, l
\end{align*}
\]  

(7.17)

According to the Karush.Kuhn.Tucker (KKT) theorem that the Lagrange parameters can be non-zero only if the corresponding inequality constraint is
an equality at the solution, only these $S_{sv} \subset S$ that lie on the surface of the optimal hypersphere have their corresponding $\alpha_i^*$ non-zero. Thus, we treat $S_{sv}$ as the support vectors of $S$.

### 7.3.3 Online String Kernels

In this stage, Meta Learning String Kernel SVMs and String Kernel MEBs can only work in batch learning mode. However, the combination of string kernel with MEB gives a new direction of string kernel online learning mode. In the future, we can implement the string kernel online learning by taking a suitable online kernel learning method and replace the kernel matrices with the string kernels matrices.

### 7.4 SUMMARY

This chapter gives a brief introduction to string kernels SVM and presents two novel string kernels learning methods: Meta Learning String Kernel SVMs and String Kernel MEBs. Although these two methods are for batch learning only, the combination of batch learning methods and online learning methods is also proper for stream data learning. These two methods are able to outperform traditional string kernels methods on both accuracy and efficiency. The evaluation of these two methods is presented in chapter 9.
THE APPLICATION OF HIERARCHICAL CORE VECTOR MACHINES TO SOLVE NETWORK INTRUSION DETECTION

This chapter presents a case study of network intrusion detection. We use RCV1-V2 (Lewis, Yang, Rose, & Li, 2004) for a simulation of network strings examination as RCV1-V2 gives a one level multi-label (i.e., not hierarchical multi-label) text classification problem. KDD’99 (KDD99, 1999) poses a hierarchical multi-label Internet security problem, we thus use KDD’99 as a comprehensive network security real application case study.

8.1 BACKGROUND

Upon the explosive growth of number of Internet users, malicious activity such as denial of service attacks, port scans or even attempts to crack into computers by monitoring network traffic are also increasing. These malicious activities on the Internet, also known as network intrusions, trouble the Internet users and cause Internet security threats. Network intrusion detection systems (NIDS) are developed for safeguarding the users and their systems from any threats of Internet intrusion. As a preventive measure, existing NIDS mostly employ two popular intrusion detection methods: (1) the white and black list methods filter spam through network ID/address validity verification (Chirita, Diederich, & Nejdl, 2005); (2) the Intrusion signature recognition methods detect any intrusions by comparing network data with a predetermined attack signature (Ye & Chen, 2001). However, both methods have proved brittle to any small alteration of attack (Kofcz, Chowdhury, & Alspector, n.d.). This requires ordi-
nary NIDS (e.g. a firewall system for a personal computer) to be updated in a regular time interval.

As an alternative solution, machine learning methods solve the above difficulty confronted by existing NIDS, because the attack signature/filter are learned dynamically from the streaming network data. In this field, Mukkamala and Sung (2003) implemented support vector machine (SVM) method on the KDD Cup dataset and obtained an overall 99% classification accuracy. Frank and Mda-c (1994) took decision trees as the most suitable classification method for intrusion categorisation. Panda and Patra (2007) compared the performance of Naive Bayes with the neural network approach, and authenticated the suitability of Naive Bayes for intrusion detection modelling. Despite that, machine learning for network intrusion detection still have underlying issues related to accurate network threats authentication. With the vulnerability of present-day software and protocols combined with the increasing sophistication of attacks, network-based attacks are on the rise (Staff, 2005), which has made an astonishing revenue loss every year. The 2005 annual computer crime and security survey (CSI & FBI, 2005) reported that the financial losses incurred by the respondent companies due to network attacks/intrusions were US $130 million.

In terms of using machine learning for intrusion detection, existing methods perform intrusion detection by grouping any network traffic into several major clusters. This in practice often reduces the detection rate/efficiency because the detailed information of network traffic, including attack messages, is ignored. For example, spam is simply divided into several major categories, such as junk mail, IM spam, TXT spam etc. It could be obviously advantageous to spam detection, if the junk mail could be dividable into unsolicited bulk e-mail (UBE) and unsolicited commercial e-mail (UCE). Thus, it is important for network intrusion detection that the whole picture of network traffic/intrusion is presented as a hierarchy as seen in Fig.5.1, and it is necessary to do multiple labelling of the network traffic/intrusions.
8.1.1 Multi-label Intrusions

Intrusion detection is formatted into a hierarchical multi-label classification (HMC) problem. A high-speed hierarchal minimum enclosing balls (HCVM) algorithm is proposed for multi labelling network intrusions in a MEB/CVM structure where MEBs/CVMs are separating, encompassing, and overlapping with each other. A minimum enclosing ball (MEB/CVM) computes a hypersphere of minimum radius which encloses a specific type of network traffic/intrusion. Given an unlabelled network message, the proposed HCVM seeks a MEB/CVM enclosing the message, and multi-labels it by the MEB/CVM’s position and its ancestor relationship in the HCVM. In the experiments, we have authenticated the proposed HCVMs classification proficiency and computational efficiency. For the HMC from KDD’99 dataset, the proposed HCVM exhibits an outstanding classification accuracy for U2R and R2L attack types. Furthermore, the capability of HCVM for handling single-level multi-label data is also demonstrated in another experiment with the RCV1-V2 dataset.

8.2 EXPERIMENTS AND DISCUSSION

For testing the capability of the proposed HCVM for multi-label, in particular for hierarchical multi-label classification, we examined the developed HCVM on the RCV1-V2 text dataset (Lewis et al., 2004), and the benchmark KDD’99 dataset (KDD99, 1999), respectively.

8.2.1 Experimental setup

The proposed HCVM is implemented in MATLAB version (7.6.0), on a 1.86Hz Intel Core 2 machine with 2GB RAM. In our experiments, we use a non-linear Gaussian kernel $k(x, y) = \exp(-\|x - y\|^2 / \beta)$ with $\beta = \frac{1}{m^2} \sum_{i,j=1}^{m} \|x_i - y_j\|^2$. We set the $\epsilon$ as $1e-6$ for $(1 + \epsilon)$-approximation of MEB/CVM, and the Gaussian kernel parameter $\beta$ as 1000.

Parameter $n$ in Algorithm 6 is a positive integer, representing the number of the nearest MEBs/CVMs to the input instance. Thus, $n$ is greater than or equal
to the maximum number of multi-labels for a single instance. For example, in the case of the RCV1-V2 dataset, \( n \) takes a value of 11. It is worth noting that \( n \) in practice is more often determined by cross-validation tests over the training data. In the case of KDD’99 dataset, \( n \) is set to 3 because the highest 93.6\% accuracy is achieved when \( n \) equals 3 in cross-validation tests, as shown in Fig. 8.1.

![Figure 8.1: Cross validation tests of Algorithm 6 for parameter \( n \) determination on KDD’99 dataset](image)

8.2.2 Simulation of Network Strings Examination: RCV1-V2 Dataset

The RCV1-V2 data set collected by Lewis et al. (2004) contains newswire stories from Reuters Ltd. RCV1-V2 and was pre-processed by several schemes, including removing stopping words, stemming, transforming the documents to vectors with TF-IDF format and cosine normalisation, etc. Among the three category sets (Topics, Industries, and Regions) of RCV1-V2, we consider, analogous to the previous works done by Lewis et al. (2004), only the Topics category set, with the statistics of training and testing dataset is given in table 8.1.
As a simulating network string examination, we implement Algorithm 5 for HCVM training, and Algorithm 6 for testing the generated HCVM model. Algorithm 6 checks the $n$ nearest MEBs/CVMs of the HCVM, and labels each new instance as one of the classes of the MEBs/CVMs enclosing the instance.

<table>
<thead>
<tr>
<th>Data set</th>
<th>#inst.</th>
<th>#feat.</th>
<th>#label</th>
<th>Label size frequency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCV1-V2 train</td>
<td>23,149</td>
<td>47,236</td>
<td>101</td>
<td>12.3% 29.5% 35.7% 10.8% 11.7%</td>
</tr>
<tr>
<td>RCV1-V2 test</td>
<td>781,265</td>
<td>47,236</td>
<td>103</td>
<td>12.3% 25.4% 45.3% 10.9% 15.3%</td>
</tr>
</tbody>
</table>

Table 8.1: Statistics of RCV1-V2 dataset

For performance evaluation, we compare HCVM with SVM (Chang & Lin, 2001) on one of the most used performance measures for information retrieval (Hripcsak & Rothschild, 2005), F-measure, which is calculated as the harmonic mean of precision ($P$) and recall ($R$):

$$F - measure = \frac{2PR}{P + R}. \quad (8.1)$$

where

$$P = \frac{\sum_{i=1}^{l} \hat{y}_j^i y_i^j}{\sum_{i=1}^{l} \hat{y}_j^i}, \quad R = \frac{\sum_{i=1}^{l} \hat{y}_j^i y_i^j}{\sum_{i=1}^{l} y_i^j}, \quad (8.2)$$

$y_i^j$ is the true label of $i^{th}$ instance for class $j$ from total $l$ testing instances and $\hat{y}_j^i$ is the predicted label. In our experiments, we use two approaches: macro-average F-measure and micro-average F-measure (Tague-Sutcliffe, 1997) which extended the F-measure from single-label to multi-label. The macro-average F-measure for a total of $d$ labels is obtained from the unweighted mean of the label F-measures and can be calculated as,

$$F = \frac{1}{d} \sum_{j=1}^{d} \frac{2 \sum_{i=1}^{l} \hat{y}_j^i y_i^j}{\sum_{i=1}^{l} \hat{y}_j^i + \sum_{i=1}^{l} y_i^j}. \quad (8.3)$$
The micro-average F-measure, which considers predictions from all instances together and calculate the F-measure across all labels,

$$ micro - F = \frac{2 \sum_{i=1}^{d} \sum_{i=1}^{l} \hat{y}_{j}^{i} y_{j}}{\sum_{i=1}^{d} \sum_{i=1}^{l} \hat{y}_{j}^{i} + \sum_{i=1}^{d} \sum_{i=1}^{l} y_{j}}. $$

The additional measurement we used for performance evaluation is AUC (the area under the Receiver Operating Characteristic (ROC) curve). AUC reflects mainly the ranking quality of predictions. A perfect classifier has an expected AUC score equal to 1. A detailed description of the AUC and ROC graphs can be found in (Fawcett, 2003).

<table>
<thead>
<tr>
<th>Methods</th>
<th>F-Measure</th>
<th>Macro-Average F-Measure</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>0.76</td>
<td>0.51</td>
<td>0.85</td>
</tr>
<tr>
<td>HCVM</td>
<td>0.86</td>
<td>0.71</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Table 8.2: AUC, F-Measure and Macro F-Measure comparison

Table 8.2 presents the comparison result. The proposed HCVM performs 10% better than SVM on Macro F-measure, 20% better on Micro F-measure, and 6% better on AUC under the condition that the complete dataset is used for training. Fig. 8.2 gives further comparison on Macro F-measure and Micro F-measure, when the sizes of training sets vary from 10% to 100%. As seen, the proposed HCVM consistently and significantly outperforms the SVM by 10% – 30% on both Macro F-measure and Micro F-measure despite the change in the size of training set. Fig. 8.3 gives the ROC comparison, where the AUC area of HCVM is seen compassing the AUC area of SVM, which demonstrates again that the proposed HCVM provides crisply better quality of predictions than SVM at any size of training set.

The poor performance of SVM can be explained in part by the fact that the SVM hinge loss is not well suited for unbalanced data (Zhang & Oles, 2000). However, HCVM is the aggregation of MEBs/CVMs on multi-label hierarchy. The advantages of HCVM on class-imbalanced problem are: (1) single MEB/CVM is capable of modelling whatever skewed class (i.e. a class
8.2 EXPERIMENTS AND DISCUSSION

Figure 8.2: (a) and (b) plot the Micro-Average F-measure and Macro-Average F-measure of different number of training dataset for both SVM and HCVM respectively.

Figure 8.3: ROC graphs with AUC results of 10 chunks training dataset with same size. The filled areas in (a) and (b) correspond to the AUC for SVM and HCVM respectively.

with fewer samples than other classes) more accurately; and (2) HCVM is convenient to allocate more computing power (i.e. more MEBs/CVMs) for the classification of the skewed-class.

8.2.3 Internet Security Application: KDD’99 Datasets

KDD’99

KDD Cup 1999 dataset (KDD99, 1999) contains 5 million Internet connection records. Each record encodes 41 connection features including 1 class label, 34 continuous features, and 7 symbolic features. The class label identifies one of 22 network connection types including normal, buffer_overflow, guess_password etc. All connection features are assumed to be conditionally...
independent. Apart from the normal connection type, the rest of 21 attack types are associated with 4 major categories of attack, they are:

1. **DOS**, denial of service, e.g. back;
2. **R2L**, unauthorised access from a remote machine, e.g. guessing password;
3. **U2R**, unauthorised access to local superuser (root) privileges, e.g. buffer overflow;
4. **PROBE**, information gathering, e.g. port sweep.

**Data Preprocessing**

For data preprocessing, we replace the original label for each instance with a set of numerical labels by the following rules: the major attack categories stay at the head followed by its subcategories. For example, 'back', as the first subcategory of DOS attack, is represented as (1,1), and the second subcategory 'land', as (1,2). In this way, all labels can be transformed into a numerical hierarchical structure as shown in Fig. 8.4. Additionally, we normalise every continuous feature into \([0, 1]\), and encode every symbolic feature to binary digit.

![Figure 8.4: KDD99 data classification label replacement](image-url)

Figure 8.4: KDD99 data classification label replacement
Comparison Results

For constructing HCVM, we compute MEBs/CVMs and core sets for 21 subcategories and 4 major categories of attacks, respectively. For each category and subcategory, we use the instances of the core set as the core vectors. As a result, we obtain 25 core vector sets in total corresponding to 4 major categories plus 21 subcategories of attacks. Table 8.3 and Figure 8.5 present the comparison of our classification results to the results achieved by Bernhard who is the winner of KDD’99 cup (Elkan, 2000).

<table>
<thead>
<tr>
<th>Actual VS Predicted</th>
<th>Actual Normal</th>
<th>Actual DOS</th>
<th>Actual U2R</th>
<th>Actual R2L</th>
<th>Actual Probe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernhard predicted Normal</td>
<td>60262</td>
<td>5299</td>
<td>168</td>
<td>14527</td>
<td>511</td>
</tr>
<tr>
<td>HCVM predicted Normal</td>
<td>920431</td>
<td>36818</td>
<td>9845</td>
<td>2603</td>
<td>2084</td>
</tr>
<tr>
<td>Bernhard predicted DOS</td>
<td>78</td>
<td>223226</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>HCVM predicted DOS</td>
<td>153064</td>
<td>3619301</td>
<td>10341</td>
<td>14732</td>
<td>85932</td>
</tr>
<tr>
<td>Bernhard predicted U2R</td>
<td>4</td>
<td>0</td>
<td>30</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>HCVM predicted U2R</td>
<td>6</td>
<td>3</td>
<td>43</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bernhard predicted R2L</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>1360</td>
<td>0</td>
</tr>
<tr>
<td>HCVM predicted R2L</td>
<td>321</td>
<td>193</td>
<td>44</td>
<td>517</td>
<td>51</td>
</tr>
<tr>
<td>Bernhard predicted Probe</td>
<td>243</td>
<td>1328</td>
<td>20</td>
<td>294</td>
<td>3471</td>
</tr>
<tr>
<td>HCVM predicted Probe</td>
<td>2843</td>
<td>2104</td>
<td>412</td>
<td>506</td>
<td>35237</td>
</tr>
<tr>
<td>Bernhard total accuracy</td>
<td>99.5%</td>
<td>97.1%</td>
<td>13.2%</td>
<td>8.4%</td>
<td>83.3%</td>
</tr>
<tr>
<td>HCVM total accuracy</td>
<td>94.6%</td>
<td>93.2%</td>
<td>82.7%</td>
<td>45.9%</td>
<td>85.7%</td>
</tr>
</tbody>
</table>

Table 8.3: Compare the classification accuracy with (Bernhard 1999) who is the winner of KDD’99 cup

As seen from the table, Bernhard (1999) achieved an extremely high classification accuracy of 99.5% on normal connect type, however U2R and R2L showed poor classification performance, with none of them exceeding 15%, because of their class size being smaller than the other classes. Although the overall classification accuracy of the proposed HCVM is slightly lower than that of the Bernhard’s method for normal connection type and DOS type attacks, the proposed HCVM outperforms Bernhard’s method on the classification of 3 most important classes. HCVM particularly increases the classification accuracy of U2R and R2L by 70% and 35%, respectively. This demonstrates the advantage of the proposed HCVM, which enables a more accurate approximation of class information, even for a very small dataset.
To evaluate the efficiency of the proposed methods we computed the number of CPU time, support vectors and test error rate of HCVM and SVM with different sizes of training datasets. As seen in Fig. 8.6a, the time cost of HCVM increases more slowly than the SVM. In addition, Fig. 8.6b shows the number of support vectors found by HCVM stays constant regardless of the training dataset size. For SVM, the number of support vectors increases proportionally to the dataset size. On comparison of test error rate, Fig. 8.6c indicates that HCVM constantly provides lower error rate than SVM, specially when the training dataset size is small.

8.3 SUMMARY

We have evaluated the proposed HCVM method for network intrusion detection applications. In our experiments, we have implemented the HCVM on benchmark datasets from UCI archives such as KDD’99 and RCV1-V2. The experimental results show that the proposed HCVM is clearly more efficient and accurate than the traditional methods, especially for the U2R and R2L problem of the KDD’99 dataset.

Next chapter presents a case study for face membership authentication.
Figure 8.6: (a) CPU time required by HCVM and SVM, (b) number of support vectors created by HCVM and SVM with different size of training dataset, respectively. (c) the error rate variation against the size of training dataset.
Chapter 9

THE APPLICATION OF DYNAMIC EVOLVING CORE VECTOR MACHINES TO SOLVE FACE MEMBERSHIP AUTHENTICATION PROBLEM

This chapter presents a case study of face membership authentication (FMA). To evaluate the class factorisation ability of DE-CVM (Y. Chen, Pang, & Kasabov, 2010), we study the FMA problem (Pang, Kim, & Bang, 2005) which is to distinguish the membership class from the non-membership in a total group through a binary class classification. FMA involves different levels of class overlapping which include the most discriminative class characteristics (Garcia, Alejo, Sinchez, Sotoca, & Mollineda, 2008) because class overlapping increases while the size of the membership group is close to the size of non-membership group. The size of the membership group can be dynamically changed which makes class characteristic of membership and non-membership manually adjustable. The smaller the size of membership group, the less discriminative class characteristics are involved.

Section 9.1 explains the background of face membership authentication, Section 9.2 presents dataset description and data pre-processing. The experiment results for DE-CVM are presented in Section 9.3. Section 9.4 gives the conclusion.

9.1 BACKGROUND

Membership authentication is a typical problem in digital security schemes (Pang, Kim, & Bang, 2003). The problem can be generally depicted as follows. Consider a certain group of people $G$ with $N$ members, which is the universal
set. If there exists an arbitrary subgroup \( M \) such that \( M \subseteq G \) and \( |M| < N/2 \), then it is a membership group, and the remaining people \( \bar{M} = G - M \) make a non-membership group. Thus, the objective of membership authentication is to distinguish the membership \( M \) class from the non-membership class \( \bar{M} \) in the group. Since anonymity is an essential feature of digital security schemes (M.-H. Yang, Ahuja, & Kriegman, 2000), this would require membership authentication to allow changing dynamically the size of the membership group and the members in that group. Therefore, unlike all previous types of works on face recognition for security, where identification of a given face image is needed, dynamic membership authentication requires to authenticate an individuals membership without revealing the individuals identity and without restricting the group size and/or the members of the group. For example, for security or access control, the permission such as the right to enter an important building is often assigned to many individuals. To get the permission, it is required that members of the group be distinguished from nonmembers, while the members need not be distinguished from one another due to privacy concerns (Schechter, Parnell, & Hartemink, 1999).

For dynamic face membership authentication, Xie, Xu, and Hundt (2001) introduced a verification system, in which they authenticated face memberships by combining a face recognition method using template matching and a face verification using single support vector machine (SVM) classifier. Since the system is not entirely independent from face identification, it is not a real dynamic membership authentication system according to our definition. Pang et al. (2003) introduced an SVM ensemble method for membership authentication in dynamic face groups. It was a novel membership authentication method for two main reasons. First, it performed the membership authentication in terms of binary classification without revealing the individuals identity, \( i.e. \) it was only concerned with whether a member was included in a membership group or not. A powerful SVM ensemble combining several binary SVM classifiers was introduced for supporting this property. Second, it performed dynamic membership authentication without restricting the group size and/or the members of the group, \( i.e. \) the membership authentication environments could be changing dynamically. An effective face representation using an eigenfeature fusing technique was introduced to support this requirement.
However, Pang et al. (2003) also found that the SVM ensemble method could only remain stable for a membership group whose size is less than 45 persons (16.6% of total group). As the membership group size increases, its membership authentication performance becomes poorer and very unstable. Furthermore, when the size of the membership group becomes similar to the size of the non-membership group, it is almost impossible to obtain a satisfactory authentication performance. This is due to a complicated mixed data distribution among the membership and non-membership face images, as it is very difficult to discriminate such data in terms of only one classifier even if its classification performance is powerful.

9.2 Image Data Pre-processing

In pre-processing, a data splitting procedure divides the data iteratively, and a number of CVM classifiers follow each step of the data partition. Here, by using locally linear embedding (LLE) (Pang et al., 2005) dimensionality reduction theory, we compressed member faces and nonmember faces as a set of LLE eigenfaces which together characterise the variation between all the member face images and nonmember face images, respectively. We partitioned the training set by a membership-based clustering with the membership eigenfaces and non-membership eigenfaces as two cluster centres.

9.2.1 LLE Eigenface

Compared with the linear dimensionality reduction method PCA (Pang et al., 2003), LLE is a method for nonlinear dimensionality reduction introduced by Roweis and Saul (2000). This method recovers global nonlinear structure from locally linear fits. It attempts to preserve as much as possible the local neighbourhood of each object, while preserving the global distances to the rest of the objects. These properties of the method might not be of benefit for data classification, but they definitely imply a better clustering of the data. Here, we represent the whole set of face images in the membership group (or non-
membership group) as a set of membership (or non-membership) eigenfaces, which we obtained from the LLE eigenface technique as explained below.

A given face feature dataset consists of \( N \) real-valued vectors \( \mathbf{x} \). Each vector \( \mathbf{x} \) is a high-dimensional vector with dimensionality \( D \), and \( y_i \) is the low-dimensional vector embedded in \( x_i \) with embedding dimensionality \( d \), where \( D \gg d \). The computation of LLE eigenface involves an optimal embedding procedure that reduces a vector from high-dimensional data \( x \) to low-dimensional data \( y_i \) by minimising the following cost function:

\[
\Phi(y) = \sum_{i} \left| y_i - \sum_{j} W_{ij} y_j \right|^2.
\] (9.1)

This procedure consists of three steps:

Step 1. Compute the neighbours of each data point \( x_i \) by computing pairwise distances and finding neighbours. In the simplest formulation of LLE, one can identify \( K \) nearest neighbours per data point by measuring the Euclidean distance.

Step 2. Compute the weights \( W_{ij} \) that best reconstruct each data point \( x_i \) from its neighbours, minimising the following cost function by constrained linear fits:

\[
\epsilon(W) = \sum_{i} \left| x_i - \sum_{j} W_{ij} x_j \right|^2.
\] (9.2)

Step 3. Compute the matrix \( M \) in terms of the previous weights computation

\[
M = (I - W)^T (I - W).
\] (9.3)

where \( I \) is the \( N \times N \) identity matrix.

Note that the bottom \( d + 1 \) eigenvectors of the matrix \( M \) are corresponding to its smallest \( d + 1 \) eigenvalues. Thus, for fixed weights vectors \( W_{ij} \), the embedding vectors \( y_i \) are found by minimising the cost function in equation 9.1. That is, the optimal embedding can be found by computing the bottom
$d + 1$ eigenvector of the matrix and $M$. $d/D$ identifies the compression ratio of the embedded data space to the original data space. A bigger ratio means that more local points of data variations in the original space are kept in the embedded eigenspace, and a smaller ratio means that more global information are reserved by each LLE eigenvector of the embedded space. To reserve the local face variations, $d/D$ is constantly set as $1/3$ in our experiments.

Therefore, for face image data, LLE eigenfaces are a subset of eigenvectors of matrix $M$, which assumes that a facial image $x$ from training set $\{x_i\}_{i=1}^N$ can be reconstructed from its neighbours with the lowest reconstruction error in equation 9.2.

Figure 9.1b shows 10 LLE eigenfaces derived from the 20 face images of Figure 9.1a, and which are included in the membership group for authentication. Figure 9.1c shows 10 PCA eigenfaces derived from the same 20 face images of Figure 9.1a. As we can see, each PCA eigenface contains the global components from all 20 faces, which look obviously very different from one another; while the differences between LLE eigenfaces are not very distinct, because each LLE eigenface is obtained from a locally embedding computation, that contains only the local information of 20 face images.
9.2.2 Data Partition

In membership authentication, the group members can be divided into membership and non-membership group members as \( G = M \cup \bar{M} \). Applying the previous LLE eigenface technique to \( M \) and \( \bar{M} \) respectively, we obtain two representative eigenface sets such that the membership eigenfaces \( U_M = [u_1, u_2, \ldots, u_K] \) and the non-membership eigenfaces \( U_{\bar{M}} = [\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_K] \). They characterise the “membership-face” and “non-membership-face” respectively. Figure 9.1 is an example of membership LLE eigenfaces with membership size equal to 20.

On partitioning, we identify the two partitioned groups as a \( 2 \times n \) binary matrix \( V \), where the element is \( v_{ij} \) if the \( j^{th} \) data point \( x_j \) belongs to group \( i \), otherwise it is 0. Once the two cluster centers \( U_M \) and \( \bar{U}_M \) are fixed, then the clustering based on membership can be performed as follows:

\[
v_{ij} = \begin{cases} 
1 & \text{if } \min_{l=1}^{K} x_j \cdot u_l \leq \min_{k=1}^{K} x_j \cdot \bar{u}_k \\
0 & \text{otherwise}
\end{cases}
\]

Where \( \min_{l=1}^{K} x \cdot u_l \) is the minimum distance projected onto the membership eigenfaces \( (U_M) \), and \( \min_{k=1}^{K} x \cdot \bar{u}_k \) is the minimum distance projected onto the non-membership eigenfaces \( (\bar{U}_M) \).

Figure 9.2 illustrates an example of binary LLE data partition, where two dotted lines represent the two principal membership LLE eigenvectors (or eigenfaces) and the two solid lines represent two principal non-membership LLE eigenvectors (or eigenfaces). Each group member \( x_i \) in the space is projected to those eigenvectors, and is assigned to a class whose projected distance sum is the smallest. After completing the assignment of all group members, we obtain two disjoint subgroups that correspond to the membership group and the non-membership group, respectively. Each subgroup will be used as two labeled class datasets to train the CVM classifier.

We modified the LLE technique to take into consideration the labeling (membership and non-membership) of the data. Thus, as the evolving clusters are updating, members and nonmembers are forced to gather in numbers of separated sub-clusters by a recursive membership-based LLE clustering.
9.3 EXPERIMENT RESULTS AND DISCUSSION

FMA is performed on MPEG-7 face dataset, which consists of 1355 face images of 271 persons (5 different face images per person are taken), where each image has the size of 56 × 46 pixels. The images have been selected from AR(Purdue), AT&T, Yale, UMIST, University of Berne, and some face images obtained from MPEG-7 news videos (M. S. Kim, Kim, & Lee, 2003). We set the membership size ranging from 35 to 230 with a 10 persons interval to achieve datasets with dynamic class characteristics, and compared the proposed DE-CVM with CVM under the condition of dynamic distinctive class characteristics in FMA.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Group Size</th>
<th>CVM True-positive</th>
<th>Membership CVs</th>
<th>DE-CVM True-positive</th>
<th>Membership CVs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>88.57% (= 36/40)</td>
<td>36</td>
<td>88.57% (= 14/16)</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>55</td>
<td>85.78% (= 37/43)</td>
<td>37</td>
<td>89.09% (= 17/19)</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>84.00% (= 40/48)</td>
<td>40</td>
<td>90.66% (= 20/22)</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>95</td>
<td>83.15% (= 41/49)</td>
<td>41</td>
<td>85.26% (= 26/30)</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>115</td>
<td>81.74% (= 39/48)</td>
<td>39</td>
<td>91.30% (= 30/33)</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>135</td>
<td>79.25% (= 38/48)</td>
<td>38</td>
<td>88.88% (= 33/37)</td>
<td>33</td>
</tr>
</tbody>
</table>

Table 9.1: Classification results of the membership authentication
9.3 EXPERIMENT RESULTS AND DISCUSSION

Figure 9.3: A comparison of CVM and DE-CVM on classification accuracy under the condition of different membership group size

Figure 9.4: A comparison of CVM and DE-CVM on number of core vectors under the condition of different membership group size

Table 9.1 summarises the comparison result of membership authentication true-positive rate, and the number of core vectors between CVM and DE-CVM when the group size varies from 35 to 135 with a 20 interval, respectively. From Table 9.1, it can be observed that DE-CVM provides a better performance on membership authentication by obtaining higher true positive rates using fewer core vectors than CVM with all different group size. In addition, when the group size increases the true-positive rate of CVM decreases remarkably, while that of DE-CVM is always around 88% regardless of the group size.

Fig. 9.3 and Fig. 9.4 illustrate the number of core vectors and classification accuracy for CVM and DE-CVM under the condition of different membership group size. As seen in Fig. 9.3, the number of core vectors from CVM stays...
constantly around 80, while the membership group size changes between 35 and 135 (equals to 50% of total group size). However, although the number of core vectors from DE-CVM has a spiking increase, it never goes above the number of core vectors from CVM. On the other hand, Fig. 9.4 shows that generally DE-CVM achieves higher FMA accuracy than CVM, and the difference becomes as significant as 8% when the member group size ranges between 50 and 135. Recall that the number of DE-CVM core vectors is always smaller than the number of CVM core vectors, which indicates that DE-CVM core vectors are more discriminative than CVM for FMA. In other words, DE-CVM is more capable than CVM of factorising the class characteristics of membership and non-membership as fewer DE-CVM core vectors often deliver better FMA.

9.4 SUMMARY

This chapter presents a case study for dynamic face membership authentication. The results show that compared to the global modelling method CVM, local modelling method DE-CVM factorises class characteristics by reducing the sparseness area, discriminating core vectors on class interaction hyperplanes, and enabling outliers detection.

Next chapter presents a case study for string format stream data mining.
Chapter 10

THE APPLICATION OF META LEARNING STRING KERNEL MEBS SYSTEM TO SOLVE REAL WORLD STRING FORMAT STREAM DATA MODELING

The chapter presents a case study of modelling real world string format stream data that is carried out to demonstrate the effectiveness of the proposed meta learning algorithm for string kernel MEB optimisation. Three string kernel MEBS were experimented namely: edit-distance, bag-of-words and n-gram on four string datasets: Spam, Reuters-21578, Network Application Detection and e-News categorisation.

Section 10.1 explains the experimental setup and the evaluation criteria, section 10.2 describes the datasets description and the data pre-processing. The experimental results for the three string kernel SVMs are presented in section 10.3. Section 10.4 indicates the limitation of the method found from the case study and how to improve it in the future. Finally, section 10.5 gives the conclusion.

10.1 EXPERIMENT SETUP

The proposed algorithm was experimented on three string kernel MEBS (*edit-distance* MEB, *bag-of-words* MEB and *n-gram* MEB). As shown in Table 10.1, the algorithm was trained using training string dataset pool $L_{TR}$, and was tested on testing string dataset pool $L_{TS}$, for each string kernel MEB. In the experiments, the MEB cost parameter ($c$) was selected as $2^0, 2^1, ..., 2^{16}$ for all string kernel MEBS. The string kernel parameters $\lambda_L$ and $\lambda_B$ for edit-distance and bag-of-words string kernels, were selected as 0.001, 0.1, 0.25, 0.5. The substring
length in n-gram string kernel was selected as 1, 2, ..., 8 in the experiments. The string meta-feature, AvgMinTokenVal was not considered in the training stage, as it was having the value 0 for all datasets. Also, the global threshold for the AvgTokenThr was set to 2 in all the experiments. Support Vector Regression (SVR) was used to build the meta model. In the training stage, the parameters which yield lowest cross validation RMSE for SVR, were considered in regression (in building the meta model), for each string kernel MEB. 10 fold- cross validation was done for the top 10 predicted parameter combinations, on each string dataset. The performance evaluation was done considering Root Mean Squared Error (RMSE) for the top 10 predicted parameters on each dataset.

<table>
<thead>
<tr>
<th>Dataset Pool</th>
<th>String Dataset (Dataset Label)</th>
<th>Sting Kernel MEB</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTR</td>
<td>Spam</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Reuters-21578</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Network Application Detection</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>e-News Categorisation</td>
<td>-</td>
</tr>
<tr>
<td>LTS</td>
<td>Spam(1)</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Reuters-21578(2)</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Network Application Detection(3)</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>e-News Categorisation(4)</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 10.1: The string datasets used in training and testing the proposed algorithm

In the experiments, edit-distance and bag-of-words string kernels were implemented using LibMEB-2.9 (Chang & Lin, 2001) and n-gram string kernel was implemented using shogun octave interface (S. Sonnenburg, 2006). The string meta-feature computation program was coded using C++ language. All the experiments were run on a PC having Intel Core2 Duo 3GHz processor and 2.96 Gb RAM.

10.2 DATASETS AND DATA PRE-PROCESSING

Four string datasets were used in the string dataset pool \( L = \{ \text{Spam}, \text{Reuters-21578}, \text{Network Application Detection}, \text{e-News Categorisation} \} \). It was again subdivided into training dataset pool \( (L_{ TR}) \) and testing dataset pool \( (L_{ TS}) \),
where each consisted of unique string datasets. String dataset pool $L_{TR}$ was used to train the meta model in the proposed algorithm and string datasets pool $L_{TS}$ was used to test the proposed algorithm. For edit-distance and bag-of-words string kernel MEBs, training dataset pool $L_{TR} = \{Spam, Reuters-21578, Network Application Detection\}$ was used, and for n-gram MEB, $L_{TR} = \{Spam, Reuters-21578, Network Application Detection, e-News Categorisation\}$ was used. In testing, for edit-distance and bag-of-words string kernel MEBs, testing pool $L_{TS} = \{Spam, Reuters-21578, Network Application Detection\}$ was used, and for n-gram MEB, the testing pool $L_{TR} = \{Spam, Reuters-21578, Network Application Detection, e-News Categorisation\}$ was used. Table 10.1 summarises algorithm training and testing information on each string dataset. A detailed description about each string dataset is given below.

10.2.1 Spam Dataset

This dataset consists of 696 ham messages and 384 Spam messages from (Spam Assassin public mail corpus, 2002). There are two types of ham e-mails: easy ham (646) and hard ham (50). Easy ham e-mails are non-Spam messages without any Spam signatures and hard ham are non-Spam messages similar in many aspects to Spam messages which use unusual HTML markup, coloured text, Spam-sounding phrases, etc. Each e-mail message has a header, a body and some potential attachments. The training dataset consists with 810 messages (484 easy ham, 38 hard ham and 288 Spam) and testing dataset has 270 messages (162 easy ham, 12 hard ham and 96 Spam).

10.2.2 Reuters-21578 Dataset

The Reuters dataset used in the experiments has the exact split to (Lodhi et al., 2002). It consists of 470 documents: 380 for training and 90 for testing. Four document categories, those of earn, acquisition, crude and corn are available in the dataset. Table 10.2 shows the document distribution among the different categories.
10.2 Datasets and Data Pre-processing

<table>
<thead>
<tr>
<th>Class Label</th>
<th>Document Category</th>
<th>Training</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Earn</td>
<td>152</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>Acquisition</td>
<td>114</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>Crude</td>
<td>76</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>Corn</td>
<td>38</td>
<td>10</td>
</tr>
</tbody>
</table>

| Total       | 370               | 90       |

Table 10.2: Data distribution-reuters-21578 dataset

10.2.3 Network Application Detection Data

The dataset consists of network traffic data produced by network applications, such as http, https, imap, pop3, ssh, ftp and bittorent. All network data were captured, and sorted according to their protocols using “Wireshark” (Combs, 2008) and split into individual connections using tcpflow (Elson, 2003). Only TCP traffic was taken into account in the data capturing stage. The option ‘-s’ in “tcpflow” (Elson, 2003) was used to remove all non printable characters in a connection. Also, only the first 50 bytes of a connection were considered in preparing the dataset. Every connection was labelled according to the application type. Table 10.3 shows the label number for every application type and the number of instances in training and testing datasets.

<table>
<thead>
<tr>
<th>Class Label</th>
<th>Application/protocol</th>
<th>Training</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AIM</td>
<td>18</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>Bittorrent</td>
<td>140</td>
<td>59</td>
</tr>
<tr>
<td>2</td>
<td>http</td>
<td>583</td>
<td>249</td>
</tr>
<tr>
<td>3</td>
<td>pop</td>
<td>17</td>
<td>7</td>
</tr>
</tbody>
</table>

| Total       | 770                | 329      |

Table 10.3: Data distribution-application detection dataset
10.2.4 e-News Categorisation Data

The dataset is collected from four electronic newspapers: (New Zealand Herald, 2010), (The Australian, 2010), (The Independent, 2010) and (The Times, 2010), on five news topics (business, education, entertainments, sport and travel). Each document is labelled manually by skimming over the text to identify the category. Punctuations and stop words were removed from the dataset in advance. Table 10.4 shows detailed information about the dataset.

<table>
<thead>
<tr>
<th>Class Label</th>
<th>News Category</th>
<th>Training</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Business</td>
<td>227</td>
<td>97</td>
</tr>
<tr>
<td>1</td>
<td>Education</td>
<td>93</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>Entertainments</td>
<td>99</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>Sport</td>
<td>118</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>Travel</td>
<td>131</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>668</td>
<td>285</td>
</tr>
</tbody>
</table>

Table 10.4: Data distribution e-Newsgroup dataset

10.3 RESULTS

The result of the experiment for three string kernel MEBs are discussed in this section. Section 10.3.1 explains the results for edit-distance MEB optimisation, using the proposed algorithm. The experimental results for bag-of-words MEB optimisation are presented in section 10.3.2. Section 10.3.3 explains the results for n-gram MEB optimisation, using the proposed algorithm.

10.3.1 Meta Learning for Edit-Distance MEB Optimisation

Here, the proposed algorithm was used to optimise edit-distance MEB. The algorithm attempts to find the optimum parameter combination ($\lambda_L$ and MEB cost) for edit-distance MEB on three string datasets: Spam, Reuters-21578 and
network application detection, in test dataset pool $L_{TS}$ (refer Table 10.1). In the experiments, the SVR parameters: $\gamma = 0.084$ and $SVR \ Cost=5400$ were used in regression. The actual accuracy and the predicted accuracy for the top 10 predicted parameter combinations are shown in Table 10.5. Also, the table shows the RMSE for top 10 predicted parameter combinations on each dataset.

According to Table 10.5a and Table 10.5b, the optimum parameters produced by the proposed algorithm yield very low predicted and actual classification accuracies, for edit-distance MEB, on Spam and Reuters-21578 string datasets. This shows that the edit-distance MEB is not suitable for string classification on Spam and Reuters-21578 datasets. However, according to Table 10.5c, the algorithm produces optimum parameters which yield good string classification accuracies on network application detection dataset, for edit-distance MEB (with a low RMSE).

<table>
<thead>
<tr>
<th>cost</th>
<th>$\lambda_\ell$</th>
<th>rank</th>
<th>predicted %</th>
<th>actual %</th>
</tr>
</thead>
<tbody>
<tr>
<td>65536</td>
<td>0.000488</td>
<td>1</td>
<td>2.36387</td>
<td>0.37</td>
</tr>
<tr>
<td>2</td>
<td>0.000488</td>
<td>2</td>
<td>2.36381</td>
<td>5.19</td>
</tr>
<tr>
<td>4</td>
<td>0.000488</td>
<td>3</td>
<td>2.36377</td>
<td>4.44</td>
</tr>
<tr>
<td>8</td>
<td>0.000488</td>
<td>4</td>
<td>2.36371</td>
<td>3.33</td>
</tr>
<tr>
<td>16</td>
<td>0.000488</td>
<td>5</td>
<td>2.36357</td>
<td>1.11</td>
</tr>
<tr>
<td>32</td>
<td>0.000488</td>
<td>6</td>
<td>2.36331</td>
<td>0.37</td>
</tr>
<tr>
<td>2</td>
<td>0.000976</td>
<td>7</td>
<td>2.36302</td>
<td>4.44</td>
</tr>
<tr>
<td>4</td>
<td>0.000976</td>
<td>8</td>
<td>2.36299</td>
<td>3.33</td>
</tr>
<tr>
<td>8</td>
<td>0.000976</td>
<td>9</td>
<td>2.36292</td>
<td>1.11</td>
</tr>
<tr>
<td>65536</td>
<td>0.000976</td>
<td>10</td>
<td>2.36286</td>
<td>0.37</td>
</tr>
</tbody>
</table>

root mean squared error: 1.831455

<table>
<thead>
<tr>
<th>cost</th>
<th>$\lambda_\ell$</th>
<th>rank</th>
<th>predicted %</th>
<th>actual %</th>
</tr>
</thead>
<tbody>
<tr>
<td>65536</td>
<td>0.000488</td>
<td>1</td>
<td>3.30463</td>
<td>23.33</td>
</tr>
<tr>
<td>2</td>
<td>0.000488</td>
<td>2</td>
<td>3.30454</td>
<td>24.44</td>
</tr>
<tr>
<td>4</td>
<td>0.000488</td>
<td>3</td>
<td>3.30451</td>
<td>26.67</td>
</tr>
<tr>
<td>8</td>
<td>0.000488</td>
<td>4</td>
<td>3.30445</td>
<td>23.33</td>
</tr>
<tr>
<td>16</td>
<td>0.000488</td>
<td>5</td>
<td>3.30432</td>
<td>23.33</td>
</tr>
<tr>
<td>32</td>
<td>0.000488</td>
<td>6</td>
<td>3.30407</td>
<td>23.33</td>
</tr>
<tr>
<td>65536</td>
<td>0.000488</td>
<td>7</td>
<td>3.30379</td>
<td>23.33</td>
</tr>
<tr>
<td>2</td>
<td>0.000976</td>
<td>8</td>
<td>3.30377</td>
<td>23.33</td>
</tr>
<tr>
<td>4</td>
<td>0.000976</td>
<td>9</td>
<td>3.30371</td>
<td>23.33</td>
</tr>
<tr>
<td>8</td>
<td>0.000976</td>
<td>10</td>
<td>3.30371</td>
<td>23.33</td>
</tr>
</tbody>
</table>

root mean squared error: 20.846794

(a) Spam Data

<table>
<thead>
<tr>
<th>cost</th>
<th>$\lambda_\ell$</th>
<th>rank</th>
<th>predicted %</th>
<th>actual %</th>
</tr>
</thead>
<tbody>
<tr>
<td>32768</td>
<td>0.0625</td>
<td>1</td>
<td>73.1729</td>
<td>75.99</td>
</tr>
<tr>
<td>32768</td>
<td>0.03125</td>
<td>2</td>
<td>73.1653</td>
<td>75.99</td>
</tr>
<tr>
<td>32768</td>
<td>0.125</td>
<td>3</td>
<td>73.1602</td>
<td>75.99</td>
</tr>
<tr>
<td>32768</td>
<td>0.015625</td>
<td>4</td>
<td>73.1556</td>
<td>75.99</td>
</tr>
<tr>
<td>8192</td>
<td>0.125</td>
<td>5</td>
<td>73.1526</td>
<td>75.99</td>
</tr>
<tr>
<td>8192</td>
<td>0.25</td>
<td>6</td>
<td>73.1526</td>
<td>75.99</td>
</tr>
<tr>
<td>32768</td>
<td>0.015625</td>
<td>7</td>
<td>73.1518</td>
<td>75.99</td>
</tr>
<tr>
<td>32768</td>
<td>0.5</td>
<td>8</td>
<td>73.1515</td>
<td>75.99</td>
</tr>
<tr>
<td>8192</td>
<td>0.25</td>
<td>9</td>
<td>73.1514</td>
<td>75.99</td>
</tr>
<tr>
<td>65536</td>
<td>0.5</td>
<td>10</td>
<td>73.1512</td>
<td>75.99</td>
</tr>
</tbody>
</table>

root mean squared error: 2.833499

(b) Reuters-21578

<table>
<thead>
<tr>
<th>cost</th>
<th>$\lambda_\ell$</th>
<th>rank</th>
<th>predicted %</th>
<th>actual %</th>
</tr>
</thead>
<tbody>
<tr>
<td>32768</td>
<td>0.0625</td>
<td>1</td>
<td>73.1729</td>
<td>75.99</td>
</tr>
<tr>
<td>32768</td>
<td>0.03125</td>
<td>2</td>
<td>73.1653</td>
<td>75.99</td>
</tr>
<tr>
<td>32768</td>
<td>0.125</td>
<td>3</td>
<td>73.1602</td>
<td>75.99</td>
</tr>
<tr>
<td>32768</td>
<td>0.015625</td>
<td>4</td>
<td>73.1556</td>
<td>75.99</td>
</tr>
<tr>
<td>8192</td>
<td>0.125</td>
<td>5</td>
<td>73.1526</td>
<td>75.99</td>
</tr>
<tr>
<td>8192</td>
<td>0.25</td>
<td>6</td>
<td>73.1526</td>
<td>75.99</td>
</tr>
<tr>
<td>32768</td>
<td>0.015625</td>
<td>7</td>
<td>73.1518</td>
<td>75.99</td>
</tr>
<tr>
<td>32768</td>
<td>0.5</td>
<td>8</td>
<td>73.1515</td>
<td>75.99</td>
</tr>
<tr>
<td>8192</td>
<td>0.25</td>
<td>9</td>
<td>73.1514</td>
<td>75.99</td>
</tr>
<tr>
<td>65536</td>
<td>0.5</td>
<td>10</td>
<td>73.1512</td>
<td>75.99</td>
</tr>
</tbody>
</table>

root mean squared error: 2.833499

(c) Network Application Detection

Table 10.5: Experimental results for Edit-Distance MEB optimisation: (the top 10 predicted parameter combinations using the proposed algorithm on each string dataset)
10.3.2 Meta Learning for Bag-of-Words MEB Optimisation

Here, the proposed algorithm attempts to find the optimum parameter combination ($\lambda_B$ and MEB cost) for bag-of-words MEB. Initially, the algorithm was trained on a training dataset pool $L_{TR}$ (refer Table 10.1). Then, the algorithm predicted the string classification accuracies for bag-of-words MEB on three different string datasets in test dataset pool $L_{TS}$ (refer Table 10.1). The SVR parameters: $\gamma = 0.88$ and $SVR\ Cost=450$ were used in regression. 10-fold cross validation was done for the top 10 predicted parameter combinations, and RMSE was calculated for the same. Table 10.6 shows the top 10 predicted parameter combinations on three string datasets for bag-of-words MEB.

According to Table 10.6b and Table 10.6c, the proposed algorithm produces parameter combinations which yield high classification accuracies on reuters-21578 and network application detection datasets with a very low RMSE. However, on Spam dataset, the optimised parameter combinations produced by the proposed algorithm yield average string classification accuracies (see Table 10.6a). Considering the overall high string classification accuracy (with a very low average RMSE) shown in Table 10.6 and Table 10.8, on all three datasets, one can say that the proposed algorithm produces optimised parameter combinations which yield good string classification accuracies, for bag-of-words MEB.

10.3.3 Meta Learning for N-gram MEB Optimisation

In this experiment, the algorithm attempts to find optimised parameters (substring length and MEB cost) for n-gram MEB. The algorithm was trained on string dataset pool $L_{TR}$ and tested on testing string dataset pool $L_{TS}$ (refer Table 10.1). The SVR parameters: $\gamma = 0.95$ and $SVR\ Cost=500$ were used in regression. 10-fold cross validation was done for the top 10 predicted parameter combinations on each string dataset. Table 10.7 summarises the experiment results for the top 10 predicted combinations on each dataset.
10.4 LIMITATIONS

According to Table 10.7, the proposed algorithm produces optimised parameters, which yield good string classification accuracies for n-gram MEB, on all four string datasets. The algorithm has a very low RMSE for top 10 predicted on Spam, Reuters-21578 and network application detection datasets (see Table 10.7a, Table 10.7b, and Table 10.7c). Even though, the algorithm has quite a high RMSE on the e-News categorisation dataset, the top 10 predicted parameter combinations yield good string classification accuracies on the dataset (see Table 10.7d).

10.4 LIMITATIONS

The limitations that identified in the research are discussed below. Firstly, for n-gram kernel, the proposed algorithm yields classification accuracies over 100% on Spam and Reuters-21578 string datasets. This can be resolved if one is able to set upper and lower bounds in the algorithm. Secondly, the string meta-features could be more descriptive, since they can explain the string dataset more clearly than just numbers representing the string dataset. Thirdly, while
10.4 LIMITATIONS

Table 10.7: Experimental results for N-gram MEB optimisation: (the top 10 predicted parameter combinations using the proposed algorithm on each string dataset)

<table>
<thead>
<tr>
<th>String Kernel</th>
<th>Dataset</th>
<th>RMSE</th>
<th>Avg RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit-Distance</td>
<td>Spam</td>
<td>1.831455</td>
<td>8.503916</td>
</tr>
<tr>
<td></td>
<td>Reuters-21578</td>
<td>20.846794</td>
<td>8.503916</td>
</tr>
<tr>
<td></td>
<td>Network Application Detection</td>
<td>2.833499</td>
<td>8.503916</td>
</tr>
<tr>
<td>Bag-of-Words</td>
<td>Spam</td>
<td>2.899333</td>
<td>5.353826</td>
</tr>
<tr>
<td></td>
<td>Reuters-21578</td>
<td>1.273886</td>
<td>5.353826</td>
</tr>
<tr>
<td></td>
<td>Network Application Detection</td>
<td>0.592261</td>
<td>5.353826</td>
</tr>
<tr>
<td>N-gram</td>
<td>Spam</td>
<td>0.971168</td>
<td>5.353826</td>
</tr>
<tr>
<td></td>
<td>Reuters-21578</td>
<td>2.712373</td>
<td>5.353826</td>
</tr>
<tr>
<td></td>
<td>Network Application Detection</td>
<td>1.386739</td>
<td>5.353826</td>
</tr>
<tr>
<td></td>
<td>e-News Categorisation</td>
<td>16.345027</td>
<td>5.353826</td>
</tr>
</tbody>
</table>

Table 10.8: Root Mean Squared Error (RMSE) for string kernel MEB optimisation on each string dataset (for top 10 predicted parameter combinations)

a meta learning algorithm is usually tested on a large number of datasets, the high computational cost of string data and the unavailability of bench-mark string datasets, have forced this researcher to use only four string datasets in the experiments.

These limitations could be also caused by complex distribution of data samples. The proposed meta learning string kernels MEBs algorithm only learns
the global model so far. In order to have a more accurate model for string data, a local modelling string kernel method needs to be developed in future.

10.5 SUMMARY

This case study shows that the proposed algorithm produces parameter combinations that yield good string classification accuracies on most of the datasets. They also reveal that some string kernel MEBs may not be suitable for carrying out the string classification required on certain string datasets. Specifically, edit-distance MEB yields poor string classification results on both Spam and Reuters-21578 string datasets.

The proposed method has three main contributions to the field of machine learning:

1. **String Meta-features**: The defined string meta-features can be used for extracting meta knowledge from any string dataset;

2. **Meta-Learning for String Classification Principle**: explains the procedure for applying meta-learning on string classification but using extracted meta-knowledge via string meta-features;

3. **Meta Learning Algorithm for String Kernel MEB Optimisation**: using the Meta-Learning for String Classification Principle, a novel string kernel optimisation method is derived, which is able to predict optimum string kernel MEB parameters for a given string kernel MEB on a string dataset by calculating relevant string meta-features.

Directions for future work and conclusion of the PhD thesis are given in the next chapter.
CONCLUSION AND FUTURE DIRECTIONS

In this final chapter, the main achievements of the study are summarised. Several directions for future work are also discussed that would help in further improving the efficacy of the entire system.

The chapter is organised as follows: first, the main achievements of this thesis are summarised from three perspectives: in terms of basic on-line learning algorithm, evolving connectionist systems adopting, and string kernels learning. The section closes by stressing the original contributions of this work and giving a summary of the experimental results. The dissertation closes with a brief discussion of future directions in five possible areas.

11.1 SUMMARY OF ACHIEVEMENTS

This PhD study shows how evolving connection systems (i.e. local modelling) are adapted to stream data mining based on a fast, on-line kernel based learning algorithm. By means of local modelling, we can solve many real world stream data modelling issues such as dealing with large size, high dimensionality, skewed class distribution. The idea of focusing on the unique problem subspaces appears to be beneficial to the stream data modelling problems. It allows identification of sub-problems and allows their further study. In addition, this research developed novel string kernels method to overcome the difficulty of mining string format stream data. At the same time, I have also published 3 conference papers, submitted 1 journal paper, delivered 3 technical reports and developed 2 prototype systems.
11.1 SUMMARY OF ACHIEVEMENTS

11.1.1 Fast, On-line Kernel Based Learning Algorithm

We have developed a novel on-line kernel based learning method with low computational cost called on-line core vector machines (OCVM). This method inherits the characteristic of original CVM whose computational complexity is independent from the size of training samples. In addition, this method allows us to on-line analyse the stream data in high dimensional space to be analysed on-line which discriminates skewed class distribution.

The effectiveness of OCVM is evaluated by comparing its classification accuracy and train/testing time to those of other traditional methods on 10 benchmark datasets.

11.1.2 Evolving Connectionist Systems on Stream Data Modeling

Evolving connectionist systems (ECOS) are modular connectionist-based systems that evolve their structure and functionality in continuous, self-organised, on-line, adaptive, interactive way from incoming information; they can process both data and knowledge in supervised learning and/or unsupervised way (Kasabov & Song, 2002; Kasabov, 2002).

The proposed method HCVM learns local models from data through clustering of child-classes data and associating a set of local core vectors for each parent-class. DE-CVM is proposed and works similarly to DENFIS (see chapter 3) but in kernel spaces.

The advantages of adopting evolving connectionist systems are confirmed in two case studies: network intrusion detection and face membership authentication where small number of core vectors can provide higher classification accuracy.

11.1.3 String Format Stream Data Learning

A novel string classification method called Meta Learning String Kernels CVMs for string format stream data mining is proposed. Since one major task for
stream data mining is to categorise texture, this method is an important part of an integrated stream data modelling system.

The capability of this method is verified in the experiments on several string classification tasks.

11.2 FUTURE DIRECTIONS

This section suggests some promising future directions for the development of the methods and systems in stream data modeling.

11.2.1 Evolving Clustering Optimisation

In our further work, we would like to address the optimisation problem of the total number of CVMs required for optimal solution. It can only be determined using cross validation method and by investigating on measuring the density of the MEB, and further exploiting new models based on ‘evolving clusters’.

11.2.2 On-line methods for personalised modeling

One of the issues with local and distance-based models is that they require good definition of the problem space in order to perform properly, which is often difficult and lacking in many problems (e.g. biological modeling) as they often involve a large number of noisy variables. In order to apply local or personalised models to this type of problem, an in-depth analysis of the features is necessary.

A personalised regression model with incremental feature selection will be proposed during next stage. This method applies incremental feature selection on variables that were ranked using univariate analysis and results from previous studies. This set of variables was then used to define the problem space and identify the relevant subset of data for each prediction. The global regression model is then optimised with this subset of training data to put a focus on the test input vectors residing in problem subspace.
11.2.3 Spatial and Temporal Processing

Compared to traditional processing, temporal contexts are particularly important in stream processing. Giannotti, Nanni, Pinelli, and Pedreschi (2007), presents a study on clustering trajectories of mobile objects (e.g., mobile phones). Problems faced in this spatio-temporal data mining task concern the identification of the proper spatial granularity level, the selection of the significant temporal sub-domains, the choice of the most promising clustering method, and the formalisation of the notion of (dis)similarity among trajectories. The authors recommend a density-based approach to trajectory clustering, and stress the importance of temporal focusing to isolate the clusters of higher quality. Both aspects have been tested on a data set automatically generated by a synthesiser of trajectory data.

Next stage we will use micro-clustering phase which is the on-line statistical data collection algorithm. This process is not dependent on any user input such as the time horizon or the required granularity of the clustering process. The aim is to maintain statistics at a sufficiently high level of (temporal and spatial) granularity so that it can be effectively used by both off-line and on-line analysis.

11.2.4 Local modeling for String Kernels

So far the proposed meta-learning string kernel CVMs only works on global modeling. Next stage we will aim on local modeling string kernels methods.
INTRUSION DETECTION VISUALISATION SYSTEM

Appendix A and B present two stream data visualisation systems developed based on our proposed HCVM algorithm (Y. Chen et al., 2009), respectively.

Network Intrusion Detection visualisation System (NIDVS) is developed to simulate the process of world-wide network intrusion detection. This visualisation system can monitor the Internet data stream. Moreover, this system is adaptive to various learning methods thus it provides the opportunity to compare the effectiveness of different methods when used on the same data.

A.1 DESIGN OVERVIEW

NIDVS simulates the process of world-wide network intrusion detection, where network traffic is visualized as data streaming from different source to the destinations, and intrusion filtered by the HCVM algorithm (Y. Chen et al., 2009). This NIDVS has two main functions: to monitor real-time network traffic and to demonstrate the effectiveness of the HCVM algorithm in terms of detection accuracy. NIDVS visualises the streaming data on a 3D world map, where network traffic is traced from the source to destination city (identified by latitude and longitude) for each data packet in transfer. A sequence of moving arrows represent network traffic as a set of data packets where data is indicated by green arrows, and spam/intrusion data is represented as red arrows. In NIDVS, network traffic data is associated with a certain speed to reflect the speed variation of real network traffic. For the convenience of observation, the height of traffic path can be modified to suit users sight.
The NIDVS is shown in Figure A.1. Users can view the data flow and detection process on the left and control the view of data flow on the right side. The statistical results (*i.e.* total accuracy, number of intrusions from each source) are also shown on the right.

### A.2 HARDWARE AND SOFTWARE REQUIREMENTS

The NIDVS requires Intel E8400@3.00 GHz or faster with at least 1.97 GB RAM. The video card should have over 256 MB RAM with support on Pixel Shader 3.0. The required software environment for NIDVS is MATLAB R2009a with Mapping toolbox 2.7.1, on the platform of Microsoft Windows XP Professional with Service Pack 3 or advanced. The best screen resolution is 1280 by 1024 pixels.

#### A.2.1 System Architectural Design

The system is structured as shown by the diagram in Figure A.2. It consists of five basic modules, namely ‘Data Acquiring’, ‘Data visualisation’, ‘Intrusion Detection’, ‘Spam Collection’ and ‘Spam labelling’. The input data can
be a set of example data or real Internet data. ‘Data Acquiring’ inputs data into the system, and ‘Data visualisation’ displays data as a set of arrows on the screen. The data can be labelled as ‘spam/intrusion’ or ‘normal’ by the ‘HCVM Intrusion Detection’ module. If the data is classified as spam/intrusion, then a detailed description of the spam will be recorded by the ‘Spam Collection’ module. The ‘Spam labelling’ module is in charge of presenting data as coloured arrows. Spam/intrusion data is presented with red colour, normal data with green colour. As seen in Figure A.2, the system runs in a loop that terminates when no data is input.

Figure A.2: The NIDVS system design diagram.
A.3 DETAILED DESCRIPTION OF COMPONENTS

The NIDVS consists of two components. The first component performs classification using the HCVM method. The second component visualizes data flow and spam detection on screen.

A.3.1 HCVM in the NIDVS

For labelling network intrusions as they state hierarchical multi-label structure, a high-speed hierarchical multi-label classification (HMC) algorithm, called hierarchical minimum enclosing balls (HCVM), is developed. We model the multi-label hierarchy into a data Hyper-Sphere constructed by numbers of MEBs. The MEBs separate, encompass and overlap with each other and form a tree structure representing the encoded multi-label hierarchy. When provided with an unlabelled sample, the HCVM seeks an MEB enclosing the sample, and multi-labels the sample according to the MEBs position in the hierarchy. The HCVM has been tested on a Gaussian synthetic data, the RCV1-V2 text data, and the KDD99 intrusion detection dataset for multi-label classification.

A.3.2 visualisation

The visualisation system provides a user interface for both novice and advanced users. It shows the flow of data between cities on the globe. During the data transfer, there is a spam/intrusion detection system processing data on each path. The spam/intrusion data will be detected and labelled with different colours. This visualises NID in real time, at the same time it tests the performance of the NID algorithm (i.e. the HCVM).

Users can replace the testing data with their own data. They simply need to replace the data files stored in ‘root/Temp Data/or_data’ with their own data files. The system will automatically allocate IPs, cities and network traffic speed to each data item. The allocation also selects data randomly from those data files and will never repeat them. Note that the input data is required to be formatted as a data matrix where the last column contains the class label.
The default NID method is the HCVM, although users can also change the detection algorithm. For example, users can choose to use SVM instead of HCVM for NID.

A.4 USER INTERFACE DESIGN

A.4.1 Description of the User Interface

The user interface consists of four components: Network Traffic visualisation, The NIDVS control panel, Network Intrusion Detection Report, NID Accuracy Tracking and they are described below.

Objects and Actions

1. Network Traffic visualisation

![Network Traffic visualisation of NIDVS](image)

Figure A.3: The network traffic visualisation of NIDVS, in which the green arrows represent normal data and red arrows indicate spam/intrusion.

Figure A.3 depicts a network traffic visualisation of the NIDVS, where a sequence of arrows in colours represent a network data stream flowing from the source to target city. The green or red arrows represent normal or intrusion/spam data packet, respectively.
2. Control Panel

The user is able to change the view of visualisation by using NIDVS control panel shown in Figure A.4. The ‘Radius of Navigation Path’ is the parameter controlling the height of the network traffic path. The ‘Number of Arrows’ affects the size of arrows. For a fixed source and target network traffic, a larger number of arrows result in relatively smaller size of the arrows.

The four buttons control the NIDVS system: ‘Start’ initiates the visualisation demo, ‘Pause/Continue’ suspends or proceeds with the visualisation of data flow. ‘Terminate’ stops the visualisation demonstration and reports the present detected spam/intrusion list as shown in Figure A.5a. The spam/intrusion list is stored meanwhile in a ‘txt’ file named ‘spam report.txt’, recording the details of every spam, its source and target city, and the time when the spam was detected. Button ‘Show results’ displays a summary of NID performance in a pop-up window shown in Figure A.5b. This summarises the performance of the used NID algorithm, the number of normal data samples, spam sample, the accuracy of normal data detection (i.e. truth negative accuracy), the accuracy of spam intrusion data (i.e. truth positive accuracy) and the general accuracy.

3. Network Intrusion Detection Report

Figure A.6 depicts the statistical report of the current status of NID, which includes information about the number of processed data samples, the source and target city of the data package (reflected by the IP address), and the number of samples that are labelled as intrusions.
4. NID Accuracy Tracking

Figure A.7 presents a real time intrusion detection report in graphical form. The performance of intrusion detection by HCVM varies over time. The final detection accuracy is fixed until all data runs out.
Figure A.7: Graph for real time intrusion detection report, where the $x$ - axis represents the amount of data processed, and $y$ - axis represents the detection accuracy in percent.
HCVM PROTOTYPE SYSTEM

The HCVM prototype system demonstrates the learning progress of HCVM which gives a better understanding of on-line learning process. This system also provides the comparison of globe modelling and local modelling. User can evaluate the advantage of using local modelling easily from the visualisation.

B.1 DESIGN OVERVIEW

The HCVM Prototype System simulates the process of learning on-line stream data. It uses 10% of KDD’99 training data. This prototype system demonstrates the MEB updating progress for each cluster as 3D balls and 2D boundary image. The core vectors for each MEB are represented as small dots. For convenience of observation, the point of observation can be changed to suit the user.

The interface of the HCVM Prototype System is shown in Figure B.1. Users are able to observe the learning progress on the left. The control panel is on right and details for each MEB are shown just under the control panel.

B.2 HARDWARE AND SOFTWARE REQUIREMENTS

The NIDVS requires Intel E8400@3.00 GHz or faster with at least 1.97 GB RAM. The video card should have over 256 MB RAM with support on Pixel Shader 3.0. The required software environment for NIDVS is: MATLAB.
R2009a on the platform of Microsoft Windows XP Professional with Service Pack 3 or advanced. The best screen resolution is 1280 by 1024 pixels.

B.3 USER INTERFACE DESIGN

B.3.1 Description of the User Interface

The user interface consists of four components: MEBs visualisation, control panel, MEB information list table as explained below.

B.3.2 Objects and Actions

1. MEBs visualisation

Figure B.2 depicts a dynamic visualisation of MEB, its core vectors and the corresponding boundary for each cluster. While 3D balls in different colours represent a cluster for a sub-class, the dots in the balls represent the core vectors. Boundaries shown in the right bottom corner are formed by the same-colour core vectors. The parent class boundary is in black colour.

2. Control Panel
The user is able to select different learning modes by using the control panel shown in Figure B.3. Global mode learning can be demonstrated if ‘Probe’ is selected. You can compare the effectiveness of global mode learning with local mode by selecting ‘Probe sub-type’. The theory of local mode learning can be seen if ‘All Types’ is selected.

The three buttons control the prototype system. ‘Start’ initiates the visualisation demo. ‘Pause/continue’ suspends or proceeds with the visualisation of data flow. ‘Terminate’ stops the visualisation demonstration.

3. MEB information list table

Figure B.4 depicts the dynamical report of the details of each MEB shown in the visualisation area, which includes information about the name of each class, current ball centre and radius corresponding to the same colour of MEB.
### Figure B.4: MEB information list table

<table>
<thead>
<tr>
<th>Name</th>
<th>Radio</th>
<th>Center</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>button</td>
<td>2.0888, 0.0048, 0.1164, 0.38246</td>
<td></td>
<td></td>
</tr>
<tr>
<td>portsve</td>
<td>1.8987, 0.05426, 0.228137, 0.034322</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iden</td>
<td>1.2874, 0.00464, 0.21497, 0.000007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>imscript</td>
<td>1.9533, 0.00071, 0.20070, 0.032715</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
REFERENCES


database systems (pp. 234–243). New York, NY, USA: ACM.


Statistical Pattern Recognition, 16–29.


Hripcsak, G., & Rothschild, A. S. (2005). Agreement, the f-measure, and reliability in information retrieval. *Journal of the American Medical In-

References 159
formatics Association, 12(3), 296 - 298.


