UNIVERSITY OF SOUTHAMPTON

One-pass Algorithms for Large and Shifting Data Sets

by

Bassam Farran

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

June 2010

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Bassam Farran

For many problem domains, practitioners are faced with the problem of ever-increasing amounts of data. Examples include the UniProt database of proteins which now contains $\sim$ 6 million sequences, and the KDD '99 data which consists of $\sim$ 5 million points. At these scales, the state-of-the-art machine learning techniques are not applicable since the multiple passes they require through the data are prohibitively expensive, and a need for different approaches arises. Another issue arising in real-world tasks, which is only recently becoming a topic of interest in the machine learning community, is distribution shift, which occurs naturally in many problem domains such as intrusion detection and EEG signal mapping in the Brain-Computer Interface domain. This means that the i.i.d. assumption between the training and test data does not hold, causing classifiers to perform poorly on the unseen test set.

We first present a novel, hierarchical, one-pass clustering technique that is capable of handling very large data. Our experiments show that the quality of the clusters generated by our method does not degrade, while making vast computational savings compared to algorithms that require multiple passes through the data. We then propose Voted Spheres, a novel, non-linear, one-pass, multi-class classification technique capable of handling millions of points in minutes. Our empirical study shows that it achieves state-of-the-art performance on real world data sets, in a fraction of the time required by other methods. We then adapt the VS to deal with covariate shift between the training and test phases using two different techniques: an importance weighting scheme and kernel mean matching. Our results on a toy problem and the real-world KDD '99 data show an increase in performance to our VS framework. Our final contribution involves applying the one-pass VS algorithm, along with the adapted counterpart (for covariate shift), to the Brain-Computer Interface domain, in which linear batch algorithms are generally used. Our VS-based methods outperform the SVM, and perform very competitively with the submissions of a recent BCI competition, which further shows the robustness of our proposed techniques to different problem domains.

Contents

List of Figures

List of Tables

Nomenclature

$\mathbf{w}$	The weight vector
b	The bias
$\mathbf{x}$	bold face letters denote vectors
$\langle \cdot, \cdot \rangle$	the inner product
$\|\| \cdot \|\|$	the 2-norm
N	number of samples or data set size
$\phi(\cdot)$	Feature projection
F	Feature space
$k(\mathbf{x}, \mathbf{x}')$	The kernel function $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$
K	the kernel matrix
ξ	The SVM slack variable
d	The dimensionality of the data
σ	The width of the Gaussian kernel
$d(\mathbf{x}, \mathbf{x}')$	Distance between two vectors
$p_{tr}(\mathbf{x})$	The density of $\mathbf{x}$ in the training distribution
$p_{te}(\mathbf{x})$	The density of $\mathbf{x}$ in the testing distribution
$w(\mathbf{x})$	The weight assigned to $\mathbf{x}$

Acknowledgements

The process of obtaining a PhD is not a simple one, and requires interaction with many people. This makes acknowledging everyone individually difficult, as everybody you meet and discuss your research with contributes in a way. However, I would like to thank first and foremost Dr Craig Saunders, with whom I started my research from day one: my MSc dissertation. Your approach to Machine Learning helped me develop a deep understanding of it, but more importantly made it enjoyable. Thanks for being extremely patient, for always believing in me, and for continuing to help to this day. I would also like to express my deepest gratitude to my current supervisor Prof. Mahesan Niranjan, who has also been very patient, understanding, and helpful despite supervising my research late into my PhD. Your immense knowledge of the field, your enthusiasm, and your charisma motivates the entire research group, and your students in particular. I would also like to thank Prof. Bob Damper for his useful comments during my MPhil transfer examination, and Dr C.Q. Chang for his insightful comments on my work, and for introducing me to the exciting world of Brain-Computer Interfaces.

I would also like to thank all of my friends at the Information: Signals, Images, Systems (ISIS) research group at the University of Southampton for their helpful comments, and for their support. You truly have made my PhD journey enjoyable. I would especially like to thank Mohamed Qasem, whom I consider a genius, for selflessly helping me with my problems. I hope I was half as useful to you as you were to me.

Last, but definitely not least, I would like to express my extreme gratitude to my friends and family, who have all helped maintain my mental sanity. I would like to thank my father, Prof. Hani Farran, who is also my role model, for constantly lifting my spirits and encouraging me when I was feeling down, and for his financial support; my mother Nadia Farran for her love and support; and my siblings Arij, Mohammad, and Reda for listening to me and always telling me to 'keep going, it's almost over'. Without you, I would not be where I am today.

To my parents Hani and Nadia Farran, and my siblings...

Chapter 1

Introduction

Learning, like intelligence, involves a broad range of processes, making a clear and accurate definition difficult. Dictionaries often use phrases such as "learning from experience" or "gaining knowledge or skill through study and experience". Using such terminology, we can define machine learning as a scientific discipline concerned with the development of algorithms that are capable of learning from past data to perform better in the future. This involves automatically extracting complex patterns from data, and making intelligent decisions - a task that is exceedingly difficult for humans to accomplish. Of course, the algorithm could not make intelligent decisions without learning, since learning is fundamental for intelligence. Since its emergence in the 1950s, machine learning has been used in many practical applications, such as computer vision, bioinformatics, spam detection, speech and handwriting recognition, and the list goes on.

In their earlier days, machine learning techniques focused on obtaining the best possible generalisation on the data available. For instance, neural networks passed multiple times over the training data in order to minimise the cost of some target function. While this was acceptable and feasible in the past, given that the available data were relatively small, this no longer reflects the case today. Many new problem domains require us to deal with gigabytes of data which continue to grow year after year, and so algorithms that could potentially trade off some of their generalisation ability in order to be faster and more scalable are the only way to progress. This new concern within the machine learning community is visible with more international challenges being geared towards large scale learning, such as the recent PASCAL Large Scale Learning Challenge[1] and the KDD Cup 2009[2].

Examples of such problem domains are bioinformatics, where the number of sequenced proteins grows year after year, and computer intrusion detection, which has no limit on the amount of data available as networks continuously generate data. The latter also has

[1] http://largescale.first.fraunhofer.de/about/
[2] http://www.kddcup-orange.com/

more stringent requirements than the former, where not only does the algorithm have to deal with gigabytes of data, it must do so in real time as well. With time, the intrusion detection system (IDS) would have come across a lot of data, and could improve its performance by extracting useful information from it. This is exactly what the field of machine learning is concerned with, since it is impossible for humans to find patterns in such vast quantities of data. Practitioners in the field are aware of this, and for decades, they have employed many different machine learning techniques ranging from anomaly detectors (built from normal data only), to signature based detectors which can only detect intrusions that the IDS has seen before. The issue, however, is that unless the applied learning technique was explicitly developed for use on large data sets ($>$ 1m points), it cannot be used for intrusion detection tasks. For example, the vanilla Support Vector Machine (SVM)(see Section 2.2) requires $O(N^2)$ memory and $O(N^3)$ time to train (where N is the size of the training data). This makes the vanilla SVM impractical for the large data we aim to look at in this thesis. To overcome this issue, researchers have developed techniques that explicitly aim to tackle large data, which fall loosely into three main categories:

- Subsampling
- Caching
- Online algorithms

Subsampling involves sampling a small amount of data to use for training, either before the training starts, or during every iteration in the training phase. This is definitely a speed-up, but for tasks such as intrusion detection, where intrusion data are often scarce and expensive to obtain, it is not the optimal choice to make. What if the algorithm missed vital intrusion data while sampling? Also, sampling prior to training to include all intrusion data is impractical for two reasons: the first is that the distribution of the data changes (the training data will no longer reflect the actual problem), and second and more importantly, it is not applicable in practice.

Caching is a family of batch methods that loop multiple times over the training data to store a subset of 'important' data. The decision rule in the testing phase is then a combination of this subset. While the training progresses faster than non-caching methods, multiple passes through the data are still required, making caching techniques impractical on very large data.

Online algorithms, which can be forced to be one-pass, see a new point at every iteration and discard it after making use of it. Naturally, algorithms that need one pass through the data and do not store any points will be much faster and require much less memory than batch algorithms. However, as one would expect, such algorithms will not necessarily generalise as well as algorithms that have the luxury of going through

the data as many times as they need to obtain a solution. But for some tasks, such as the computer intrusion detection one, this tradeoff between performance and speed is critical. For a system administrator, a quick, rough answer on whether a connection is malicious or not is much more valuable than a late but 100% accurate one. What can a system administrator do to prevent an attack if the IDS tells him a day late that an intrusion has occurred?

Another important issue in the machine learning field, that has only recently gained popularity after the NIPS 2006 workshop entitled "Learning when test and training inputs have different distributions", is distribution shift. Classic machine learning techniques assume that the training and test data are drawn from the same underlying distribution. This i.i.d. assumption does not necessarily hold in practice, and has been largely ignored by the community until recently. This problem arises naturally in intrusion detection tasks where the conditions under which the IDS was developed will most likely be different than the one in which it will be applied and will need to be adapted with time. The reason for this is that data changes over time; new intrusions emerge (which render signature-based detectors useless), normal data changes (affecting the quality of anomaly detectors), and intruders in general constantly look for ways to bypass IDSs. This problem can be seen as a case of distribution shift, where the joint distribution of inputs and outputs differs between the training and testing stages. Textbook predictive machine learning models, however, work by ignoring these differences. Therefore, the performance of IDS' based on these machine learning methods will certainly degrade, and better choices can be made.

This distribution shift is also prominent in the field of Brain-Computer Interfaces (BCI). BCIs allow their users to control specially designed computer applications using only the power of thought. The greatest benefit of this comes in medical applications, where a paralysed yet cognitively able person can make use of this new communication channel by using the EEG signals generated by their brain. However, the way data are collected causes an inherent shift between their training and test sets: training data are collected by requiring the subject to perform certain tasks on a specific day whereas data collected some time later from the same subject performing the same tasks forms the test set. The subject could be in a different state of mind (e.g. with respect to fatigue, motivation etc.), which would directly affect the signals recorded from the brain. As an example, the EEG signals in the training set that correspond to lifting your left hand would differ from the EEG signals for the same motion in the test set. This is an issue that has plagued the BCI field, and yet practitioners do not explicitly try to tackle it. Instead, they apply linear classifiers to their data to avoid overfitting the training data, in the hope that it would generalise well to their test data.

1.1 KDD '99 Cup Data

An important data set we will use in our empirical study is the KDD '99 Cup data. This is because it is a real-world data set that captures the two main issues we aim to tackle in this thesis: its very large size ($\sim 5 \times 10^6$ training and 311,029 testing points), and the fact that it contains a distribution shift between the training and test phases, which we verify using a Kolmogorov-Smirnov test for two independent samples in Section 5.1.1.

This data set was created by Lincoln Labs at MIT and used for the KDD '99 competition, and has been widely used since for testing and evaluating algorithms. The task for the KDD '99 Cup was to build a classifier capable of distinguishing between legitimate and illegitimate connections in a computer network. Lincoln Labs set up an environment to acquire nine weeks of raw TCP dump data for a LAN simulating a typical US Air Force LAN, but peppered it with multiple attacks. Seven weeks of data were used for training, while two weeks worth of data acquired at a different time interval were used for testing. A connection is defined as a sequence of TCP packets starting and ending at well defined times, between which data flows to and from a source IP address to a target IP address under some well defined protocol. The data set has 41 features, shown in Table 1.1 and Table 1.2. The first nine features were basic features of individual TCP connections, extracted from the packet headers. The next thirteen (10–22) are content attributes, extracted from the contents of the network packets based on expert human knowledge. The next nine (23–31) are time-traffic attributes, of which there are two categories: the "same host" features examine the connections in the past two seconds that have the same destination host as the current connection, while the "same service" features examine the connections in the past two seconds that have the same service as the current connection. These time-traffic attributes, however, are not able to accurately capture all types of intrusions. For instance, some probing attacks scan their target hosts at larger time intervals (e.g. once per minute). To account for this, connection records were sorted by destination host, and features were constructed using a window of 100 connections to the same host instead of a time window. These host-based traffic features (32–41) are shown in Table 1.2

Feature Number	Feature name	Description	Type
1	duration	length (number of seconds) of the connection	continuous.
2	protocol_type	type of the protocol, e.g. tcp, udp, etc.	symbolic.
3	service	network service on the destination, e.g., http, telnet, etc.	symbolic.
4	flag	connection status	symbolic.
5	src_bytes	number of data bytes from source to destination	continuous.
6	dst_bytes	number of data bytes from destination to source	continuous.
7	land	1 if connection is from/to the same host/port; 0 otherwise	binary.
8	wrong_fragment	number of "wrong" fragments	continuous.
9	urgent	number of urgent packets	continuous.
10	hot	number of "hot" indicators	continuous.
11	num_failed_logins	number of failed login attempts	continuous.
12	logged_in	1 if successfully logged in; 0 otherwise	binary.
13	num_compromised	number of "compromised" conditions	continuous.
14	root_shell	1 if root shell is obtained; 0 otherwise	binary.
15	su_attempted	1 if "su root" command attempted; 0 otherwise	binary.
16	num_root	number of "root" accesses	continuous.
17	num_file_creations	number of file creation operations	continuous.
18	num_shells	number of shell prompts	continuous.
19	num_access_files	number of operations on access control files	continuous.
20	num_outbound_cmds	number of outbound commands in an ftp session	continuous.
21	is_host_login	1 if the login belongs to the "hot" list; 0 otherwise	binary.
22	is_guest_login	1 if the login is a "guest"login; 0 otherwise	binary.
23	count	number of connections to the same destination IP address	continuous.

TABLE 1.1: The first 23 features of the KDD '99 Cup data

Feature Number	Feature name	Description	Type
24	srv_count	number of connections to the same destination port number	continuous.
25	serror_rate	% of connections that have "SYN" errors among connections aggregated in count (23)	continuous.
26	srv_serror_rate	% of connections that have "SYN" errors among connections aggregated in srv_count (24)	continuous.
27	rerror_rate	% of connections that have "REJ" errors among connections aggregated in count (23)	continuous.
28	srv_rerror_rate	% of connections that have "REJ" errors among connections aggregated in srv_count (24)	continuous.
29	same_srv_rate	% of connections to the same service	continuous.
30	diff_srv_rate	% of connections to different services	continuous.
31	srv_diff_host_rate	% of connections to different hosts	continuous.
32	dst_host_count	number of connections to the same destination IP address	continuous.
33	dst_host_srv_count	number of connections to the same destination port number	continuous.
34	dst_host_same_srv_rate	% of connections that were to the same service, among the connections aggregated in dst_host_count (32)	continuous.
35	dst_host_diff_srv_rate	% of connections that were to different services, among the connections aggregated in dst_host_count (32)	continuous.
36	dst_host_same_src_port_rate	% of connections that were to the same source port, among the connections aggregated in dst_host_srv_count (33)	continuous.
37	dst_host_srv_diff_host_rate	% of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count (33)	continuous.
38	dst_host_serror_rate	% of connections that have "SYN" errors among the connections aggregated in dst_host_count (32)	continuous.
39	dst_host_srv_serror_rate	% of connections that have "SYN" errors among the connections aggregated in dst_host_srv_count (33)	continuous.
40	dst_host_rerror_rate	% of connections that have "REJ" errors among the connections aggregated in dst_host_count (32)	continuous.
41	dst_host_srv_rerror_rate	% of connections that have "REJ" errors among the connections aggregated in dst_host_srv_count (33)	continuous.

TABLE 1.2: The last 18 features of the KDD '99 Cup data

This data set is now considered the de facto data set for intrusion detection (Eskin et al. (2002); Laskov et al. (2005); Katos (2007)), and researchers wishing to evaluate new methods or algorithms usually use it; it is very large ($4,898,424$ training examples, and $311,029$ test examples), has new classes in the test set not present in the training set, the competition's winners are available for comparison purposes, and it is probably the best ID data set among the very few publicly available ones. Since many researchers have also used this data set, it is possible to benchmark a new algorithm against others. The connections in the data set are either normal connections or intrusions, of which there are four main categories:

- Probing (surveillance, port scanning, etc.)
- DoS (Denial of Service)
- U2R (unauthorised access to local superuser privileges)
- R2L (unauthorised access from remote machine)

The training set, however, is far too large to use for the majority of available techniques, such as the standard SVM and batch techniques that need multiple passes through the data. Therefore, many authors use the publicly available 10% random subset for training. The distributions of the different connections in this 10% training set, and the supplied test set are given in Table 1.3.

	Training set	Testing set
Normal	19.69%	19.48%
Probing	0.83%	1.34%
DoS	79.24%	73.90%
U2R	0.01%	0.07%
R2L	0.23%	5.20%

TABLE 1.3: Distribution of connections in the supplied 10% KDD '99 subset

An interesting point about this data set is that some attack categories have very little training data, and so performance on these attacks is usually poor. Also, there are some intrusion types in the test set that are not present in the training set. This causes a shift in the underlying distribution of the data, which is useful for developing and testing algorithms that handle distribution change. We tackle this distribution shift, which we verify using a formal statistical test, in Chapter 5.

1.1.1 Evaluation Criteria

The performance measure used for the KDD '99 Cup competition was the average cost-per-test-example, which is the average misclassification cost incurred per test example

Predicted → Actual ↓	Normal	Probe	DoS	U2R	R2L
Normal	0	1	2	2	2
Probe	1	0	2	2	2
DoS	2	1	0	2	2
U2R	3	2	2	0	2
R2L	4	2	2	2	0

TABLE 1.4: The cost matrix used for scoring entries in the KDD '99 competition

using the cost matrix provided (see Table 1.4). When testing is complete, and the confusion matrix obtained, we multiply the corresponding elements (of the cost and the confusion matrices) and sum them. We then divide this sum by the number of test points to obtain the average cost-per-example. Other measures were the probability of detection (PD) and the false alarm rate (FAR). The former is the probability of detecting an intrusion, whereas the latter is the rate at which normal connections are being flagged as anomalous. Using Figure 1.1, we define PD and FAR as:

$$\text{PD} = \text{TP}/(\text{TP} + \text{FN}) \tag{1.1}$$

$$\text{FAR} = \text{FP}/(\text{FP} + \text{TN}) \tag{1.2}$$

However, after the competition, the PD and FAR were not widely used any more, and were replaced by the more popular prediction accuracy. This is defined as

$$\text{Prediction Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Despite this, we use the PD, the FAR, the average cost-per-example, as well as the prediction accuracy in our empirical study to compare our results on the multi-class KDD data with published results.

We also use ROC curves, which show how the number of correctly classified positive examples varies with the number of incorrectly classified negative examples for all possible threshold values. The curve is generated by plotting the true positive rate against the false positive rate (See Figure 1.1 and Figure 1.2). Each point on the ROC curve represents the combination of true positives and false positives at a given threshold value.

The advantage of the ROC over prediction accuracy is that the latter depends on a specific threshold, while the former removes the effect of this choice of the threshold, and gives more general results. An ROC curve demonstrates several things:

- the closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test. This makes the ideal case: TP = 1, FP = 0.

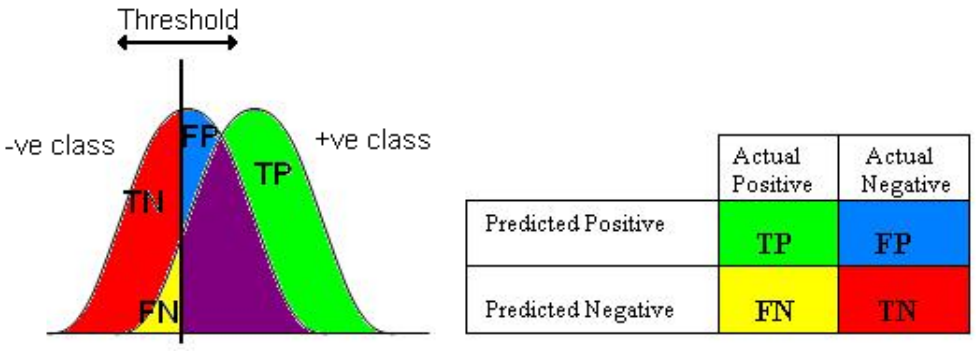


FIGURE 1.1: Confusion matrix and its representation on overlapping distributions and a classification threshold. Anything to the right of the threshold is classified as +ve, and anything to the left as -ve.

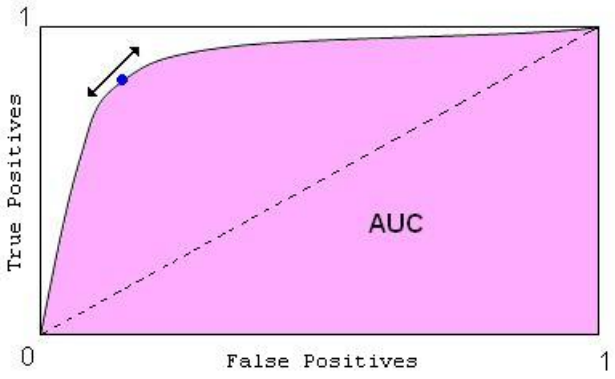


FIGURE 1.2: ROC curve and AUC. The dashed diagonal line represents a random classifier.

- the closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test. The 45-degree diagonal represents a random classifier.
- the area under the curve (AUC), which is a measure of test accuracy. It represents the probability that the classifier will assign a higher score to a randomly chosen positive point over a randomly chosen negative point.

1.1.2 Preprocessing

Before using this data set for our empirical work, we had to preprocess it to make it suitable for our algorithms to work on. There were symbolic features that needed to be converted to numerical ones, and continuous variables that needed to be normalised. The binary features were not preprocessed. Only features two, three, and four were symbolic, representing the protocol type (with three possible values), service (with 68 possible values), and the flag (with 11 possible values) respectively (see Table 1.1). Of course, the class of the connection was symbolic as well, with one normal class and 24 different attack types plus an additional 14 attacks in the test set not present in the

training set. The mappings performed on the classes are shown in Table 1.5. As an example, any of the 24 + 14 attack types that belong to the DoS class were mapped to class 2. To compare our techniques presented in the thesis with others in the field who used a two-class classification algorithm (SVMs for example), we consider all attack connections (i.e. classes 2, 3, 4, and 5) to belong to class '-1' and normal connections to belong to class '1'.

Symbolic value	Mapped to in our data
normal	1
Probing	2
DoS	3
U2R	4
R2L	5

TABLE 1.5: Mapping KDD '99 classes to numerical values

For the continuous variables, we normalised them to $[0, 1]$ by subtracting the minimum of the feature from the value, and dividing by the range of the feature. As an example, to normalise value1, which is in the 7th column, we perform:

$$\text{value1}' = \frac{\text{value1} - \min(\text{column7})}{\max(\text{column7}) - \min(\text{column7})}.$$

1.1.3 The KDD '99 Cup winner

In this section, we introduce the winning entry of the KDD '99 competition, with whom researchers compare their work when using this data in their empirical work. The winner was Pfahringer (2000), who used bagged boosting and decision trees. In his very short and informal paper, he briefly describes how he built his predictor. Initially, he experimented with several algorithms such as naïve Bayes, nearest neighbours, neural networks etc., and ruled out the algorithms requiring excessive runtime. Due to various resource limitations, he constructed an ensemble of 50×10 decision trees in such a way, that it can be described as cost-sensitive bagged boosting:

1. 50 samples drawn from the original data set (around 5 million) in a biased manner; all examples of the two smallest classes (U2R and R2L) were included, and 4000 probe, 80000 Normal, and 400000 DoS examples. Also, duplicate entries in the original data set were removed beforehand.

2. For each of the 50 samples, an ensemble of 10 (C5[3]) decision trees was induced using both C5's error-cost and boosting options.

3. The final predictions were computed on top of the 50 predictions of the sub-ensembles by minimising the conditional risk. This risk is defined as the sum of

[3]http://www.rulequest.com/see5-info.html

the error-costs predicting specific classes times the probabilities of the respective classes.

The problem with this method is the way the author subsampled his data. Given the biased sample he used, comparing his result with others that used the publicly available 10% subset is not fair. The advantage, however, was that his method was able to perform better on the two smallest classes that he oversampled (U2R and R2L). Because the competition was scored using a cost matrix, Pfahringer (2000) was able to achieve a lower error score than others, since misclassifying U2R and R2L carried the heaviest penalty. His technique required over a day to train.

1.1.4 The runner-up

The runner up to the KDD '99 Cup was the Kernel Miner by Levin (2000). He used the kernel miner tool to create a decision forest, which corresponds to the set of locally optimal decision trees. The optimal subset of trees (called the sub-forest) was then selected to predict new cases. The predictive modeling technique is based on sophisticated methods for reducing the individual prediction results received from individual classification trees. This allows the calculation of the value of the global optimisation criterion for any subset of trees. This criterion includes, for example, minimising the overall number of misclassifications, while taking parameters of stability and reliability into account (to avoid over-fitting). Levin (2000) used a multi-class detection approach to discover not only that there is an attack, but the categories of those attacks. For this, he used the publicly available 10% subset of the KDD '99 training data to train his model. The final sub-forest gave relatively impressive detection rates for all the attack categories in the testing data set (see Table 1.6). Most of the mistakes were made on the attack categories that were not present in the training set.

		Probe	DoS	U2R	R2L
Pfahringer (2000)	PD	0.833	0.971	0.132	0.084
	FAR	0.006	0.003	0.00003	0.00005
Levin (2000)	PD	0.845	0.975	0.118	0.0732
	FAR	0.216	0.731	0.364	0.017

TABLE 1.6: Top results on the KDD '99 Cup data

1.1.5 Criticism of the KDD '99 Data

In 2007, Brugger (2007) wrote a position letter criticising the KDD '99 data, questioning how close to real network traffic these data sets were. He draws his conclusions from McHugh (2000), who stated that the data sets were released without validating whether

this data reflected real network traffic or not. The issue with such criticisms is that any such data has to be generated artificially in a simulated environment, as it is impossible to know whether connections on a real-world network are benign or malicious. If this was possible, then there would be no reason for research in intrusion detection to continue. Researchers in the network security community acknowledge McHugh (2000)'s concerns, but continue to use this data anyway because, according to Kayacik et al. (2007), this benchmark provides the only labeled data set for comparing IDS systems, enabling the establishment of a clear comparison with alternative solutions trained on the KDD data set. This is echoed by many authors, and another example is Giacinto et al. (2008), who justify intrusion detection researchers' continued use of this data set, despite McHugh (2000)'s criticisms, because it is the only reference data set that allows the designers to compare results obtained using different intrusion detection techniques. Similar comments are made by all the authors that use this data, such as by Chinchani et al. (2004); van Oorschot et al. (2006); Rieck and Laskov (2007); Marin-Blazquez and Perez (2008).

The effect of McHugh (2000)'s paper (which he published right after the KDD '99 competition) on the data's use was not significant; had researchers taken his concerns seriously, there would have been far fewer publications using this data for evaluation than currently exist. A quick search for papers that use the KDD '99 intrusion detection benchmark indicates there have been over 1,500 publications using this data set since 2000. In our case, whether this data reflects real network traffic accurately or not is beyond the scope of this thesis. The KDD '99 data is among the largest publicly available data sets, and we use it for the purpose of evaluating the scalability of our novel techniques in comparison with other published methods. The evaluation of our novel algorithms was conducted on bioinformatics data sets (in Chapter 3), and on 18 benchmark data sets by Rätsch et al. (2001) and the UCI repository (in Chapter 4). Using these data sets, we show how our novel techniques achieve state-of-the-art performance in a fraction of the time. However, these problems are relatively small, and instead of using an artificially generated data set with $\sim$ 5 million points to highlight the computational savings made by our techniques, it is more rigorous and scientific to use a data set that has been used extensively, for over a decade, by the community for evaluation purposes. Very few authors use the entire training data set, given its massive size, opting to use the publicly available 10% subset instead. We use this fact to our advantage to highlight the benefits of our one-pass method, which took less than five minutes to partition the input space using $\sim$5 million points, and achieves state-of-the-art performance. To this end, regardless of what the data represents or how accurately it does so, results by different authors on the same data constitute a fair comparison. We further use subsets of this data in Chapter 5 (along with artificial data sets) to show the effect of a distribution shift on the performance of classifiers that don't explicitly account for it. The distribution shift in the KDD '99 data is undisputable; not only was it mentioned by the data's creators, but we formally confirmed it using a Kolmogorov-

Smirnov test in Section 5.1.1. Again, the claims of Brugger (2007) and McHugh (2000) are irrelevant in this case, as it is a data set that contains a shift, and we test our algorithms on the same data, which is a fair comparison between our unweighted and weighted classifiers. For extra validation, we do not simply rely on the KDD '99 data for solving the distribution shift issue, we use a more difficult problem domain in Chapter 6: Brain-Computer Interfaces (BCI). The difficulty arises from the fact that these data sets are very small, which is challenging to the weighting algorithms we use in our framework. We compare the results of our weighted techniques with all the submissions of the competition from which we obtained the BCI data, further demonstrating the robustness of our presented methods.

1.2 Thesis Outline

Using the intrusion detection and BCI problems, we aim to tackle the issues of large scale learning and shifting distributions simultaneously, with the thesis being organised as follows: In Chapter 2, we review some of the recent and most relevant algorithms for large scale learning. This review is split into three main sections: Section 2.1 describes clustering techniques that will be used in Chapter 3, Section 2.2 contains the batch algorithms, while Section 2.3 contains the online methods. We then introduce, in Chapter 3, a novel online hierarchical clustering technique which has applications on large scale problems, such as bioinformatics and computer intrusion detection. In Chapter 4, we introduce a novel one-pass, non-linear, multi-class classification technique called Voted Spheres that achieves state-of-the-art performance in a fraction of the time required by other techniques. Chapter 5 deals with the problem of data distribution shift in machine learning, with the two most recent techniques for dealing with this shift successfully incorporated into VS and logistic regression. In our final contributory chapter (Chapter 6), we apply the VS and the modified version from Chapter 5 to the Brain-Computer Interface domain, achieving state-of-the-art results on the data sets used. The work in this chapter demonstrates the robustness of our presented techniques in different domain areas. Finally, in Chapter 7, we conclude the thesis by summarising our results and highlighting the most important and significant contributions of the thesis, and discussing future directions to pursue.

1.3 Contributions

- We introduce a novel online hierarchical pattern clustering technique (see Chapter 3). This is an extremely fast clustering technique, which, after constructing an initial tree from a small sample of the data, sequentially inserts the remaining points one at a time as they arrive. Our results show that the quality of clusters

does not degrade, while making significant computational savings compared to other state-of-the-art techniques. Associated publication: "Farran, B., Ramanan, A. and Niranjan, M., Sequential Hierarchical Pattern Clustering. In: Pattern Recognition in Bioinformatics (PRIB), LNBI 5780 pp. 79-88. 2009"

- We introduce a novel, non-linear, one-pass, multi-class classification algorithm that we call Voted Spheres (see Chapter 4), along with several modifications to it. This algorithm is extremely fast (takes two seconds to train and test on $20,000$ points, and five minutes to train and test on the KDD '99 Cup's $\sim 5 \times 10^6$ training points and $311,029$ test points). It also generalises very well, achieving state-of-the-art performance in a fraction of the time required by other techniques. This is also, to the best of our knowledge, the only time a fully online algorithm was applied on the intrusion detection task with such success. Associated publication: "Farran, B. and Saunders, C., Voted Spheres: An Online, Fast Approach to Large Scale Learning. In: Advanced Information Networking and Applications Workshops '09. pp. 744-749. 2009"

- We adapt the Voted Spheres framework to incorporate the presence of a covariate shift in the data using two different techniques (see Chapter 5). This stems from the fact that many real-world data sets, such as the KDD '99 Cup data, contain a distribution shift between training and test phases. This increased the Voted Spheres' generalisation ability, further outperforming the state-of-the-art methods that never explicitly take distribution shift into account. We also perform a Kolmogorov-Smirnov two-sample test on the KDD '99 Cup data in order to verify the existence of a shift. This is, to the best of our knowledge, the only formal test to verify the existence of a shift in the KDD '99 data that has been conducted. Associated publication: "Farran, B., Saunders, C. and Niranjan, M., Machine Learning for Intrusion Detection: Modeling the Distribution Shift. In IEEE Workshop on Machine Learning for Signal Processing (MLSP), 2010. [Accepted]"
  Journal submission in progress to IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics.

- We apply the Voted Spheres technique along with its covariate shift-adapted counterpart on the Brain-Computer Interface domain[4] (see Chapter 6). The goal of the "BCI Competition III" is to validate signal processing and classification methods for Brain-Computer Interfaces (BCIs). This poses a challenge to our Voted Spheres, and its distribution-shift-adapted counterpart, as very little training data is available. Because of this, researchers generally apply batch algorithms such as SVMs, and the successful application of a one-pass technique that takes distribution shift into account is a contribution to the BCI field in its own right. Results show the robustness of the VS framework to different application areas, as

[4] http://www.bbci.de/competition/iii/

it outperforms the SVM (linear and RBF), and performs competitively with all the published results on the data sets used. Journal submission to IEEE Transactions on Biomedical Engineering in progress.

Chapter 2

Literature Review

In this chapter, we present a review of the most relevant machine learning algorithms for large scale learning. A lot of the reviewed techniques will be used in our empirical study in the rest of the thesis. First we discuss clustering methods, starting from the most widely used k-means, and continue to describe the different types of hierarchical methods, which we will use to construct our initial hierarchical tree in Chapter 3. We then describe batch classification algorithms, and bring to light their drawbacks for large scale learning. More specifically, we present the state-of-the-art SVM, which we will use extensively for comparison purposes. We then present some recent attempts at scaling the SVM to large data sets, which we will empirically evaluate using the very large KDD '99 data in Chapter 4. Finally, we present online methods, which are a family of algorithms that learn one data point at a time as they arrive. These algorithms are not necessarily one-pass, however we are able to convert them to one-pass by forcing them to terminate after one run. These will be compared with our novel one-pass technique in Chapter 4.

2.1 Clustering Algorithms

Perhaps the simplest and most well-known clustering technique is the k-means. It assumes the number of clusters k is known a priori, and chooses k random points as initial centroids. Then, the remaining points in the data set are assigned to the nearest centroid. When all the points have been assigned, the centroids are recalculated as the arithmetic means of all the points assigned to them. We repeat the above steps, looping through the data and assigning them to the closest clusters and recalculating the centroids until the centroids do not change significantly. The performance of k-means depends heavily on the initial centroids chosen. A bad choice could result in the algorithm getting stuck in local minima. To this effect, there have been certain attempts to choose "better" initial centroids, such as by Fahim et al. (2009). Another issue, which

is of higher interest to this thesis, is the multiple passes required through the data and the fact that at each step, the distance from each point to each cluster has to be calculated. The time required for it to converge would render it useless on our very large scale intrusion data.

Other clustering techniques are the agglomerative and divisive ones that produce dendrograms. These techniques are also called hierarchical algorithms, and are based in measures of dissimilarity among the current cluster set in each iteration. Agglomerative algorithms merge clusters together based on their similarities, while divisive algorithms use the dissimilarities to split clusters. The produced dendrogram has N leaf nodes or 'singleton' clusters (corresponding to the input data) and $N-1$ inner nodes (clusters) that represent groupings in coarser granularities at higher levels in the tree, and can be cut at any level to obtain different numbers of clusters. The number of clusters can also be determined as a function of a merging threshold (Fung). If this threshold was zero the number of clusters would be equal to the number of data points passed to the algorithm, and by increasing this threshold the number of clusters decreases eventually reaching one single cluster. According to Fasulo (1999), the biggest drawback to such clustering is that a data point is not allowed to change cluster membership once it has been assigned. Another issue is that these methods usually require the entire pairwise dissimilarity matrix to be precomputed and stored in advance. This can be very time consuming for large data sets, and the memory requirements ($O(N^2)$) would definitely limit the algorithm. The advantage, which applies to all non-parametric methods, is that no assumption on the underlying data distribution is made.

Given that clustering is unsupervised, it is mainly applied in fields that require some pattern to be extracted from the data. A very good example of this is bioinformatics, which has seen a rise in the use of clustering techniques to extract information from data. Different bioinformatics tasks require the use of different types of clustering. For instance to represent protein sequence family relationships, Kaplan et al. (2004) use a hierarchical clustering technique to achieve their goal. Another example is by Eisen et al. (1998), who applied a variant of the hierarchical average-linkage clustering algorithm to identify groups of co-regulated genome-wide expression patterns. Today, researchers working on these tasks are faced with ever-increasing amounts of data. To tackle this issue, some have attempted to develop more scalable algorithms. For instance, Frey and Dueck (2007) use an affinity propagation clustering technique to detect putative exons comprising genes from mouse chromosomes. However, while they claim lower computational cost in comparison to other algorithms, they omit the fact that their method requires the entire pairwise similarity matrix in advance. Since this is the most expensive (and therefore prohibitive) stage in large scale problems, the claimed advantage is actually exaggerated (see Section 3.3.2).

Another method that aims to reduce the computational complexity of hierarchical clustering is by El-Sonbaty and Ismail (1998). They proposed an online hierarchical cluster-

ing algorithm based on the single-linkage method that finds, at each step, the nearest k patterns with the arrival of a new pattern and sorts them by similarity (or dissimilarity) from closest to furthest. These nearest k objects are continuously updated upon arrival of a new point. By the end of the training phase, what is obtained is a table requiring $O(N * k)$ memory, one entry per training example, with k columns indicating the k nearest points to each training example. From this table of ordered pairings, the hierarchical dendrogram can be drawn by iteratively merging the objects contained at the same pair. This should be done starting from the first pair with the minimum distance, all the way until all N input patterns are covered. While they claim their method is sequential, at the arrival of each data item they compute similarity to all the data seen previously. Thus there is little computational saving in their method, and it is equivalent to re-training a new model at the arrival of new data. Furthermore, like most machine learning algorithms, the proposed algorithm is sensitive to the similarity/dissimilarity measure used, as different distance measures can create different hierarchical dendrograms for the same input data set. Finally, there is no clear way to choose k; the authors use trial and error to obtain the optimal value. We later show in our empirical study (in Section 3.2) that this algorithm does not produce good trees, and therefore degrades the performance of our novel Sequential Hierarchical Pattern Clustering (SHPC) method presented in Chapter 3.

The most commonly used hierarchical clustering formulation is the average linkage by Sokal and Michener (1958), which is more commonly referred to as the Unweighted Pair Group Method with Arithmetic Mean (UPGMA). This formulation uses the average similarity across all cluster data points, making it more robust and practical than its single linkage counterpart. This is due to the stability of the arithmetic mean. UPGMA takes as an input the pairwise dissimilarity (i.e. distance) matrix of all input patterns, without self loops (i.e. dissimilarity of a point to itself). During training when clusters are being merged, the possible multiple edges between pairs of non-singleton edges are averaged and are referred to as thick edges. These edges are cluster-pair unique. More sophisticated algorithms exist for constructing hierarchical trees, however as Lazareva-Ulitsky et al. (2005) point out, UPGMA's runtime scales well with the size of the input data and is the preferred algorithm for clustering large data sets. The drawback to this technique is that it requires the entire similarity matrix to be present in memory. This makes it impossible to run on large data sets, as it requires $\mathrm{O}(N^2)$ memory. Even though in our empirical study we will run an initial hierarchical clustering technique on a small fraction of our data, say 5% of our total number of points, it could still be prohibitively large for very large N.

To deal with this, Loewenstein et al. (2008) attempt to develop a framework for finding the correct UPGMA tree for very large data, which is not limited by memory requirements. They devise the Single-Round Memory-Constrained UPGMA which holds only a subset of the dissimilarity matrix in memory, and outputs successive parts of the overall

hierarchy. The entire data set is clustered in a single round. We use this version of the algorithm in our empirical study in Section 3.2 and Section 3.3.

2.2 Batch Algorithms

The k-Nearest Neighbours (k-NN) is among the simplest of the machine learning techniques in the literature. It is a non-parametric decision rule in which a new test sample is assigned to the class to which the majority of its k-nearest neighbouring labeled examples are assigned. The algorithm is a lazy learner, deferring all the computation to the testing phase. A serious disadvantage of this method is that it does not simplify the distribution of the training points to something more manageable. The training stage involves simply storing the training data ($O(N)$). This means the testing phase requires $O(M.N)$ calculations, where M is the size of the test set. This is because the k-NN needs to calculate the distance between the current test point $\hat{\mathbf{x}}_j$ and all N input points. It is still widely used in the machine learning literature because of its effectiveness when the probability distributions of the feature values are not known. However, due to the expensive nature of the testing phase, it is not practical for application on very large data sets.

The edited k-Nearest Neighbour rule for classification, introduced by Wilson. (1972), is partly motivated by the need to reduce the computational complexity of k-NN at run time. The objective is to design a reduced reference of 'clean' data by a process of backward deletion of data. A family of such algorithms exists in the literature, and below we describe a recent one by Hattori and Takahashi (2000). They propose a new edited k-nearest neighbour (k-NN) rule for classification, where higher classification accuracy is favoured over decreasing the number of samples in the reference set. Hattori and Takahashi (2000)'s method involves taking each input point $\mathbf{x}$ one at a time and retrieving its k or $k + l$ nearest neighbours, where l is the number of points that are equidistant to $\mathbf{x}$ as the kth nearest neighbour of $\mathbf{x}$. This input point is retained for the final reference set if the $k + l$ examples belong to the same class as $\mathbf{x}$. During the testing phase, each test point is classified using the regular k-NN rule using the edited reference set. We critically evaluate this algorithm and compare it to our novel approach in Section 4.3.

A powerful technique, which was also used by the KDD '99 Cup winner, is boosting. We describe this method since we compare our multi-class Voted Spheres algorithm in Chapter 4 with three boosting-based algorithms: Pfahringer (2000)'s method, MC-SLIPPER by Yu and Tsai (2004), and a recent AdaBoost-based method by Hu et al. (2008). Boosting is a general method of converting rough rules of thumb into a highly accurate classifier. It occurs in stages by incrementally adding knowledge to the learnt function. At every stage, a weak learner is trained with the data, where a weak learner is

an algorithm which can consistently find classifiers at least slightly better than random guessing.

The AdaBoost (Adaptive Boosting) algorithm by Freund and Schapire (1994, 1996) is generally considered as a first step towards more practical boosting algorithms (Meir and Rätsch (2003)). It is a meta-algorithm and can be used in conjunction with many other learning algorithms to improve their performance. AdaBoost is adaptive in the sense that subsequent classifiers built are tweaked in favour of those instances misclassified by previous classifiers. On the downside, AdaBoost is sensitive to noisy data and outliers, but otherwise, it is less inclined to over-fit the data than most other learning algorithms. The algorithm is shown in Algorithm 1. According to Schapire (1999), if each weak hypothesis is slightly better than random, then the training error drops exponentially fast. Also, the equation to update the distribution D_t is constructed so that after selecting the optimal classifier h_t for the distribution D_t, the examples $\mathbf{x}_i$ that the classifier h_t identified correctly are weighted less and those that it identified incorrectly are weighted more. Therefore, when the algorithm is testing the classifiers on the distribution D_{t+1}, it will select a classifier that better identifies those examples that the previous classifier missed. Practically, AdaBoost has many advantages. It is simple, fast and easy to program. The algorithm does not have parameters that need tuning (except for the number of rounds T). Also, it does not need prior knowledge about the weak learner, and so can be combined with any method for finding weak hypotheses. Furthermore, it comes with a set of theoretical guarantees, given that there is sufficient data and a weak learner that can reliably provide only moderately accurate weak hypotheses. Since AdaBoost focuses

Algorithm 1 The AdaBoost algorithm

Input: $(\mathbf{x}_1, y_1)$, ..., $(\mathbf{x}_N, y_N)$ where $\mathbf{x}_i \in X$, $y_i \in Y = \{-1, 1\}$, T: number of rounds

Initialise: Initialise $D_1(i) = \frac{1}{N}$ for i=1, ..., N

for $t = 1, 2, \cdots, T$ **do**

- Train weak learner using distribution D_t
- Get weak hypothesis $h_t : X \to \{-1, 1\}$ with error $\epsilon_t = \Pr_{i \sim D_t}[h_t(\mathbf{x}_i) \neq y_i]$
- $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
- Update:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(\mathbf{x}_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(\mathbf{x}_i) \neq y_i \end{cases}$$

$$= \frac{D_t(i) \exp\left(-\alpha_t y_i h_t(\mathbf{x}_i)\right)}{Z_t}$$

where Z_t is a normalisation factor (chosen so that D_{t+1} will be a distribution)

end for

Output: $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

its weight on the hardest examples, the examples with the highest weight often turn out to be outliers. On the other hand, the actual performance of boosting on a particular problem is clearly dependent on the data and the weak learner (Freund and Schapire (1999)). This is consistent with theory, as boosting can fail to perform well given insufficient data, overly complex weak learners, or weak learners that are too weak as shown by Wickramaratna et al. (2001). Furthermore, experimental results show boosting to be susceptible to noise. As Diettrich (2000) demonstrated, the emphasis placed on the hard examples can harm the performance of AdaBoost when the number of outliers is large. To deal with this, a variant of AdaBoost has been suggested by Friedman et al. (1998), called "Gentle AdaBoost" which places less emphasis on outliers.

Given that kernel methods have grown in popularity over the last decade, and are currently considered the state-of-the-art, we will use them in our empirical study in the rest of the thesis. Kernels are used as part of the 'kernel trick', which is a method for using a linear classifier to solve a non-linear problem. This is done by mapping the original non-linear points into a higher-dimensional space, where the linear classifier is subsequently used; this makes a linear classification in the new space equivalent to non-linear classification in the original space. This is done by replacing dot products with

$$k(\mathbf{x}, \mathbf{v}) = \langle \phi(\mathbf{x}), \phi(\mathbf{v}) \rangle$$

where $\mathbf{x}, \mathbf{v} \in \mathcal{X}$ and ϕ is a mapping from $\mathcal{X}$ to a feature space F. The input space is defined with $\mathcal{X}$, and the feature space is

$$F = \{\phi(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}.$$

However, the use of kernels allows us to never explicitly compute the ϕ function. This is advantageous because the high-dimensional space may be infinite-dimensional.
Two popular kernel functions are listed below:

- Gaussian Radial Basis Function: $k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right)$, where σ is the width of the kernel
- Polynomial Function: $(\langle \mathbf{x}, \mathbf{x}' \rangle + 1)^d$, where d is the polynomial degree.

The SVM is the most well-known technique that uses the kernel trick. SVMs are based on the concept of finding the optimal hyperplanes separating data from different classes (Boser et al. (1992); Cortes and Vapnik (1995); Vapnik (1998)). An optimal separating hyperplane is a hyperplane that separates the data from different classes and also maximises the margin (See Figure 2.1). More formally, consider training data of the form: $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N)\}$ where $\mathbf{x}_t$ is a d-dimensional real vector and y_t is either -1 or 1 (the class of $\mathbf{x}_t$ in a binary SVM). The separating hyperplane will take the form:

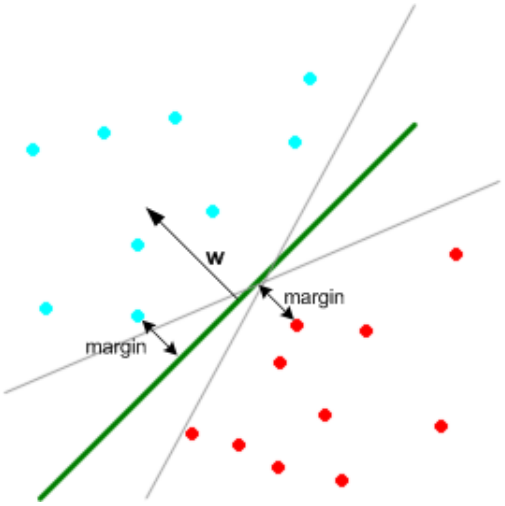


FIGURE 2.1: The optimal separating hyperplane, which maximises the distances of the closest points from each class (the margin).

$$\mathbf{w} \cdot \mathbf{x} + b = 0.$$

The vector $\mathbf{w}$ is the normal vector to the hyperplane. Without the offset b, the solution has to pass through the origin, restricting the solution. The input points are said to be optimally separated if they are separated without error, and the distance between the hyperplane and the points closest to it from either class is maximal. Without loss of generality, we consider a canonical hyperplane (Cortes and Vapnik (1995)) where the parameters $\mathbf{w}$ and b are constrained by

$$\min_i |\mathbf{w} \cdot \mathbf{x}_i + b| = 1.$$

This constraint simplifies the formulation of the problem, and means that the norm of the weight vector $\mathbf{w}$ should be equal to the inverse of the distance of the closest input point to the hyperplane. The separating canonical hyperplane should therefore satisfy the following constraints:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad i = 1, \cdots, N.$$

Also, the distance $d(\mathbf{w}, b; \mathbf{x})$ of a point $\mathbf{x}$ from the hyperplane defined by $(\mathbf{w}, b)$ is:

$$d(\mathbf{w}, b; \mathbf{x}) = \frac{|\mathbf{w} \cdot \mathbf{x}_i + b|}{||\mathbf{w}||.}$$

For a point $\mathbf{x}_i$ that lies on the hyperplane $\min_i |\mathbf{w} \cdot \mathbf{x}_i + b| = 1$, its distance d_+ to the hyperplane is $\frac{1}{||\mathbf{w}||}$. For a point $\mathbf{x}_j$ that lies on the hyperplane $\min_i |\mathbf{w} \cdot \mathbf{x}_i + b| = -1$, its distance d_- is also $\frac{1}{||\mathbf{w}||}$. This makes the margin equal to $d_+ + d_- = \frac{2}{||\mathbf{w}||}$. We therefore look for the pair of hyperplanes that give the maximum margin (by minimising $||w||$),

using the following optimisation in primal form:

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2$$

subject to

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

$$1 \leq i \leq N.$$

The problem can be transformed to its dual representation using Lagrange multipliers:

$$\max_{\alpha} \sum_{k=1}^{N} \alpha_k - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j {\mathbf{x}_i}^T \mathbf{x}_j$$

and the solution is therefore given by

$$\alpha^* = \arg\min_{\alpha} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j {\mathbf{x}_i}^T \mathbf{x}_j - \sum_{k=1}^{N} \alpha_k$$

subject to

$$\alpha_i \geq 0 \ \sum_i \alpha_i y_i = 0,$$

where the α terms constitute a dual representation for the weight vector in terms of the training set. The optimal separating hyperplane is given by

$$\mathbf{w}^* = \sum_i \alpha_i y_i \mathbf{x}_i b^* = -\frac{1}{2} \langle \mathbf{w}^*, \mathbf{x}_r + \mathbf{x}_s \rangle$$

where $\mathbf{x}_r$ and $\mathbf{x}_s$ are any support vectors from each class satisfying

$$\alpha_r, \alpha_s > 0, \ y_r = -1, \ y_s = 1.$$

Some α_i values are 0, which shows that classification is only a function of the support vectors (which are vectors $\mathbf{x}_i$ with corresponding $\alpha_i > 0$). This makes the hard classifier:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^* \cdot \mathbf{x} + b).$$

This linear classifier can be transformed into a non-linear one using the kernel trick, as discussed above.

There is also a soft-margin formulation for the SVM, which allows for mislabeled exam-

ples. If there is no hyperplane that can split the two classes, the soft-margin formulation will choose a hyperplane that splits the examples as cleanly as possible, while still maximising the distance to the nearest cleanly split examples. This is achieved by using a slack variable ξ_i, which measures the degree of misclassification of example $\mathbf{x}_i$. The quadratic optimisation problem in primal form becomes:

$$\min_{\mathbf{w}} \ \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i$$

subject to

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i$$

$$1 \leq i \leq N$$

$$\xi_i \geq 0$$

The algorithmic complexity of the quadratic programming ($O(N^3)$) and the memory required to store the kernel matrix ($O(N^2)$) means that the vanilla SVM cannot be applied on large-scale tasks where multiple scans of the data are too expensive to perform (Yu et al. (2003)). Also, care must be taken when selecting the kernel function, and the kernel function's parameters, as this greatly affects the performance, speed, and scalability. It is difficult for conventional kernel methods to run on very large scale data, since kernel methods such as the vanilla SVM need $O(N^2)$ memory and $O(N^3)$ time to train. Many researchers have attempted to overcome these limitations, and many techniques have emerged. We therefore searched the literature for kernel-based algorithms that are capable of handling larger data sets, or techniques that enable existing techniques to scale better. Some algorithms that were designed to scale to large data sets will be described in the rest of the section.

One such attempt was by Vishwanathan et al. (2003). They named their fast, iterative support vector training algorithm 'simpleSVM'. It works by incrementally changing a candidate support vector set using a greedy approach, until the supporting hyperplane is found within a finite number of iterations. It is derived from a simple active set method which sweeps through the set of Lagrange multipliers and keeps optimality in the unconstrained variables, while discarding large amounts of bound-constrained variables. Given an optimal solution on a subset, only one point is added to the set of SVs at a time and the exact solution is computed. The computations are performed at O(m^2) cost, where m is the number of current SVs, as the solution is not recomputed from scratch. The real benefit of this technique arises when the number of SVs is small relative to the data set. To evaluate their algorithm's improvement over existing SVM training algorithms, they compared the number of kernel evaluations performed by simpleSVM, Sequential Minimal Optimisation (SMO) for the linear soft margin (Platt (1999)), and Nearest Point Algorithm (NPA) for quadratic soft margin (Keerthi et al. (1999)), as opposed to reporting generalisation performance. SimpleSVM outperformed

NPA on all five data sets used, and outperformed SMO when the number of margin SVs was small. Also, unlike NPA and SMO, simpleSVM's runtime behaviour (number of kernel evaluations) does not critically depend on the regularisation constant's value. The algorithm is efficient, intuitive, fast, and numerically stable. It also does not deal with the overall optimality problem that the Incremental SVM (Cauwenberghs and Poggio (2000)) has to. However, for simpleSVM to perform well, the data set has to be relatively clean (i.e. small number of SVs), otherwise SMO would be preferable. Also, storage becomes an issue with simpleSVM when applied to generic dense matrices on large noisy data sets. Kernel caching could be used to reduce the number of kernel evaluations, and finally, the addition of a point to the SV set is entirely reversible, making the calculation of leave-one-out errors possible.

The core vector machine (CVM) by Tsang et al. (2005) is another attempt to speed up the SVM's training process and to increase scalability. It is a very fast SVM type algorithm that uses a subsampling trick, enabling it to handle very large data sets ($>$ 1m points) efficiently. At every iteration, 59 points are randomly sampled making sure there are no repetitions, and a minimum enclosing ball (MEB) is fit around these points. An approximate method is used for this process, since algorithms for finding exact MEBs (such as the one proposed by Welzl (1991)) do not scale well with the dimensionality of the points. Then, the most distant of the points from the MEB's centre is added to the core set. Quadratic programming (QP) is then used on this core set as opposed to the entire training set, making this algorithm much faster. Also, creating and storing a large kernel matrix is no longer necessary.

After formulating the MEB problem, Tsang et al. (2005) obtained a transformed kernel $\widetilde{k}(\mathbf{x}_i, \mathbf{x}_j) = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) + y_i y_j + \frac{\delta_{ij}}{C}$, together with its associated feature space $\widetilde{F}$, mapping $\widetilde{\varphi}$, and constant $\widetilde{\kappa} = \widetilde{k}(\mathbf{x}, \mathbf{x})$. The CVM is shown in Algorithm 2, where we denote the core set as S_t, the ball's centre as c_t, and its radius as R_t at the tth iteration, and the centre and radius of a ball B as C_B and r_B respectively. More details and calculations are available in the original paper (Tsang et al. (2005)). It is interesting to note here that the significant speed-up lies in the calculation of the MEB in step 3 of this algorithm. Since the size of the core set $|S_t|$ is much smaller than N (as mentioned in Tsang et al. (2005) and demonstrated in our experiments in Chapter 4)), the computational complexity of each QP sub-problem is smaller than solving the QP on all the data. A disadvantage of CVM is that SMO can be faster when the data set is relatively small. Another drawback is that, unlike the OCBudget (see Section 2.3), when a core vector is added to the core vector set, it is never removed. This could lead to a significant number or redundant core vectors which could slow down testing. Finally, there are no clear ways of choosing the right values for the three hyperparameters. We use the CVM extensively in our empirical study to compare with our novel one-pass technique in Chapter 4, given its ability to scale to the large KDD '99 data set.

Shortly thereafter, Canu and Loosli (2006) tried to reproduce the CVM paper's results

Algorithm 2 The Core Vector Machine

Input: labeled training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$,$\epsilon > 0$
Initialise: $S_0 = \{\mathbf{x}_a, \mathbf{x}_b\}$, where $\mathbf{x}_{a,b} \in S$ belong to the 2 different classes, $c_0 = \frac{1}{2}(\tilde{\varphi}(\mathbf{x}_a) + \tilde{\varphi}(\mathbf{x}_b))$,$R_0 = \frac{1}{2}\sqrt{2\tilde{k}(\mathbf{x}, \mathbf{x}) - 2\tilde{k}(\mathbf{x}_a, \mathbf{x}_b)}$

1) Terminate if there is no training point $\mathbf{x}$ such that $\widetilde{\varphi}(\mathbf{x})$ falls outside the $(1+\epsilon)$-ball $\mathrm{B}(c_t, (1+\epsilon)R_t)$.
2) Find $\mathbf{x}$ such that $\widetilde{\varphi}(\mathbf{x})$ is furthest away from c_t. Set $S_{t+1} = S_t \cup \{\mathbf{x}\}$.
3) Find the new MEB(S_{t+1}) by solving the optimisation

$$\max_{\alpha} -\alpha' K \alpha : \alpha \geq \mathbf{0},\ \alpha'\mathbf{1} = 1$$

and set

$$\begin{aligned} c_{t+1} &= c_{MEB(S_{t+1})} \\ R_{t+1} &= r_{MEB(S_{t+1})} \\ c &= \sum_{i=1}^{|S_{t+1}|} \alpha_i \varphi(x_i) \\ R &= \sqrt{\alpha' \mathrm{diag}(K) - \alpha' K \alpha}. \end{aligned}$$

4) Increment t by 1 and go to step 1.

because they were convinced that the simpleSVM (Vishwanathan et al. (2003)), used in the CVM paper for comparison purposes, could handle the given problem more efficiently. Their results are, to some extent, contradictory to those reported in Tsang et al. (2005). Through some simulations, they showed that the CVM is not always as accurate as existing SVM implementations, and that sometimes, it does not even converge towards the solution. The algorithm is also very sensitive to changes in its hyperparameters, sometimes causing very unstable behaviour. Furthermore, the authors make an interesting observation that caution should be taken when comparing CVM with regular SVM solvers because of the difference in stopping criteria. Because of these issues, it seems worthwhile to study the effects of the random subsampling performed by the algorithm, and whether it significantly affects performance. We undertake this task in Chapter 4.

Even though the MEB was used as part of the formulation for the CVM, it can be used as a stand-alone novelty detector. It works by fitting a hypersphere onto the data in feature space using data from one class only, which is beneficial because in some problem domains, such as the computer intrusion detection one, data from one class could be expensive and difficult to obtain. Any points that fall within the hypersphere during testing are classified as 'normal', and any points outside as 'anomalous'. The MEB formulation in Tsang et al. (2005) uses the same subsampling trick as the CVM, randomly selecting 59 points at every iteration and using the furthest one (greedy approach) to update the hypersphere. It is a very fast algorithm and given the correct hyperparam-

eters, converges in a short period of time. Anomaly detectors such as the MEB fail in situations like the KDD '99 Cup, where the normal data distribution (used to fit the hypersphere) in the training set is different than the normal data distribution in the testing data set, causing the MEB to misclassify many normal points as intrusions. The MEB is compared against our method in Chapter 4.

Other techniques in the literature aim to speed the learning process by preprocessing the training data. One such algorithm was proposed by Yu et al. (2003), which they call Clustering-Based SVM (CB-SVM). They apply a hierarchical micro-clustering algorithm (BIRCH) that scans the data set once to provide the SVM with high-quality samples that carry the statistical summaries of the data such that SVM learning is improved. The algorithm is shown in Algorithm 3, and results on the KDD '99 Cup are compared with our methods in Chapter 4. The reported empirical results show better performance rates

Algorithm 3 The CB-SVM algorithm

(1) Construct 2 Cluster-Feature (CF) trees (see Yu et al. (2003)), one from the data labeled -1, and the other from the data labeled $+1$.

(2) Train an SVM boundary function from the centroids of the root entries.

(3) De-cluster the entries near the boundary in the next level. The de-clustered child entries are added to the training set with the non-de-clustered parent entries.

(4) Construct another SVM from the centroids of the entries in the training set, and repeat from step 3 until nothing is accumulated.

than randomly sampling a small subset of the data. However, sampling time was rather large. It has a very similar performance level to the ASVM (Mangasarian and Musicant (2000)) (uses adaptive learning), but only requires one pass through the data and is therefore much faster. They compared their algorithm to ASVM because it uses selective sampling, and also compared CB-SVM to random sampling. The biggest drawback of this algorithm is that it is only restricted to linear kernels (since the hierarchical micro-clusters would not be isomorphic to a new high-dimensional feature space). Another disadvantage is that the clustering algorithm does not allow backtracking, and so logical inaccuracies may exist (depending on the order of data input). Furthermore, the time required to create the clusters is rather large and can impair the use of this algorithm in some cases. Finally, there is no obvious way of setting the threshold parameter, which is crucial for building a CF tree of the right size which fits into memory.

Asharaf et al. (2006) describe a method that overcomes the CB-SVM's restriction to linear kernels. It uses a scalable hierarchical clustering algorithm to scale SVMs (with Gaussian kernel functions) in order to handle large data sets. The clustering algorithm creates cluster indexing structures of the training data in the kernel induced feature space, which are then used in the selective sampling of the data to train the SVM. The

clustering algorithm (KBHC) is a kernelised form of the BIRCH algorithm, and it uses a single scan of the data set (which improves scalability) to create a structure similar to the CF tree in a Gaussian induced feature space. The outputs of this algorithm are two MCFs (Modified Clustering Features), one for each class, which is basically the hierarchical tree structure discussed above. Then, SVM training begins using the mean values computed from the MCF entries of the root nodes. When an SVM training round is over, the training set of the subsequent iterations is obtained by selectively de-clustering the MCF entries that are near the decision boundary. This selective sampling process and SVM re-training continues until the leaf nodes of the MCF trees are encountered. Since the authors use a soft margin SVM, the clusters that fall on the wrong side of the decision boundary are taken care of. Experiments were performed using the KDD '99 Cup data. The empirical results show that the KBHC-CBSVM outperforms the aforementioned CB-SVM (Yu et al. (2003)), because the latter can only handle linear kernels, at the expense of slightly more computation time. However, the KBHC-CBSVM's training time is much less than the SVM's and has comparable performance while also using far fewer support vectors, translating into a faster testing phase. Results on the KDD '99 data are compared with our method in Chapter 4.

There has also been work on scaling Bayesian inference methods to large data sets. Notable recent work on Gaussian processes for large data sets is described in Rasmussen and Williams (2006). One popular Bayesian technique is the relevance vector machine (RVM) by Tipping (2001), which has identical functional form to the regular SVM, but allows probabilistic classification. The RVM uses an expectation minimisation (EM) type algorithm to set its parameters, which avoids the cross-validation required by the SVM. On the downside, the EM could get stuck in local minima, which is not the issue for techniques such as the SMO. Of more relevance to this thesis is the memory and computational complexity of the learning algorithm, which, according to Tipping (2001), scale to the square and cube of the number of basis functions respectively. This means that the RVM is impractical for data sets numbering several thousands, let alone millions. We therefore do not use Bayesian-based techniques in our empirical study, and focus on online one-pass algorithms instead.

2.3 Online Algorithms

Online learning algorithms process each training instance once on arrival without the need for storage and reprocessing, and maintain a current hypothesis that reflects all the training instances seen so far. Such algorithms have advantages over typical batch algorithms in situations where data arrive continuously. They are also useful with very large data sets on secondary storage, for which the multiple passes required by most batch algorithms are prohibitively expensive. Given that training is the most computationally intensive task, it is not surprising that availability of online algorithms is a

major prerequisite imposed by practitioners that work on large data sets (see LeCun et al. (1998)). Online learning algorithms are also typically fast, memory efficient, and simple to implement. Also, in many settings, such as the intrusion detection one (where data arrive continuously in large quantities), response time is more important than exactness. In other words, a good and fast answer is better than an exact but late answer. Furthermore, online learning is advantageous when dealing with non-stationary data. According to Murata (1992) and Murata et al. (2002) batch algorithms will generally fail if ambiguous information (such as different distributions varying over time) is present and is erroneously integrated by the batch algorithm. Finally, an important practical advantage of online algorithms is that they are able to incorporate additional training data, when it is available, without re-training from scratch.

Crammer et al. (2003) attempt to tackle the problems online algorithms often entail when used with kernels, which are vast memory and computational requirements. They described and analysed an approach to minimise the number of past examples used for prediction. They tested their algorithm, which they call online classification on a budget (OCBudget), on real data sets which showed that their algorithm with a single epoch was competitive with the SVM, which has to solve an expensive QP problem. Many researchers have attempted to modify and enhance the original Perceptron algorithm based on work done on SVMs. Such algorithms (which can also be used with kernel functions) include:

- Approximate Maximal Margin Classification Algorithm (ALMA) (Gentile (2001))
- Relaxed Online Maximum Margin Algorithm (ROMMA) (Li and Long (2002))
- Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer (2003))

Even though the main drawback of kernel-based methods is the $N \times N$ kernel matrix they require, being able to minimise the linear increase of support vectors with the number of prediction errors is still desirable. This way, processing takes place on an $m \times N$ matrix, where $m \ll N$. Crammer et al. (2003) present an online algorithm that is sparse, but more importantly, they claim, generalises well. The algorithm stores the support patterns in a cache, and revises the weight vector after each prediction mistake, using insertion and deletion phases. When a prediction mistake occurs, the algorithm adds the new erroneous example (insertion phase), then looks for redundant past examples given the recent addition (deletion phase). The algorithm is shown in Algorithm 4 and the DistillCache function in Algorithm 5. We compare this algorithm with our novel one-pass technique in Chapter 4.

Weston et al. (2005) present the 'even tighter budget' algorithm (ETBudget), which is a natural extension to the OCBudget above. The idea is to simply replace the margin

Algorithm 4 Crammer et al. (2003)'s OCBudget algorithm

Input: Training data $\{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}$, Tolerance $\beta > 0$
Initialize: Set $\forall t,\ \alpha_t = 0$, $\mathbf{w}_0 = 0$, $C_0 = \phi$

for $t = 1, 2, \ldots, N$ **do**
  Get a new instance $\mathbf{x}_t \in \mathbb{R}^d$
  Predict $\hat{y}_t = \text{sign}(y_t(\mathbf{x}_t.\mathbf{w}_{t-1}))$
  Get label y_t

  if $y_t(\mathbf{x}_t.\mathbf{w}_{t-1}) \leq \beta$ **then**
    Insert $C_t \leftarrow C_{t-1} \cup \{t\}$
    Set $\alpha_t = 1$
    Compute $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} + y_t\alpha_t\mathbf{x}_t$
    DistillCache$(C_t, \mathbf{w}_t, (\alpha_1, \ldots, \alpha_t))$

  end if

end for

Output: $H(\mathbf{x}) = \text{sign}(\mathbf{w}_T \cdot \mathbf{x})$

Algorithm 5 Function DistillCache

Input: $C, \mathbf{w}, (\alpha_1, \ldots, \alpha_t)$
Loop:

Choose $i \in C$ such that $\beta \leq y_i(\mathbf{w} - \alpha_i y_i \mathbf{x}_i) \cdot \mathbf{x}_i$

if no such i exists **then**
  return

end if
Remove the example i from C:
1) $\mathbf{w} \leftarrow \mathbf{w} - y_i\alpha_i\mathbf{x}_i$
2) $\alpha_i = 0$
3) $C \leftarrow C/\{i\}$ **End**

Return $C, \mathbf{w}, (\alpha_1, \ldots, \alpha_t)$

measure:

$$i = \arg\max_{j \in C_t} \{y_j(\mathbf{w}_{t-1} - \alpha_j y_j \mathbf{x}_j) \cdot \mathbf{x}_j\}$$

with the Leave One Out error on all currently seen examples:

$$i = \arg\min_{j \in C_t} \left(\sum_{k=1}^{t} L\left(y_k, \text{sign}((\mathbf{w}_{t-1} - \alpha_j y_j \mathbf{x}_j) \cdot \mathbf{x}_k)\right) \right).$$

where L is the measure of loss on example k. Unlike the OCBudget, which simulates the removal of each point from the cache and chooses the example that remains recognised

with the largest margin, the ETBudget uses the overall error rate on all currently seen examples. Therefore, if an example is well classified (i.e. has a large margin), then it will also be correctly classified when removed as in the equation above, and moreover, all other examples will still be well classified as well. When compared to the OCBudget, the removal rule is significantly more expensive to compute. To this effect, the authors discuss ways of approximating this computation while retaining its desirable properties. They suggest reducing the measure of loss only to the examples in the cache. Another suggestion was to randomly choose a fixed number of examples from the entire data set. However, their preferred method uses a secondary caching scheme to choose the q examples with which to estimate the error. Intuitively, the most desirable examples to keep are those that are most likely to change label, as those are most likely to give information about the performance of the classifier. This is because a well classified example will not change label easily when the classifier changes slightly. Similarly, if an example is an outlier, it will be consistently misclassified. Therefore, the authors suggest keeping a count s_i of the number of times example $\mathbf{x}_i$ has changed label, divided by the amount of time it has been in the cache. If this value is small, one can consider removing the corresponding point from the secondary cache. We compare this algorithm empirically with the OCBudget and our novel technique in Chapter 4.

Orabona et al. (2008) present a discriminative online algorithm called the Projectron that is based on the kernel-based Perceptron. The advantage of their algorithm is that it bounds memory growth, without discarding support vectors like the OCBudget does. However, the actual size of the support set cannot be determined in advance. In the proposed method, the support vectors will, instead of being discarded, be projected onto the space spanned by the previous online hypotheses. The user also has the ability to tune a parameter to trade the accuracy for sparseness. In their empirical study, the authors achieved similar generalisation performance to the standard Perceptron, with fewer support vectors. Also, a study by Ma et al. (2009) showed no improvement of the Projectron over linear classifiers. While the proposed algorithm bounds memory, it comes at a higher computational cost than the Perceptron, and is therefore slower. Since our aim is to find fast online methods, this technique will not be used in our empirical study.

Ma et al. (2009)'s contribution is to use online algorithms to detect malicious web sites through their URLs. They argue that for such a task, which is similar to our intrusion detection one, online methods are the most suited because the training data is larger than what batch algorithms can process efficiently, and because the distribution of features that characterise malicious URLs is constantly changing with time. Despite these reasons, practitioners usually apply batch algorithms, as they are more well-established and are a safer option. However, the online methods used in Ma et al. (2009)'s empirical study proved highly accurate on the task at hand, and continuously re-training these algorithms is crucial to cope with the ever-evolving features of URLs.

Boosting is a very powerful technique for building classifiers, as mentioned in Section 2.2. However, they cannot handle large data sets because they have to maintain a distribution over the entire training set. Therefore, Bradley and Schapire (2008) present FilterBoost, an adaptive algorithm that is online and is applicable to both conditional probability and classification. An "Oracle" generates and feeds data (from D, the underlying distribution of the training data) to the algorithm, which in turn accepts each example with probability proportional to the example's weight $D_t(\mathbf{x}, y)$. This mechanism is called the *filter*.

On each round t, FilterBoost draws m_t examples from the filter to train the weak learner and get $h_t : X \to Y$, where X are the examples, and Y are the binary labels $\{-1, +1\}$. Then, they calculate the weight α_t using the edge γ_t of h_t, defined as $\gamma_t = 1/2 - \epsilon_t$. The final hypothesis is $H_t(x) = \text{sign} \sum_{t'=1}^{t} \alpha_{t'} h_{t'}$. Empirically, FilterBoost proves more robust to noise and overfitting than normal batch boosters for conditional probability estimation, and is competitive for classification.

Algorithm 6 The Voted Perceptron algorithm

Input: A labeled training set $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, number of Epochs T
Initialise: $k = 0$,$\mathbf{v}_1 = \mathbf{0}$,$c_1 = 0$
Training:

```
for j = 1, ··· , T do
  for i = 1, ..., N do
    Compute prediction: ŷ = sign(v_k · x_i)
    if ŷ = y then
      c_k = c_k + 1
    else
      v_{k+1} = v_k + y_i x_i
      c_{k+1} = 1
      k = k + 1
    end if
  end for
end for
```

Prediction:
Given : List of weighted Perceptrons: $\langle(\mathbf{v}_1, c_1), ..., (\mathbf{v}_k, c_k)\rangle$, and an unlabeled instance $\mathbf{x}$ compute $\hat{y} = \text{sign}(\sum_{i=1}^{k} c_i \text{sign}(\mathbf{v}_i \cdot \mathbf{x}))$

Freund and Schapire (1998) present the Voted Perceptron, which is a modification of the original Perceptron algorithm by Rosenblatt (1958, 1962). In this algorithm, the authors store more information during training, then use this information to obtain better predictions on the unseen data. The information they store is the list of all prediction vectors that were generated after every mistake. For every such vector ($\mathbf{v}_k$), the count of the number of iterations it survived until the next mistake ($\mathbf{c}_k$) is stored (see Algorithm 6). This is called the 'weight'. So, when a training point is received, a prediction ($\hat{y}$) is made using the current prediction vector $\mathbf{v}_k$. The true class y is then received, and if $\hat{y} = y$, then the weight of the prediction vector is incremented ($c_k = c_k + 1$). If $\hat{y} \neq y$, then the prediction vector is updated ($\mathbf{v}_{k+1} = \mathbf{v}_k + y_i \mathbf{x}_i$),

the weight is set to one ($c_{k+1} = 1$), and k is incremented by 1. Upon receiving a test point, the binary prediction of each of the prediction vectors is calculated and a weighted majority vote is used to determine the outcome. We implement the Voted Perceptron and compare it with our novel method on the KDD '99 data in Chapter 4.

To extract even more information during training, we propose a modification to the Voted Perceptron so that even after a prediction vector makes a mistake and is placed aside, its weight will still be modified. This means that the current example is presented to all k prediction vectors, not just the kth one. The weight of any prediction vector that classifies the current point correctly is incremented by one. This captures more information from the training set and could return weights that are more accurate. The reasoning behind this modification stems from the idea that a hard-to-classify point could be presented to a prediction vector early on, prematurely terminating it's weight calculation. We show in Section 4.7 that this modification obtains an improvement in generalisation accuracy over the original Voted Perceptron. On the KDD '99 test set, the Voted Perceptron achieved a prediction accuracy of 91.89%, while the memory modification improved this result to 92.72%. This does not come at a very high cost to the algorithm, as can be seen from the difference in the time required by the VP and the modified VP in Table 4.3

2.4 Summary

Hundreds of different machine learning techniques exist in the literature. In this chapter, we presented a review of recent techniques from the literature that are both relevant to large scale learning and our proposed techniques in Chapter 3 and Chapter 4. We presented clustering techniques that will be used to create an initial tree for our novel one-pass clustering technique of Chapter 3. We then present batch and online classification algorithms that can scale to large data. These will be used in our empirical study to compare against our novel one-pass Voted Spheres technique (in Chapter 4 and Chapter 6).

Chapter 3

Sequential Hierarchical Pattern Clustering

This chapter deals with clustering and its applications in machine learning. Clustering can be considered the most important unsupervised learning technique, and is a fundamental task in the automatic processing of large data sets. Since labels are not provided, clustering methods have to find structures in the data; the idea being that points that belong in the same cluster are more 'similar' to each other than points belonging to other clusters. However, there is one main problem affecting traditional clustering methods, such as k-means clustering: they require multiple passes through the data, which is further complicated by the huge increase in the amount of data available in the domains in which they are being applied. To tackle this issue, we present a novel, hierarchical, one-pass clustering technique capable of handling very large data sets. The idea is that given a good initial hierarchical tree, we sequentially process the remainder of the training points based on which branch of the tree is 'closest' to our input point. We also maintain a novelty threshold, which causes a new branch in the tree to be created whenever the distance of our current point to our current branch is larger than it. Our experiments on the Capitals data, bioinformatics data sets, and subsets of the KDD '99 Cup data show that the quality of clusters does not degrade, while affording us great computational savings.

3.1 Introduction

Clustering, discussed in Section 2.1, is a powerful unsupervised method used to find a structure in a collection of unlabeled data. Given that many recent diverse research areas generate unlabeled data that needs to be mined, clustering has become quite popular over recent years. For instance, Achtert et al. (2005) use clustering in mining large data warehouse environments, Hasan and Jue (2008) for dynamic routing in optical networks,

Zhao et al. (2005) for text classification, and Zhang et al. (2007) for codebook construction for bag-of-keypoint visual scene analysis problems. Clustering is especially popular in bioinformatics, for tasks such as gene expression analysis (Eisen et al. (1998)), where expression profiles of genes measured across different biological conditions are clustered. Genes that fall into the same cluster may be assumed to have common functional properties, such as acting under the control of the same regulatory mechanism, or acting in tandem along a signaling pathway. Another example of clustering in bioinformatics is the analysis of protein sequences to assign putative function by Frey and Dueck (2007). However, repositories of protein sequences have seen massive growth in recent years (Wu et al. (2006)). As the number of sequenced proteins grows at a much faster rate than those whose structure is determined, or function characterised, automatically predicting function by clustering the sequence space is an active topic of interest. While clustering algorithms and their performance characteristics have been studied extensively over recent years, a particular property of several of the new problems, including the bioinformatics problems, is their massive scale. In other areas too, data mining examples with a million or more data points are becoming available such as the KDD Cup Challenges[1]. Another example is the UniProt database of proteins (Wu et al. (2006)), which now consists of over six million sequences. A matrix of pairwise similarity scores of all these proteins has a file size of 2.6GB. With data of this magnitude, classical clustering algorithms such as k-means or hierarchical clustering are not straightforward to apply and a need for novel, faster approaches arises.

To deal with the issues of learning with such large scale data, we will need to use constructive online algorithms, such as the resource allocating network by Platt (1991) (see Section 4.3.2) and its variants by Kadirkamanathan and Niranjan (1993), and Molina and Niranjan (1996). In this chapter we put forward a formulation for hierarchical clustering by sequentially processing the data in a one-pass setting, after constructing the initial tree from a small random subsample of the data. Any hierarchical clustering technique (batch or online) can be used for this initial stage. Then, the remainder of the data are sequentially inserted into the appropriate branch of the tree, adapting the tree's structure as we proceed. In this setting, we need not calculate the $N \times N$ pairwise similarity matrix other techniques need in order to work.

3.2 Sequential Hierarchical Pattern Clustering Algorithm

The algorithm we present, which we call Sequential Hierarchical Pattern Clustering (SHPC), has two phases:

1. The construction of an initial tree from a small random subsample of data;

[1] http://www.sigkdd.org/kddcup/index.php

2. Sequentially inserting the remaining data points into the tree, and updating the tree structure.

For the initial tree, any hierarchical clustering method can be used. In our empirical study, we use the Single-Round-MC-UPGMA algorithm by Loewenstein et al. (2008), described in Section 2.1.

Following the construction of the initial tree, the remaining data is sequentially processed using Algorithm 7. Upon receiving a new input pattern $\mathbf{x}_i$, we calculate its (Euclidean) distance d to the root ($d(\mathbf{x}_i, \mathbf{root})$) of the current hierarchical tree. If $d(\mathbf{x}_i, \mathbf{root})$ is greater than a predefined threshold (θ), we deem the point too different to that branch of the tree. We create a new root, with the previous root and $\mathbf{x}_i$ as its children. This increases the depth of the tree by one. The value of the new root is assigned with the arithmetic mean of all the leaf nodes. However, if $d(\mathbf{x}_i, \mathbf{root})$ is less than θ, then the point belongs to this branch of the tree, and the nearest child of the current node is retrieved. If the distance of $\mathbf{x}_i$ to this child node is also smaller than θ then we continue to repeat finding the closest child until either the distance to the current node is greater than θ or we reach a leaf node. In either of the two cases, $\mathbf{x}_i$ is created as a sibling to the node under consideration, and $\mathbf{x}_i$'s value is propagated up the tree to update its ancestors. Each intermediate node holds a value representing it's leaf nodes; If the features of $\mathbf{x}_i$ are numeric, then the intermediate nodes hold the arithmetic means, otherwise the nodes hold the most representative leaf node. Because of this, we need not traverse the entire tree during the update process. As for the most representative leaf, there is no specific way to select this, however we discuss several possible ways in Section 3.4.

The only hyperparameter in our algorithm is θ. This can be tuned in several ways, and the specific way we do this in our empirical study is described in Section 3.3. The value of θ determines how similar patterns should be to one another in order to belong to the same branch of the tree. Setting different values for this, we can obtain different numbers of clusters at different levels of granularity. As for the input patterns, they are always represented as leaf nodes in our tree, and as discussed earlier, the intermediate nodes (up to and including the root) contain either the arithmetic means of the leaves they represent, or the most informative leaf node they represent.

To illustrate and evaluate our algorithm, we use the Capitals data set[2]. This is because the structure of the clusters is known, enabling us to determine whether we are correctly adapting our tree. We numbered the capital cities as in Table 3.1. In Figure 3.1, we show the ground-truth tree, which is constructed using the Single-Round-MC-UPGMA on the entire capitals data set, which is identical to the tree depicted on the data's website. Figure 3.2 shows how our SHPC method inserts the last two points into the tree, given that the first 15 points were used for the construction of the initial tree. If we

[2]http://www.quretec.com/HappieClust/

Algorithm 7 Our SHPC method to update a hierarchical tree.

Input: Root of the initial tree (CurNode), the new pattern (NewNode), and the novelty threshold (θ)

Output: Updated hierarchical tree

```
simdist_CN ← similarity_distance(CurNode, NewNode)
if (simdist_CN ≤ θ) then
  (⋆) Children ← getChildrenOf(CurNode)
  if (Children == NULL) then
    Make NewNode as a sibling of CurNode and update ancestors
  else
    {CurNode has children}
    nearestNode ← min (similarity_distance(Children, NewNode))
    if (nearestNode ≤ thresh) then
      CurNode ← nearestNode
      Go to (⋆)
    else
      Make NewNode as sibling of CurNode and update ancestors
    end if
  end if
else
  Make NewNode as sibling of CurNode by creating a new root
end if
```

Capital	Mapped to
Tallinn	1
Beijing	2
Berlin	3
Buenos Aires	4
Cairo	5
Canberra	6
Cape Town	7
Helsinki	8
London	9
Moscow	10
Ottawa	11
Paris	12
Riga	13
Rome	14
Singapore	15
Stockholm	16
Washington	17

TABLE 3.1: Mapping of the Capitals data set

cut our tree at the second level, we get the exact same four clusters obtained by cutting the tree in Figure 3.1 at the same level.

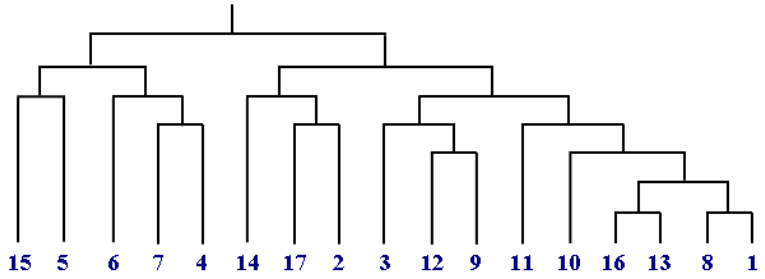


FIGURE 3.1: Hierarchical tree constructed using Single-Round-MC-UPGMA on the entire capitals data set.

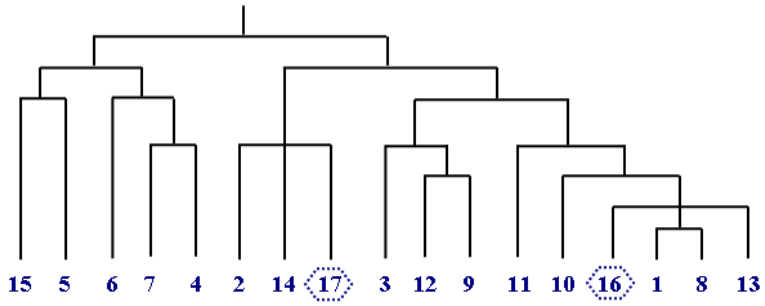


FIGURE 3.2: Hierarchical tree constructed by our approach in an online fashion with the aid of an initial tree constructed by Single-Round-MC-UPGMA. The initial tree was constructed with the first 15 capitals in the data set. Capitals Stockholm and Washington (depicted in dotted hexagons) were sequentially inserted using Algorithm 7.

To illustrate the importance of having a good initial tree, we use El-Sonbaty and Ismail (1998)'s method on the entire capitals data set (shown in Figure 3.3). Figure 3.4 shows how our method (Algorithm 7) inserts the last two points, given that the first 15 points were used for the initial tree using El-Sonbaty and Ismail (1998)'s method. Even though Algorithm 7 inserted the last two points correctly with respect to Figure 3.3, the same four clusters obtained by the trees in Figure 3.1 and Figure 3.2 are not attainable due to the incorrect initial tree, which is beyond the control of Algorithm 7.

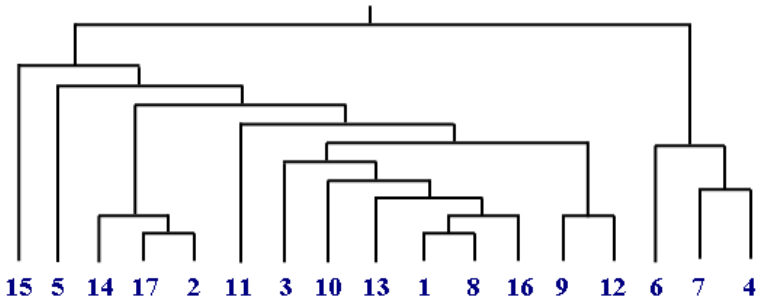


FIGURE 3.3: Hierarchical tree constructed by the approach proposed in El-Sonbaty and Ismail (1998) on the entire capitals data set.

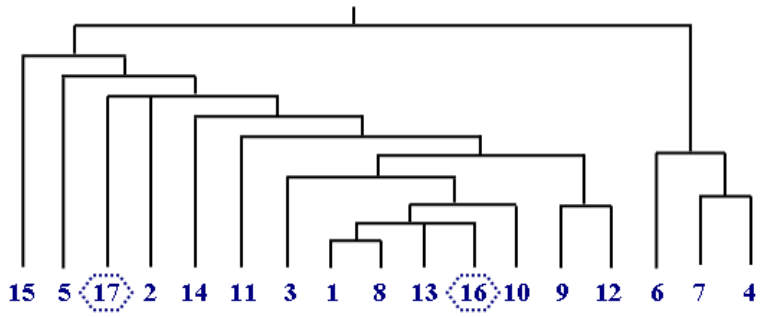


FIGURE 3.4: Hierarchical tree constructed by our approach with the aid of an initial tree constructed using the method by El-Sonbaty and Ismail (1998) on the capitals data set.

	Actual Positive	Actual Negative
Predicted Positive	TP	FP
Predicted Negative	FN	TN

FIGURE 3.5: Confusion table used to calculate F1 measure.

3.3 Experiments and results

In this section, we use bioinformatics data sets, two UCI data sets, and subsets of the KDD '99 data to evaluate our SHPC technique. We also use the F1 measure, which is widely used in the information retrieval literature to quantify the quality of clusters. Given that the data sets we use in our empirical study are classification problems, we have the ground truth results, and are able to calculate the F1 measure easily. Using Figure 3.5, we define:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{F1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

3.3.1 On Bioinformatics Data sets

To evaluate our algorithm we used two bioinformatic data sets (see Table 3.2). The first is Eisen et al. (1998)'s gene expression clusters consisting of ten clusters formed by their clustering algorithm. We make the weak assumption that the clusters published by these authors are perfect associations and quantify how close our approach gets to this solution. The second data set is from a protein fold classification problem, constructed by Ding and Dubchak (2001) on a subset of the Structural Classification of Proteins (SCOP) (Conte et al. (2000)) database. This is essentially a classification problem,

but we apply clustering to it (without using the class labels) and evaluate how well the resulting cluster membership matches the clusters returned by Single-Round-MC-UPGMA applied on the entire subset. We combined both training and testing sets provided in Ding and Dubchak (2001). The results are given in Table 3.2, and are the average of the 10 runs where we randomised the initial subset and the order of presentation of the remaining data. The number of points to use for the initial tree were selected at random.

The threshold θ was determined from the data sample used to construct the initial tree. θ was set as the sum of the pairwise Euclidean distances between the patterns in this data sample. This was set to disallow the SHPC from increasing the depth, since if the initial tree obtained a good partition of the input space, we need not create new roots or branches. All the input points would be added as leaf nodes.

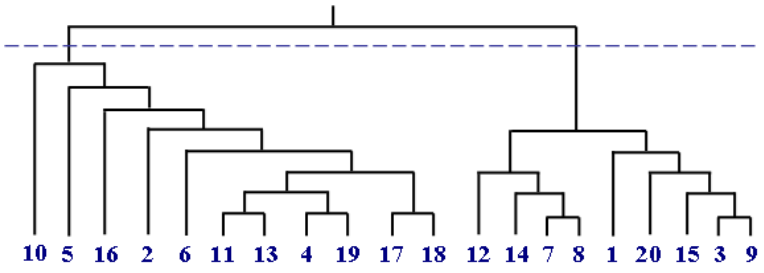


FIGURE 3.6: Hierarchical tree constructed by the Single-Round-MC-UPGMA scheme on Eisen's data clusters labeled as B and D Eisen et al. (1998). The tree was constructed by using the whole data of the selected two clusters. The dendrogram was cut at the root node (shown in dotted lines) to obtain two clusters.

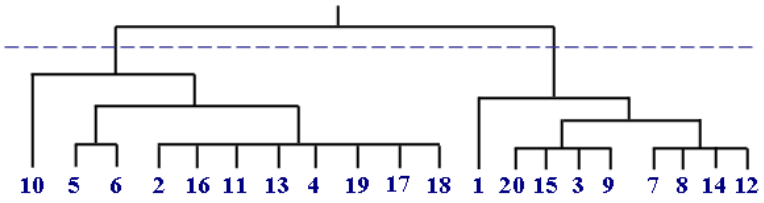


FIGURE 3.7: Tree constructed by the proposed approach with the aid of an initial tree constructed by Single-Round-MC-UPGMA on Eisen et al. (1998)'s clusters labeled as B and D. The initial tree was constructed with 20% of the data and the rest was clustered using Algorithm 7. The dendrogram was cut at the root node (shown in dotted lines) to obtain two clusters.

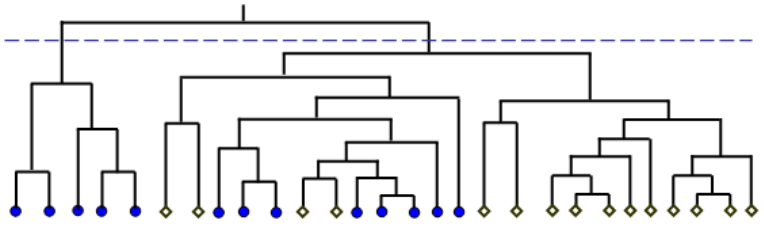


FIGURE 3.8: Tree constructed by Single-Round-MC-UPGMA on the two selected folds Alpha: four-helical up-and-down bundle (depicted as filled circles) and Beta: ConA-like lectins (depicted as diamonds) of the SCOP data subset Ding and Dubchak (2001). We used the whole 28 data points for the construction of this tree.

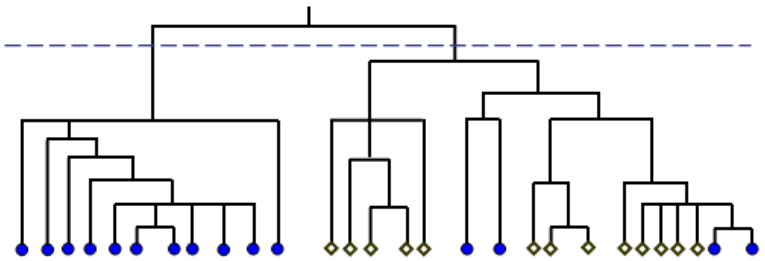


FIGURE 3.9: Tree constructed by the proposed approach with the aid of an initial tree constructed by Single-Round-MC-UPGMA on the two selected folds Alpha: four-helical up-and-down bundle (depicted as filled circles) and Beta: ConA-like lectins (depicted as diamonds) of the SCOP data subset (Ding and Dubchak (2001)). We used 20 out of 28 data for the construction of initial tree and used the rest in an online manner.

Figure 3.6 shows the tree constructed by the Single-Round-MC-UPGMA scheme on the Eisen's clusters labeled as B and D in Eisen et al. (1998). Figure 3.7 shows the tree obtained by using four points for the initial tree, and the remaining 16 points inserted using our approach. Cutting at depth 1 to obtain two clusters shows that we get the exact same two clusters as in Figure 3.6. However when we used our approach on the protein fold data of Ding and Dubchak (2001) (see Figure 3.9), we obtained a better separation than when the entire data was used with Single-Round-MC-UPGMA (see Figure 3.8). Finally, Figure 3.10 shows the trees obtained by using Eisen's data clusters (C,B,I and C,B,D, & I), and the cuts that return the desired clusters.

Data set	No. of data	No. of data used or the initial tree	F1-measure
SCOP (1) & (2)	28	20	1.0 ±0.0
SCOP (3) & (4)	151	100	0.9921 ± 0.0086
Eisen (B & D)	20	4	1.0 ±0.0
Eisen (C & I)	104	26	1.0 ±0.0
Eisen (C, B & I)	113	25	1.0 ±0.0
Eisen (C, B, D & I)	124	30	0.9247 ± 0.0652

TABLE 3.2: Results of the hierarchical clustering performed on a subset of SCOP and Eisen's data. (1) Beta: ConA-like lectins, (2) Alpha: Four-helical up-and-down bundle, (3) Beta: Immunoglobulin-like beta-sandwich, and (4) A/B: beta/alpha (TIM)-barrel.

In this section, we showed the ability of the SHPC to obtain good hierarchical trees using only a small subset of the data for the initial tree. In some cases, such as on Eisen's data, we used as little as 20% of the data for the initial tree. Since the labels of the clusters are available to us, we are able to calculate the F1 measures of our obtained clusters. Table 3.2 shows that on the six subsets used in our empirical study, we achieve very good results, while saving on memory and computational savings.

(a) (b)

FIGURE 3.10: Tree constructed by our approach on the selected clusters of Eisen's data (a) Clusters B, C, D and I; 30 out of 124 data were used for the construction of the initial tree. The tree was cut at the level indicated by the dotted line to yield four perfect clusters. (b) Clusters B, C, and I; 25 out of 113 data were used for the construction of the initial tree. The tree was cut at the level indicated by dotted line to yield three perfect clusters.

3.3.2 Comparison with AP and VSH on UCI Data Sets

In this section, we compare our novel SHPC to Frey and Dueck (2007)'s AP clustering method, which the authors claim has lower computational costs and produces higher quality clusters than other methods. We take into account the time required for the pairwise similarity calculations, which is omitted from Frey and Dueck (2007)'s paper. The AP simultaneously considers all data points as potential exemplars (which are centres that are selected from the actual data points). They consider their data points as nodes in a network, which pass messages along the edges of the network until a 'good' set of exemplars (and their corresponding clusters) are retrieved. The messages are updated using formulas that search for minima of an appropriately chosen energy function. At any given point in time, the magnitude of the message portrays the current affinity that a data point has for choosing another point as its exemplar, hence the name "affinity propagation".

AP takes the $N \times N$ similarity matrix as input, where each similarity value $s(i, k)$ describes how well data point k is suited to be the exemplar of point i. This is a major drawback as it is very computationally expensive, and can prohibit the use of AP for large N. This is not mentioned in the paper however, and Frey and Dueck (2007) actually claim that AP is fast and makes computational savings. Of course, if the time required for the similarity matrix calculations is removed from the time comparison, AP will fare well against other methods. However, an advantage of AP is that the number of clusters need not be specified in advance, such as in the case of k-means or k-centres. Instead, the method requires that the self-similarities ($s(k, k)$) be defined for all points, where a larger value indicates that point k has a higher probability of becoming an exemplar. These values are referred to as "preferences", and the value assigned to them affects the number of exemplars returned, as shown by our experiments below. For instance, setting the preferences to the median of the input similarities results in a moderate number of

clusters, while a smaller value returns fewer clusters and a larger value returns more clusters.

As for the messages passed between points, there are two different types. The first is the "responsibility" $r(i,k)$, which is sent from data point i to a candidate exemplar k, indicating the evidence for how well suited point k is to represent point i, which also takes into account the other potential exemplars for point i. The second type of message is the "availability" $a(i,k)$, which is sent from candidate exemplar point k to point i, reflecting the evidence for how appropriate it would be for point i to choose point k as its exemplar, taking into account the evidence from other points that point k should be their exemplar. Messages are passed between points and updated using the formulas given in Frey and Dueck (2007) for either a fixed number of iterations, until changes in the messages are less than a threshold, or until local decisions do not change much for a certain number of iterations.

Shortly after Frey and Dueck (2007) published their paper, Brusco and Köhn (2008) wrote a response to the AP claiming that the well-established heuristic for the p-median problem frequently returns clustering solutions with lower error rates than AP in comparable computational time. In short, the p-median model (PMM) takes as an input an $m \times n$ similarity matrix $\mathbf{S}$, where each $\mathbf{S}_{i,j}$ represents the similarity of point i to point j, and selects p columns from $\mathbf{S}$ such that the sum of the maximum values from each row of the selected p columns is maximised (Mladenovic et al. (2007)). In other words, this algorithm is assigning each row to its most similar exemplar (represented by the columns), with the goal of maximising the overall similarity. In our application, $\mathbf{S}$ is an $N \times N$ matrix of negative squared Euclidean distances, and so clustering them using PMM is the same as selecting the p exemplars that minimise the error, which is the sum of squared Euclidean distances of each object to its nearest exemplar. Brusco and Köhn (2008) point out that Lagrangian relaxation methods can obtain the exact solution of the PMM when $N \leq 500$, as shown by Cornuejols et al. (1977). For problems with larger N, Brusco and Köhn (2008) use a Vertex Substitution Heuristic developed by Teitz and Bart (1968), which has been the standard for comparison for nearly four decades. The VSH randomly selects p initial exemplars, which it refines iteratively by evaluating the effect of substituting a non-exemplar point for one of the exemplars. For completeness of our results, we will include the VSH in our empirical evaluation (shown in Table 3.3).

We used two data sets in our empirical comparison. The first is the Sequential Control Chart time series obtained from the UCI machine learning repository. The data has 600 examples, with 60 features. These fall into six classes, and so we aim to achieve around six clusters from our clustering methods. The second data set is the Pageblocks classification from the UCI machine learning repository, containing 5,473 examples with 10 features. There are a total of five different classes, and so we aim to get five clusters from each of the clustering algorithms. For the AP and the SHPC, we are unable to select the exact number of clusters, and so we reported results based on the closest to

Data set	Method	# Required clusters	# Clusters returned	Time (secs)	Error
1	VSH	6	7	7.61	1.72×10^5
	AP		7	9.37	1.74×10^5
	SHPC		7	**1.75**	$\mathbf{1.39} \times 10^5$
2	VSH	5	5	1528.91	2.78×10^9
	AP		2	12280.97	3.15×10^{10}
	SHPC		4	**198.797**	$\mathbf{1.66} \times 10^6$

TABLE 3.3: Comparison between VSH, AP, and SHPC on two UCI machine learning data sets: 1) Sequential Control Chart Time Series (600 points with 60 features), and 2) Pageblocks classification (5,473 points with 10 features).

five clusters we were able to get. For the VSH, we can select the exact number of clusters we want, since it is an input parameter to the algorithm, just like k-means. For the first data set, the AP and SHPC (using 400 points for the initial tree) obtained seven clusters, so we set $p = 7$ for VSH to make a fair comparison. For the second data set, the AP and SHPC (using 2,773 points for the initial tree) each return a different number of clusters, and so we set $p = 5$ for the VSH. The error rate used was the average intra-cluster distance of points to their respective centres. More specifically, the reported error was the sum of the Euclidean distances of each point to its cluster centre, divided by the number of returned clusters. From the results shown in Table 3.3, we see that the VSH does indeed achieve similar levels of generalisation to the AP, at lower computational cost. This is consistent to what was reported by Brusco and Köhn (2008). However, the SHPC achieves the lowest error rate among the three tested algorithms, in a small fraction of the time required by the other two methods. These experiments further prove that our one-pass technique does not degrade the performance of the returned clusters, and makes significant computational savings.

3.3.3 Results on the KDD '99 Cup Data

In this section, we use subsets of the KDD '99 data set to test the performance of the SHPC on large classification tasks. We randomly subsampled five subsets of size $30,000$, which are much larger than the data sets we used in the previous sections. We use 15,000 for the initial tree using MC-UPGMA, and inserted the last 15,000 points using Algorithm 7. The novelty threshold θ was set to the sum of the pairwise distances of the data points used to create the initial tree. These are already calculated during the initial tree construction phase, and so no extra effort was required on our part. Since we randomly subsampled our data from the 10% KDD '99 data set, they were roughly 80% intrusions and 20% normal connections. Our method required under a minute to insert the 15,000 points into the tree, and the tree was cut at depth 3 to obtain eight clusters. Since the KDD '99 Cup is actually a classification task, we have the labels of the training data and are able to calculate the prediction accuracy of SHPC. Due to the

high imbalance of the data, all of the clusters were overwhelmed by intrusions, and the end results were therefore random-like. To overcome this issue, we ran the SHPC on five balanced (50% intrusion and 50% normal) randomly subsampled subsets containing 30,000 points from the 10% KDD '99 data. Five runs gave a prediction accuracy of $66.97\% \pm 3\%$. Despite this being lower than the state-of-the-art, it is still much better than random (which is 50% in this case). It is important to note that the learning is unsupervised; the available labels provided with the data set were not used, putting our method at a disadvantage to other supervised classification methods. Also, the KDD '99 data does not have any underlying hierarchical structure, further putting the SHPC at a disadvantage. Nevertheless, when the SHPC is used on unlabeled data for clustering purposes, it performs competitively with other clustering techniques as was shown previously in this chapter.

3.4 Dealing with categorical or symbolic patterns

In the previous sections, the illustration of the SHPC was restricted to the Euclidean space. Here, we discuss the capability of our algorithm to handle categorical or symbolic patterns. Instead of updating the parent nodes using the arithmetic means of their leaf nodes, the most informative child node can be selected to act as a parent. This can be decided by the user, and an example would be to select the child node that had the least average intra-class distance. This would mean that the child that was the most central, i.e. the most neutral, would be selected. For simple comparison purposes, and to show the robustness of our technique, we choose a random child to act as the parent node in Algorithm 7. We compare this with the previously shown cluster trees in Section 3.2 using the Capitals data and the selected subsets of Eisen's data, using the Euclidean distances with the selective node approach. This means that we used the Euclidean distance measure to calculate the distances between points, but instead of using the arithmetic average for the parent, we select one of the child nodes randomly. Our results gave us exactly the same clusters as in the arithmetic average case, and hence the same F1-measure.

Also, the measure of similarity used depends on the application of interest, and is not limited to numerical data. For example, if we are clustering protein sequences, then Smith-Waterman local alignment (Waterman and Smith (1981)) or Needleman-Wunsch global alignment (Needleman and Wunsch (1970)) measures can be used instead of Euclidean distances.

3.5 Conclusion

In this chapter we present an algorithm for online hierarchical clustering. The approach depends on the construction of an initial tree using a random subset of the data. This establishes a scale structure of the input space, and subsequent data can be processed sequentially and the tree adapted constructively. For the initial tree, any hierarchical clustering algorithm in the literature can be used. If the initial tree was constructed incorrectly, then regardless of the fact that our method would insert the remaining points correctly, the final outcome of the tree would be wrong since the required clusters would be unobtainable. This was demonstrated by our experiment using El-Sonbaty and Ismail (1998)'s algorithm, which generated an incorrect initial tree. We demonstrate the effectiveness of our SHPC on bioinformatics tasks and subsets of the KDD '99 data, showing that the quality of the clusters obtained using our method did not degrade, while making significant computational savings.

As an analogy, our overall technique can be thought of as inserting newly purchased books into a library. The initial tree can be considered like the already well-organised library, and upon acquiring a new book, we aim to insert it into its most relevant place on the shelf. If the book does not fit anywhere, we create a new shelf for it close to its closest subject.

The proposed technique could be significantly improved with a more fine-tuned choice of the novelty threshold (θ). θ can be better estimated by taking into account the inter-cluster and/or intra-cluster information of the initial tree. This can be subsequently updated after the insertion of a newly arrived pattern. Another way of better estimating θ might be to use local thresholds associated with each parent or level of the tree, instead of a global threshold. We leave this for future work.

Chapter 4

Voted Spheres

In this chapter, we introduce a novel, non-linear, fast, online algorithm for learning on large data sets. This algorithm which we call Voted Spheres (VS) is a combination of hypersphere-fitting, and the idea of voting. The algorithm builds hyperspheres around points, with different hyperspheres belonging to different classes allowed to overlap. The advantages of the algorithm, which falls under the local learning family of algorithms such as that of Platt (1991) and the modified Kanerva model of Prager and Fallside (1989), are that it is simple to implement, very efficient, and generalises well while being able to handle millions of data points. For the KDD '99 intrusion detection data set consisting of $4,898,424$ data points, the VS algorithm requires just over five minutes to train and test on a standard desktop PC and achieves state-of-the-art performance. This is by far the fastest algorithm among all the ones tested for our empirical study in this chapter. Our method also performs competitively with published results on benchmark data sets, and achieves generalisation rates comparable to the published state-of-the-art on the two-class and multi-class KDD '99 data in a fraction of the time required by other methods. We also present modifications that can be incorporated into the VS, most of which improve the base performance of the algorithm. We conclude the chapter by showing that VS is a compression scheme, and therefore we are able to bound its generalisation error using risk bounds.

4.1 Introduction

In machine learning, the issue of large scale learning is becoming an increasingly important topic of interest. With more and more problem domains generating large amounts of data to be classified, the need for scalable machine learning techniques is very clear to practitioners today. To encourage research into this, the community has been organising competitions aimed at very large scale learning. Some recent examples of this include

the PASCAL Large Scale Challenge[1], the KDD Cup 2009[2], and the PASCAL Visual Object Classes Challenge 2010[3].

Typical batch algorithms that require multiple passes through the data will fail in such applications, as the computational cost will render them impractical. To overcome this problem, people either incorporate a trick into the batch method, or use online algorithms. Tricks that are common include subsampling the data, such as by the CVM, or doing some preprocessing to reduce the data, such as the edited k-NN and CB-SVM methods. While these do reduce computational cost, the preprocessing could be time-consuming, and the learning algorithms still require multiple passes through the data. Online algorithms, on the other hand, process points one at a time as they arrive and maintain a current hypothesis that reflects all the training instances seen so far. The real benefit of such algorithms lies in situations where data arrive continuously, such as in intrusion detection, and on very large data on secondary storage where the computational cost required by the multiple passes of batch algorithms is prohibitively high.

Therefore in this chapter, we show how a powerful one-pass algorithm may be designed for very large scale machine learning problems. We discuss the similarities and differences of our method to two related techniques, namely the edited k-NN and the Resource-Allocating Network in Section 4.3. We then show the flexibility of our algorithm by incorporating modifications, each that solves a different problem. Using 18 popular benchmark data sets, we show the effectiveness of the VS on small problems, which performs competitively against published results on the same data. We then compare the VS and its modifications on the large KDD '99 data, which clearly highlights the computational savings the VS makes, while still outperforming recent published results on the two-class and multi-class 10% KDD subsets, as well as the entire $\sim 5 \times 10^6$ KDD '99 data. We conclude the chapter by showing that the VS can be viewed as a compression scheme, enabling us to theoretically bound the generalisation error using only the training set.

4.2 Voted Spheres Algorithm

In this section we will present and discuss a new non-linear, one-pass, multi-class classification algorithm that is fast, simple, and generalises very well. The algorithm, which we call Voted Spheres (VS), combines two different ideas well. The first is the idea of voting to compress the data, and the second is hypersphere-fitting. The algorithm builds hyperspheres of fixed radius around points, with different hyperspheres belonging to different classes allowed to overlap. Each distinct class can have its own radius, making

[1] http://largescale.first.fraunhofer.de/about/
[2] http://www.kddcup-orange.com/
[3] http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2010/index.html

the algorithm more powerful, as different classes can be spread differently in the input space and have different distances among each other. The resulting decision boundary would have a cloud-like effect around clusters of points in space. This is shown in Figure 4.11, which shows the hyperspheres created by VS on a subset of the two-dimensional vowel data set by Peterson and Barney (1952). In the training phase, if the training example $\mathbf{x}_i$ does not fall into any hypersphere, a new one will be created centred around $\mathbf{x}_i$. However if the input point does fall into at least one hypersphere, the counts of all hyperspheres that the incoming point falls within are incremented, and the point is discarded. These counts directly estimate the density of a class of points in the area, and in this light, VS can be regarded as a one-pass density estimator. Yeung and Chow (2002) argue in favour of such non-parametric methods, as they are less restrictive than parametric ones, require very little training, and can easily be adapted under situations with time-varying data. During testing, hyperspheres that the test point falls within are retrieved, and a voting scheme decides the class of the unlabeled test datum. During this testing phase, the VS relies on the same concept as the k-nearest neighbours algorithm to work: points that are closer in space are assumed to be more likely to belong to the same class. A discussion of similarities and differences of VS to the nearest-neighbour family of algorithms is given in Section 4.3.1.

The VS algorithm is simple to implement, very efficient, and gave a better classification performance than published results on other techniques (on the majority of the data sets tested in this chapter) while being able to handle millions of data points, as opposed to several thousands for the vanilla SVM. For example on training and testing sets of size $4,898,424$ and $311,029$ respectively, the entire program takes on average five minutes to both train and test on an Intel Core 2 running at 2.13 GHz, with 3.25 GB of RAM. This is by far the fastest of the algorithms we have run in our empirical study (while going through all of the data); faster than Tsang et al. (2005)'s CVM that uses subsampling and therefore does not see all the data. Another very important advantage is that the VS can incorporate new data into it whenever it arrives, without the need to re-train from scratch. After training is complete, should the need to train the VS further arise, we simply follow the same training procedure: if the new point does not fall into any of our current hyperspheres, create a new one centred around the point, otherwise increment all spheres it does fall within and discard the point. This is very advantageous in fields such as intrusion detection, since a lot of 'after-the-event' data could be labeled by administrators and used by the VS to keep it updated. This also helps to track any shifts in the data, as this constant incremental training would allow the VS to learn new attacks or the changing definition of 'normal' as they happen.

The following is a more formal description of the algorithm, shown in Algorithm 8. We assume that all inputs $\mathbf{x} \in \mathbb{R}^d$ and that labels $y \in \mathcal{Y}$. At all times, we maintain the centres of all the hyperspheres, and their counts (weights). Initially, we start with empty centres $V_y = \phi$. This will make the first encountered training point of each class its first

Algorithm 8 Voted Spheres training algorithm

Input: labeled training set $\langle(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\rangle$, where $\mathbf{x}_t \in \mathbb{R}^d$, $y_t \in \mathcal{Y}$, $R_y \in \mathbb{R}$, a maximum radius for each distinct class $y \in \mathcal{Y}$

Output: O, which is a triple $\langle \mathbf{v}, C_\mathbf{v}, y_\mathbf{v}\rangle$, where $\mathbf{v} \in \mathbb{R}^d$ is a hypersphere centre, $C_\mathbf{v} \in \mathbb{N}$ is its associated count (weight), and $y_\mathbf{v} \in \mathcal{Y}$ is the hypersphere's associated class.

Initialise: $V_y \leftarrow \phi \; \forall y \in \mathcal{Y}$
for $t = 1, 2, \ldots, N$ **do**
  inSphere $\leftarrow$ false
  for $(\mathbf{v} \in V_{y_t})$ **do**
    if $(d(\mathbf{x}_t, \mathbf{v}) < R_{y_t})$ **then**
      inSphere $\leftarrow$ true
      $C_\mathbf{v} \leftarrow C_\mathbf{v} + 1$
    end if
  end for
  if inSphere=false **then**
    Create new hypersphere $\mathbf{v} \leftarrow \mathbf{x}_t$
    $V_{y_t} \leftarrow V_{y_t} \cup \{\mathbf{v}\}$
    Initialise count: $C_\mathbf{v} \leftarrow 1$
  end if
end for
$O \leftarrow \phi$
for $y \in \mathcal{Y}$ **do**
  for $\mathbf{v} \in V_y$ **do**
    $O \leftarrow O \cup \{\langle \mathbf{v}, C_\mathbf{v}, y_\mathbf{v}\rangle\}$
  end for
end for
Return O

corresponding hypersphere. In the training process, when a point $\mathbf{x}_t$ is received, it is checked against all hyperspheres with a centre of the same label as y_t. This is done by checking whether the point is further than R_{y_t} from the hypersphere centre, where R_{y_t} is a predefined radius for class y_t. Assume P is the set of hyperspheres (of class y_t) that $\mathbf{x}_t$ lies within, then all of the counts of those hyperspheres will have their counts incremented i.e. $C_p = C_p + 1 \; \forall p \in P$. If, on the other hand, the set P is empty, then a new hypersphere is created, with point $\mathbf{x}_t$ at its centre. Note that we only make a single pass through the training set, so each training point is observed but only once.

During testing (show in Algorithm 9), when a point $\hat{\mathbf{x}}_j$ comes in, the counts of the hyperspheres that this point lies in are summed up independently for each class. Assume P_y is the set of hyperspheres that $\hat{\mathbf{x}}_j$ falls in where the associated label of the hypersphere centre is y. We then obtain a count of votes T_y for each possible class y by

$$T_y = \sum_{p \in P_y} C_p$$

The point $\hat{\mathbf{x}}_j$ is then classified as $\hat{y}_j = \arg\max_y T_y$. However, it is possible that the

Algorithm 9 Voted Spheres prediction algorithm

Input: $\mathbf{x}$, O, $R_y \; \forall y \in \mathcal{Y}$
Output: $y \in \mathcal{Y}$

Initialise: $T_y \leftarrow 0 \; \forall y \in \mathcal{Y}$, inSphere $\leftarrow$ false
for all $\langle \mathbf{v}, C_{\mathbf{v}}, y_{\mathbf{v}} \rangle \in O$ **do**
  if $(d(\mathbf{x}, \mathbf{v}) < R_{y_{\mathbf{v}}})$ **then**
    inSphere $\leftarrow$ true
    $T_{y_{\mathbf{v}}} = T_{y_{\mathbf{v}}} + C_{\mathbf{v}}$
  end if
end for
if inSphere = false **then**
  for each class $y \in \mathcal{Y}$ **do**
    $\mathbf{p}_y \leftarrow \arg\min_{\mathbf{v}} d(\mathbf{x}, \mathbf{v})$ such that $y_{\mathbf{v}} = y$
  end for
  $y \leftarrow \arg\max_y C_{\mathbf{p}_y}$
  Return y
else
  $y \leftarrow \arg\max_y T_y$ //a set
  if $|y| = 1$ **then**
    Return $y \in \mathcal{Y}$
  else
    Return $y \in \mathcal{Y}$ of closest hypersphere to $\mathbf{x}$ (1-NN rule).
  end if
end if

test point does not lie in any of the hyperspheres. When this is the case, the closest hypersphere to $\hat{\mathbf{x}}_j$ from each class is obtained. The class of the sphere with the highest vote is chosen. This is based on the fact that since the highest density in the area is of that class, there is a higher possibility that $\hat{\mathbf{x}}_j$ will also belong to that class. However, in the case where the sums of the votes of these closest hyperspheres are equal, the VS uses the nearest neighbour rule. Assume, without loss of generality, that we are dealing with a two-class problem, and the situation arises as in Figure 4.1. In this case, the current test point (indicated by a triangle) lies outside all hyperspheres, and the two closest hyperspheres from either class carry the same weight (C1 = C2 = n). Instead of the VS making a random guess, we let the distances between the point and the hyperspheres be the deciding factor. If d1<d2, the test point will be classified as the dark class, otherwise it would classified as the light class.

The training phase of the VS is shown in Algorithm 8 and the testing stage is given in Algorithm 9, where $d(\mathbf{x}, \mathbf{v})$ is the Euclidean distance between $\mathbf{x}$ and the point $\mathbf{v}$, which can be replaced with any distance measure. Flowcharts describing the training and testing phases are given in Figure 4.2 and Figure 4.3 respectively.

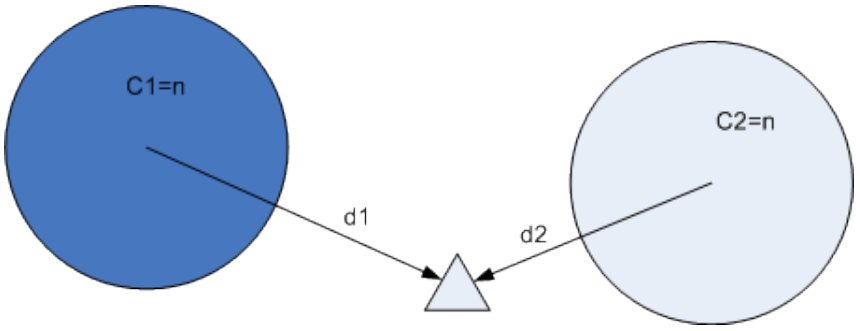


FIGURE 4.1: Plot showing a point outside all hyperspheres generated by VS. d1 and d2 are the distances of the point from the first and second hyperspheres respectively, and C1 and C2 are the votes associated with the hyperspheres.

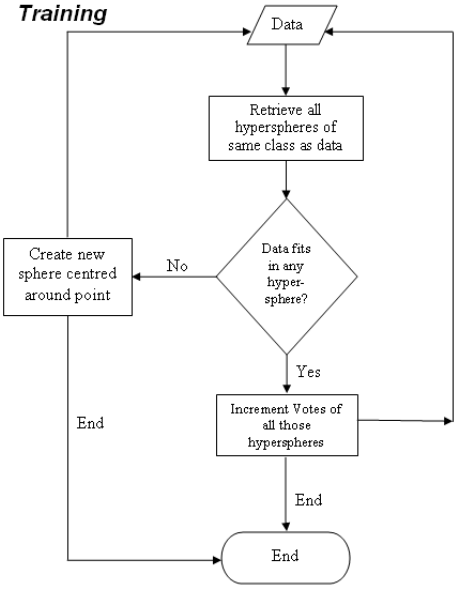


FIGURE 4.2: The flowchart for training the Voted Spheres

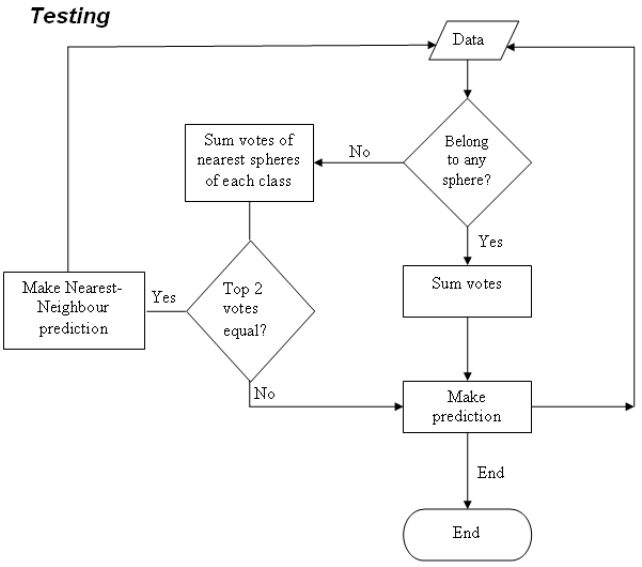


FIGURE 4.3: The flowchart for the testing phase of the Voted Spheres

4.2.1 The hyperparameter

Like any machine learning technique, the VS has a hyperparameter that needs tuning. However, unlike other methods whose hyperparameters can take values in a large range, we are able to greatly limit the range of values our hyperparameter can take. We do this by identifying an appropriate range for the radius by subsampling a small fraction of the data and calculating the average pairwise distance of all points to each other. This is necessary as different data sets can have Euclidean distances in very different ranges. Once this information is known, we can apply cross validation on values in the vicinity of the obtained average. Given the speed of our one-pass VS, we can afford to perform cross-validation on more radii values than other techniques, such as the SVM. We use

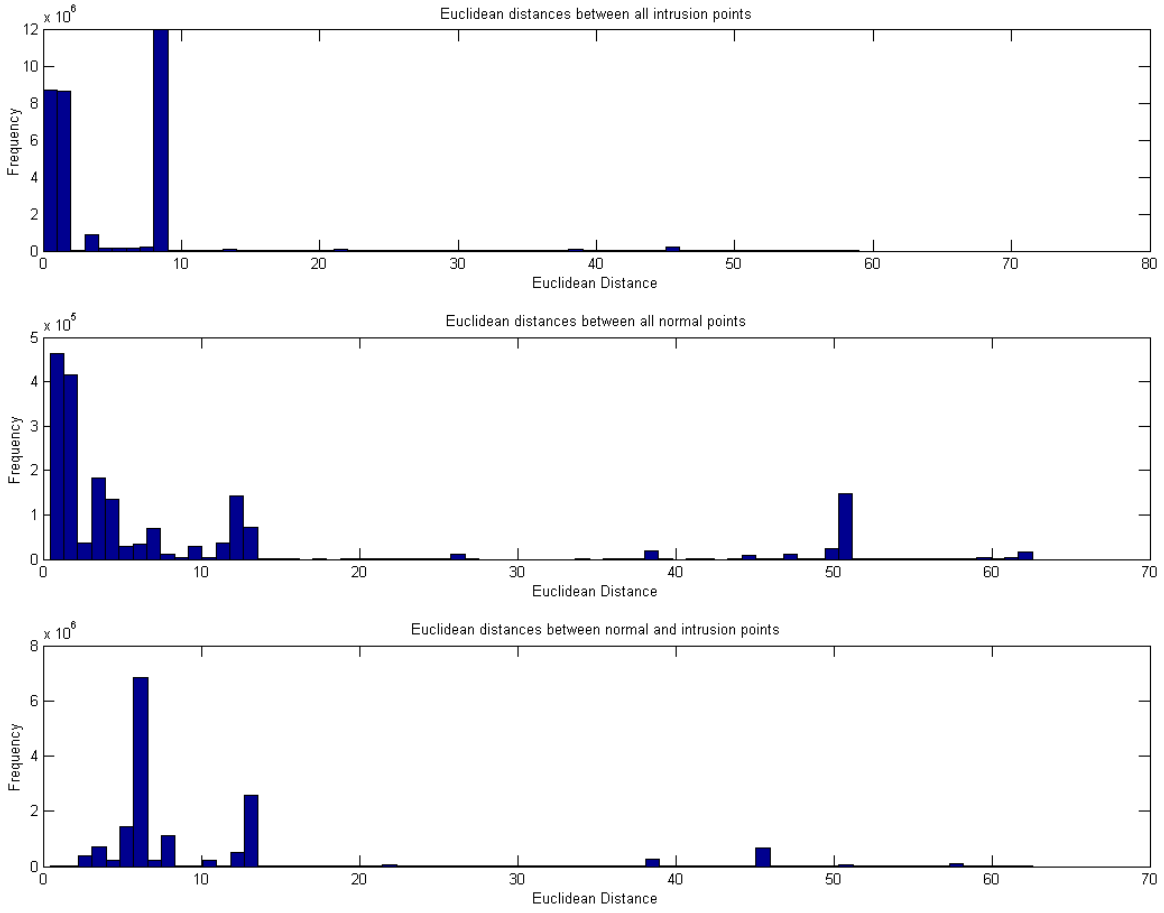


FIGURE 4.4: Histograms showing the Euclidean distances between the different classes of points of the KDD '99 Cup data

this approach on the KDD '99 data, and histograms showing the pairwise Euclidean distances between normal points among each other, intrusion points among each other, and between normal and intrusion points are given in Figure 4.4. From the plot of intrusions vs normals, we can see that there are very few normals and intrusions within a Euclidean distance of 2 from each other, and the majority of distances are less than 14. We use this information to create an initial plot (see Figure 4.5) of the prediction accuracy against the different combinations of the two hyperparameters (maxIDis: the radius for intrusion connections and maxNDis: the radius for normal connections) using $10,000$ randomly selected training points from the training set of the KDD '99 data. From the figure, we can see that the algorithm is very stable with respect to its hyperparameters. For sufficiently large values of maxIDis in which the hyperspheres of both classes completely overlap, the algorithm degrades to a random classifier. This is because the majority (if not all) of the test points would lie in intrusion spheres (regardless of whether they fall into normal ones as well), and due to the heavy imbalance of the two classes in the KDD '99 data, the test points would be outweighed by the intrusion votes. We also observe that many different combinations give a prediction accuracy of over 97%, especially within the $(0.5, 3)$ range for both paramters. To see this area more

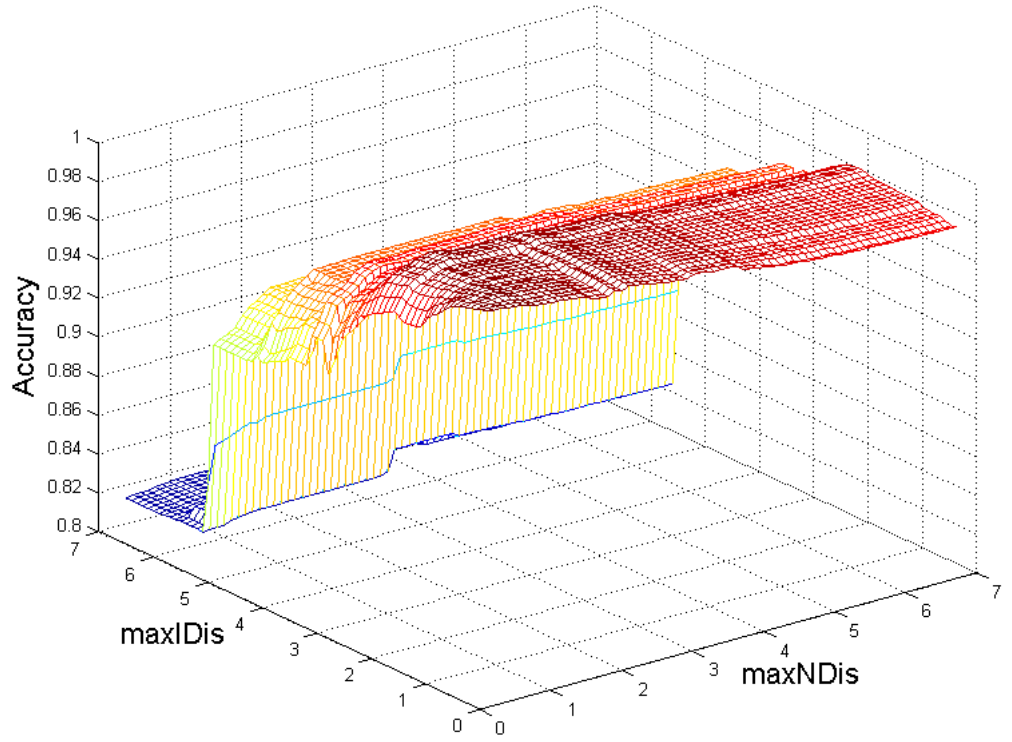


FIGURE 4.5: Generalisation performance of VS for different values of maxIDis and maxNDis on the KDD '99 data. Note that the z-axis starts from 0.8.

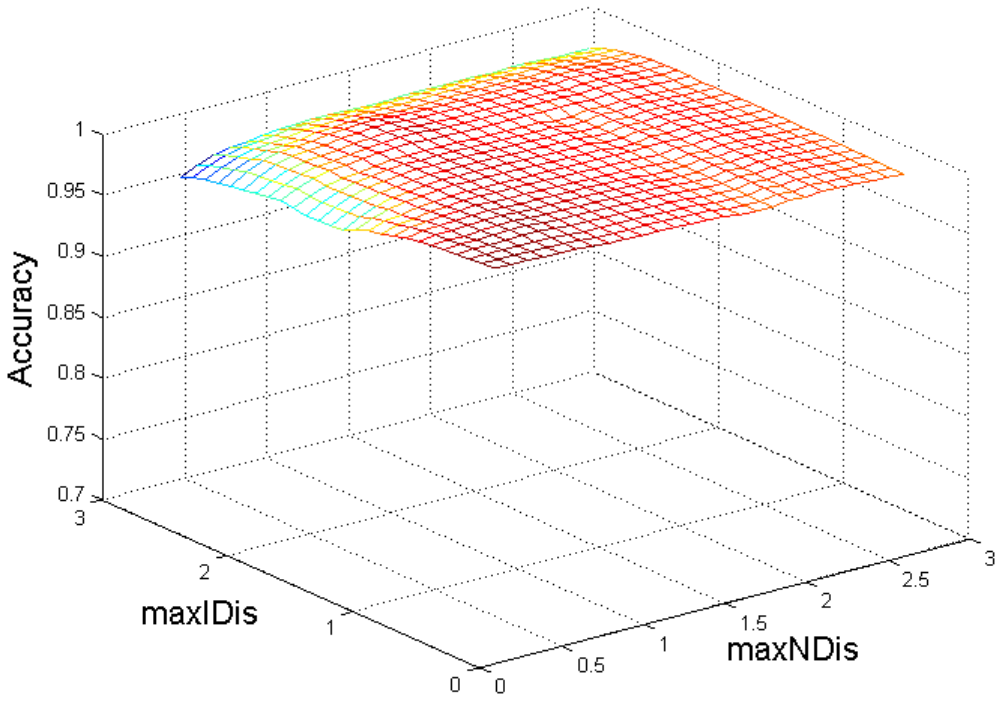


FIGURE 4.6: VS generalisation on 10,000 randomly sampled data points (from the KDD '99 data) for different combinations of maxIDis and maxNDis

clearly, a second plot focused on that region is given in Figure 4.6, which shows that all the combinations of radii gave prediction accuracies of over 97%. Such plots can greatly assist in selecting the range of hyperparamters for which to perform cross-validation to obtain the best radii, which is described below.

The final step to obtain the optimal hyperparameters is to perform k-fold cross-validation. The available training data is divided into k subsets. Each time, one of the k subsets is used as the test set, while the other $k-1$ are put together to form a training set. Then the average generalisation accuracy over all k trials is calculated. Every data point in our original set appears in a test subset only once, and appears k times as part of a training subset. We perform this method on a set of possible hyperparameter values, and the value that gives the highest generalisation accuracy over the five folds is used on the test set. An obvious (yet unavoidable) disadvantage with this is that the training algorithm has to be re-run k times, consuming more time. However once this is over, and the optimal hyperparameters have been obtained, we reap the benefits during testing by having a classifier that is not overfit.

4.2.2 Properties of the Voted Spheres

For very small radii, in which every input point becomes a hypersphere, the VS will reduce to a 1-NN classifier. For such small radii on a randomly subsampled set of 10,000 points from the training set, and a randomly subsampled 10,000 point subset from the shifted testing set, we get the plot of the radii vs prediction rate given in Figure 4.7, where the circled point indicates the 1-NN value. The prediction accuracy achieved when each training point was a hypersphere was 92.77%, which is lower than in other regions of the plot where the prediction reaches a maximum of 93.71%. Even if the 1-NN were to perform equally well or slightly better, VS has the advantage of being much more efficient computationally, allowing it to be applicable on very large data sets. Empirical results comparing VS against other techniques in the literature (including k-NN) are given in Section 4.5, Section 4.6, and Section 4.7. On the 10% KDD '99 Cup data, the VS achieved a 94.70% prediction accuracy, which outperforms the KDD '99 Cup winner (Pfahringer (2000) achieves 92.71%).

The testing time complexity of VS is clearly dependent on the number of hyperspheres generated. Let the total number of hyperspheres created in the training phase be m. Then for each test point $i \in M$, VS loops through the m hyperspheres, making the time complexity to classify the entire testing set $O(m \cdot M)$. Since $m \ll N$ for appropriately chosen radii, the VS has a linear increase in time (see Figure 4.18 in Section 4.7), and is much more efficient than $O(M \cdot N)$ algorithms such as the NN. Using $R_{-1} = 2.6$ and $R_1 = 0.6$, which are the radii obtained by cross-validation (in Section 4.7) and reported in Table 4.3, we get the plot of hyperspheres versus sample size shown in Figure 4.8 on the 10% KDD data. The plot was generated using five random shuffles of the

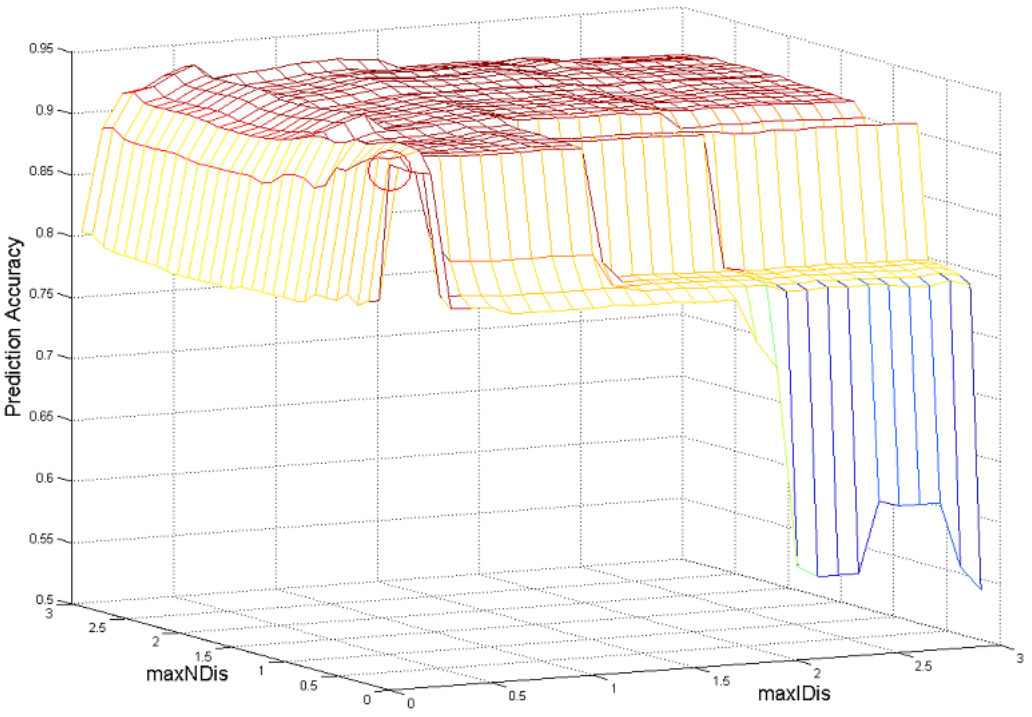


FIGURE 4.7: Prediction accuracy for VS on subsets of the KDD '99 training and test sets, where the circled point indicates the achieved rate when all training points are hyperspheres.

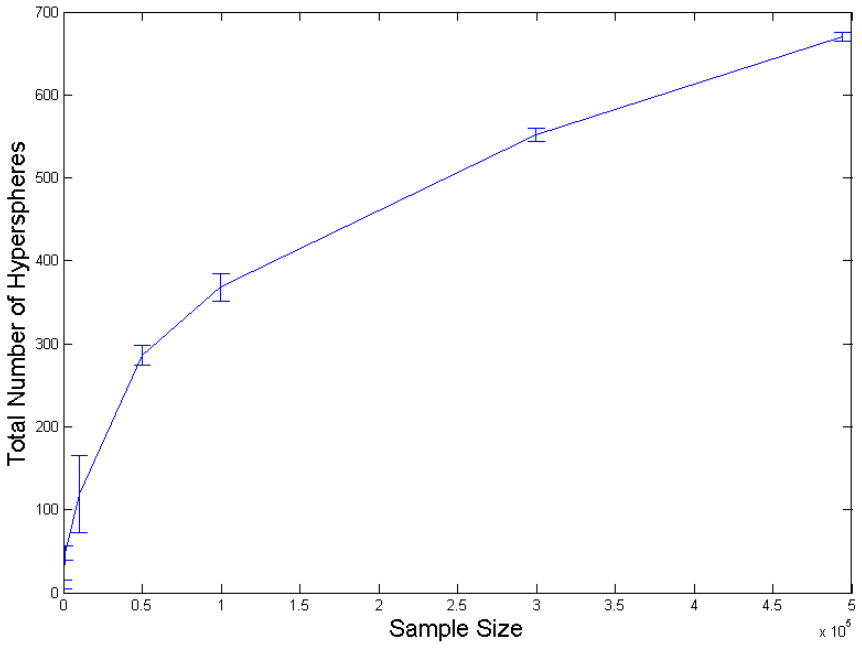


FIGURE 4.8: Number of hyperspheres generated by VS as a function of the sample size. The radii were $R_{-1} = 2.6$ and $R_1 = 0.6$ (obtained via cross-validation in Section 4.7), and the data were subsets of the KDD '99 data. Five random shuffles of the data were used to obtain the uncertainties.

data, indicating the stability of VS to the order of data input, since different random shuffles gave almost the same number of hyperspheres. Had the VS been very sensitive to the order of data input, the uncertainties for each point would be much larger. Since training points in the earlier stages of the VS are more likely to become hyperspheres, the number of hyperspheres generated grows sublinearly with the size of the data. Once the VS has obtained a good partition of the input space, hypersphere creations are fewer. This is demonstrated in Figure 4.9, where we run the VS on the intrusion data from the 10% KDD '99 subset, using $R_{-1} = 2.6$ (the radius for the intrusion spheres reported in Table 4.3). On the x-axis, we have the training sample number (which is equivalent to time since the VS is an online one-pass algorithm), and we plot the number of hyperspheres created on the y-axis. Note that out of the $396,743$ intrusion points, only 79 were hyperspheres. At the initial stages of training, points are more likely to become hyperspheres, as the algorithm has not obtained a sufficiently good partition of the space. With time, input points are less likely to become hyperspheres, for instance between the last two points in the figure, no hypersphere was created for $\sim 127,000$ points. Finally, the space complexity is simply related to the number of stored hyperspheres, making it $O(m)$.

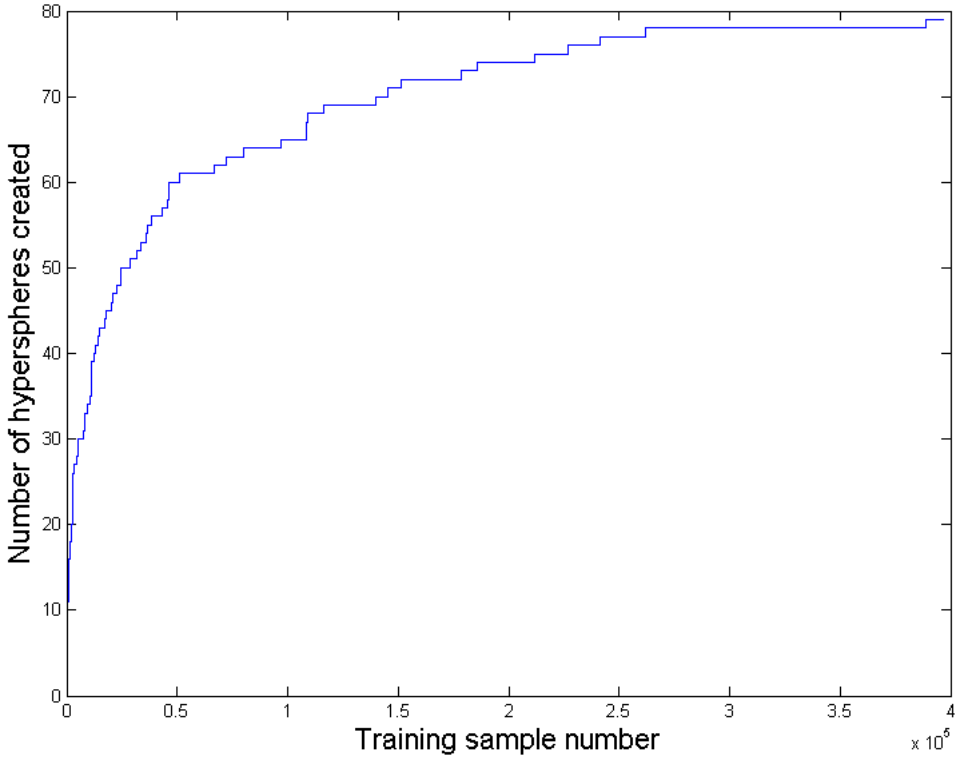


FIGURE 4.9: A plot showing the number of hyperspheres created versus the training sample number on the intrusion data of the 10% KDD '99 data.

Since the order of data input can cause logical inaccuracies for online algorithms, we explore the VS' robustness to this by giving a plot of prediction accuracy on the test set against 35 random shuffles of the 10% KDD training data with the same radii ($R_{-1} =$

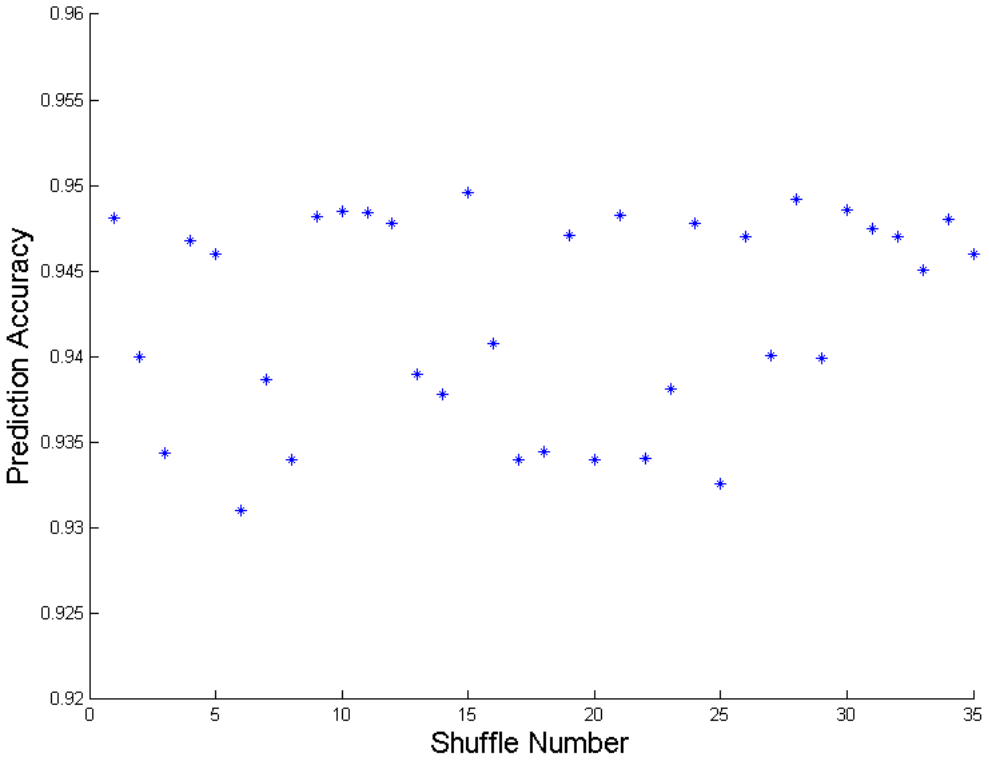


FIGURE 4.10: A plot of prediction accuracy on the test set for 35 different shuffles of the 10% KDD '99 training data using $R_{-1} = 2.6$ and $R_1 = 0.6$. Note the range of the y-axis $\{0.92, 0.96\}$

2.6 and $R_1 = 0.6$) as above. This is given in Figure 4.10. The results show a slight perturbation in the performance, which is an expected consequence of algorithms that process data one at a time as they arrive. However, the prediction accuracy of each run for the VS lies between $\{93\%, 95\%\}$, which is a very small range indeed. Despite seeing the data in a different order, the VS creates a very similar number of hyperspheres, and gives almost identical performance. We discuss ways to limit the effect of the order of data input in Section 4.4.3.

4.3 Related Methods

In this section we discuss the two most related methods to VS: the edited k-Nearest Neighbours and the Resource-Allocating Network. We systematically point out the similarities and differences of these techniques to VS, and show that despite some similar concepts, they are significantly different.

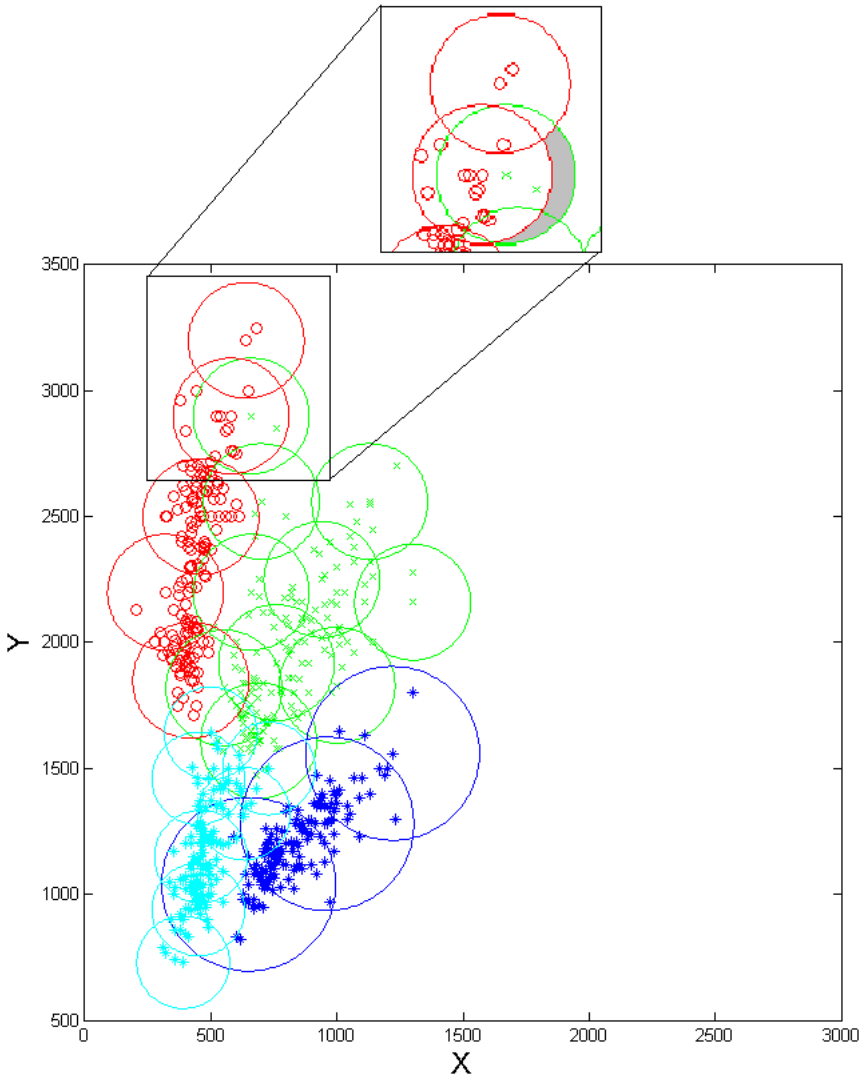


FIGURE 4.11: Hyperspheres generated by the Voted Spheres algorithm on a subset of the two-dimensional data set by Peterson and Barney (1952).

4.3.1 Edited k-Nearest Neighbours

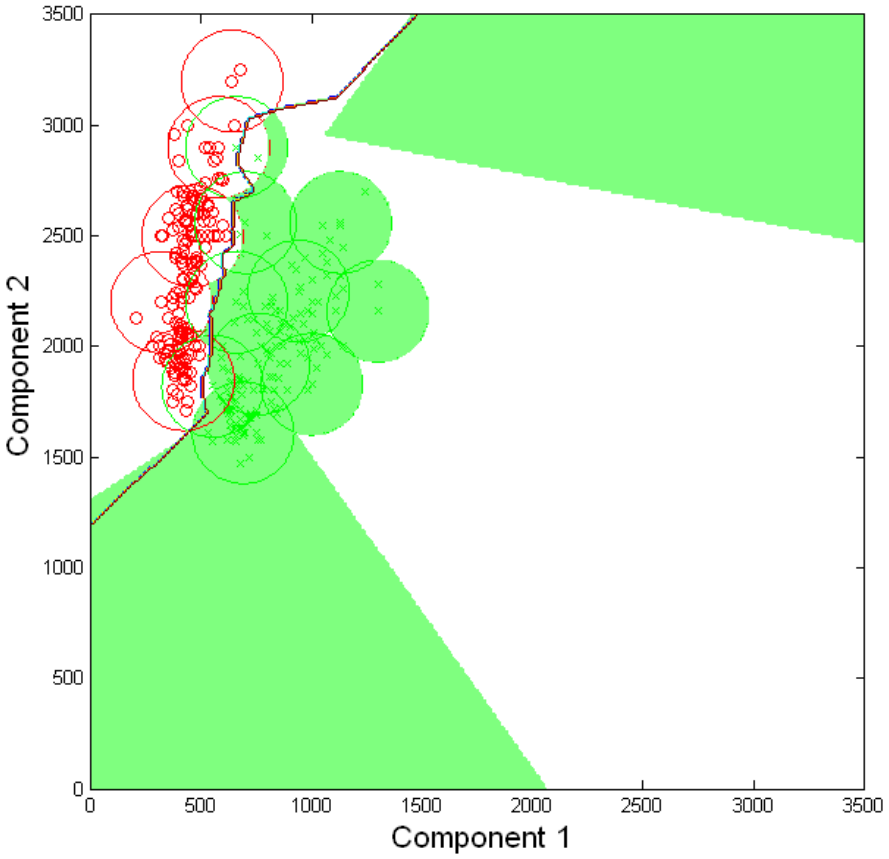
This method, discussed in Section 2.2, differs from VS in two important ways. Firstly, the VS is a constructive algorithm rather than a backwards deletion one, making it computationally much cheaper. The edited k-NN family of algorithms need to compute all the pairwise distances of the data set before being able to make decisions on the reference set; a task that is infeasible on very large data sets. Secondly, the edited k-NN algorithms attempt to 'clean' interclass overlap regions. In the VS, points from different classes that lie close to each other are crucial; the voting scheme would take care of any test points that lie in the overlapping zone. An illustration to clarify this point is given in Figure 4.11, where the radius was set to 230. The edited k-NN would have eliminated the point within the green hypersphere, since upon arrival, the k nearest neighbours to this point would be red 'o's. In contrast, the VS creates a hypersphere in that area, and

points that lie within the grey-shaded area will be classified as the green class. Note that the green hypersphere also affects the classification of test points that lie outside of any hypersphere, since the VS uses the closest hypersphere from each class to make a decision in such cases.

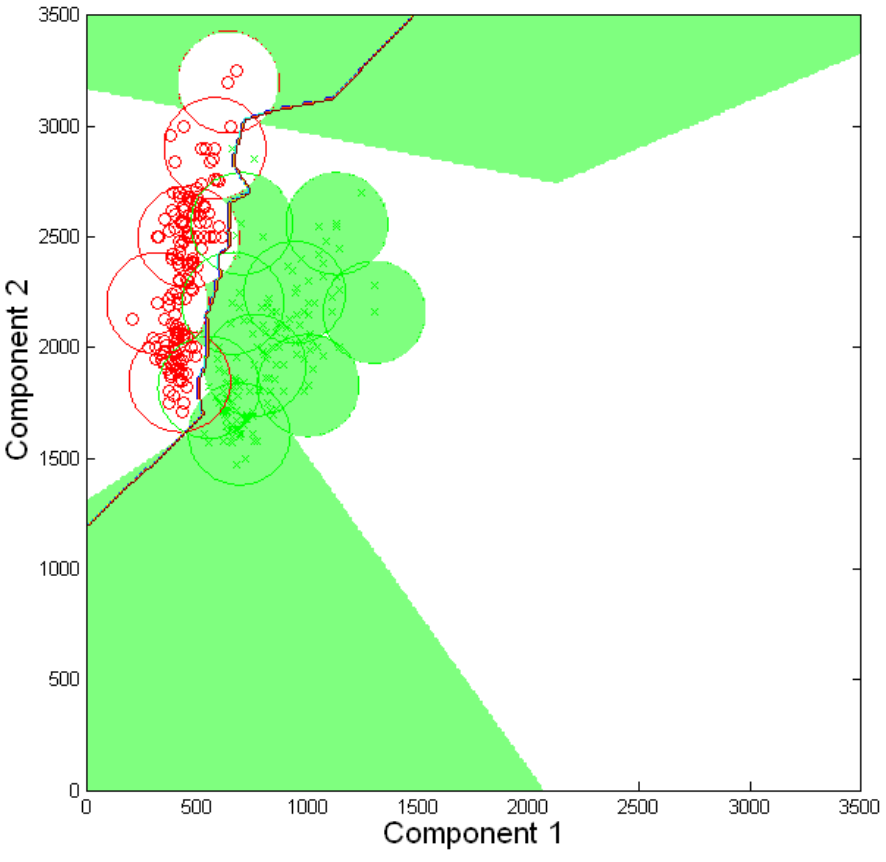
Decision boundaries for the edited k-NN (for $k = 1$) and VS are given in Figure 4.12(a), where the green shaded area depicts the area that VS will classify as green, and the unshaded area is what VS will classify as red. The solid line depicts the edited k-NN decision rule, where $k = 1$, since the VS would default to this in the worst case as described in Section 4.2.2. What is important from Figure 4.12(a) is that the decision boundary of VS and the edited k-NN are almost identical in the region between the two classes (where the edited k-NN boundary lies), despite the VS being much more computationally efficient. The two algorithms disagree only in the area of interest shown in Figure 4.11. The reason for this is clear: the red class is more dense in this region, and the red votes would overshadow the green ones in the overlapping zone of the two hyperspheres. The difference in the boundary elsewhere in the input space arises from the difference in the way the algorithms work: the k-NN algorithms check the classes of the nearest k points, whereas the the VS checks how many points from each class are within a specific distance R. We see that the VS has captured the distribution of the supplied training data, and given the behaviour of the data, it is very rare for points to fall in the disputed region.

Despite giving a similar decision boundary with the VS in the zone between the two classes, the edited k-NN came at a much higher cost. It removed three points out of the 300 training points, at a cost of $O(N^2)$. Testing required a further $O(M.N)$ operations, where M is the size of the test set. Despite this heavy cost, which would render the edited k-NN too slow on large data, the edited k-NN classified only one extra point correctly compared to the VS. To see the effect on the VS' decision boundary if we remove hyperspheres, we remove hyperspheres that have a vote ≤ 2. This removes only one green hypersphere, the one shown in Figure 4.11. A plot showing the resulting decision boundary is given in Figure 4.12(b). We see that the removal of one single hypersphere caused a large difference in the decision boundary in the top left corner. In this region, the VS decision boundary is no longer similar to the edited k-NN's, where the latter seems to be more intuitively correct. This clarifies our second point above that all training points are important for learning, and 'cleaning' overlap regions in the input space loses important information, which the VS uses to create a good separating boundary. A discussion on removing points that are potential outliers is given in Section 4.4.7.

Finally, the edited k-NN reference set would not be an accurate depiction of the data's density distribution since it eliminates points that could carry important information, whereas in the VS the training part of the algorithm can be used as a one-pass density estimator while still compressing the size of its reference set.



(a)



(b)

FIGURE 4.12: Decision boundaries on a subset of the two-dimensional data set by Peterson and Barney (1952) for (a) VS and edited k-NN with $k = 1$, and (b) VS after removing hyperspheres with count ≤ 2, and edited k-NN with $k = 1$.

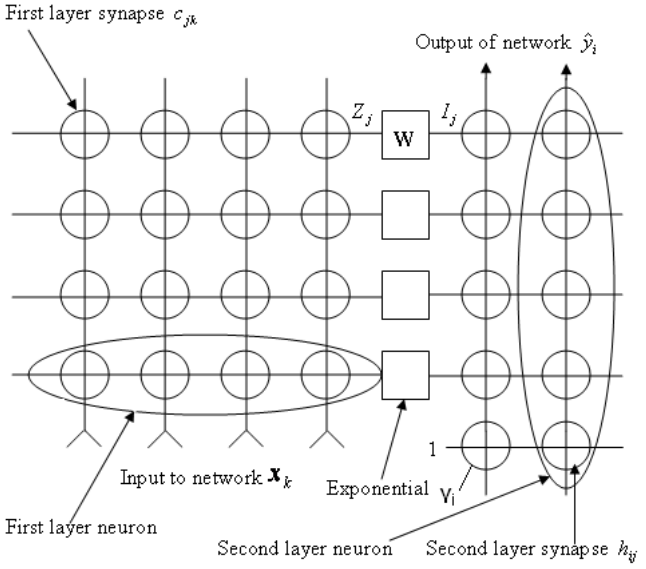


FIGURE 4.13: The architecture of the RAN

4.3.2 Resource-Allocating Network

Platt (1991) created a network that allocates a new computational unit every time an unusual input point is presented to the network, similar in fashion to the VS. The method, which is called a resource-allocating network (RAN) consists of a network, a strategy for allocating new units, and a learning rule for refining the network. The network itself is two-layered, and a figure is given in Figure 4.13 to aid the description below.

The first layer is made up of units that respond to a local region of the space of input values, while the second layer combines outputs from the first layer and creates a function that approximates the input-output mapping over the entire space. In the first layer, when the input moves away from the stored unit its response decreases. A simple function that implements a locally tuned unit is a Gaussian:

$$z_j = \sum_k (c_{jk} - x_k)$$

$$I_j = \exp\left(z_j / w_j^2\right)$$

To speed up the algorithm, an approximation to a Gaussian is used instead:

$$I_j = \begin{cases} \left(1 - (z_j / q w_j^2)\right)^2, \text{ if } z_j < q w_j^2; \\ 0, \text{ otherwise} \end{cases}$$

where $q = 2.67$ was chosen empirically to make the function best fit a Gaussian. The

output of the network, $\hat{\mathbf{y}}$ is defined as follows (see Figure 4.13):

$$\hat{\mathbf{y}} = \sum_j \mathbf{h}_j I_j + \boldsymbol{\gamma}.$$

As can be seen from the equation, the network output $\hat{\mathbf{y}}$ is the sum of the first-layer outputs I_j, each of which is weighted by the synaptic strength $\mathbf{h}_j$, plus a constant vector $\boldsymbol{\gamma}$, which is independent of the first layer outputs. The purpose of each second-layer synapse is to define the first-layer units' contribution to $\hat{\mathbf{y}}$.

As the RAN's learning progresses, the algorithm chooses to store some input points that are presented to it. As in the VS, at any point in time the RAN has a current state which reflects all the training points seen to far. Also, in a similar fashion to VS, a training point $\mathbf{x}$ that is not well represented by the RAN causes a new unit to be allocated centred around it:

$$\mathbf{c}_n = \mathbf{x}$$

The synapses on the second-layer are set to the difference between the output of the network and the newly allocated unit:

$$\mathbf{h}_n = \mathbf{y} - \hat{\boldsymbol{y}}$$

The width of the response of the new unit is proportional to the distance from the nearest stored vector to the novel input vector ($\mathbf{x}$):

$$w_n = \kappa \left\| \mathbf{x} - \mathbf{c}_{nearest} \right\|.$$

As in the VS, the units in the RAN are allowed to overlap, and the larger κ (the overlap factor) is, the more the units overlap. Furthermore, in contrast to VS, the RAN uses a two-part novelty condition. An input-output pair $(\mathbf{x}, \mathbf{y})$ is considered novel if:

$$\left\| \mathbf{x} - \mathbf{c}_{nearest} \right\| > \delta(t)$$

and

$$\left\| \mathbf{y} - \hat{\mathbf{y}}(\mathbf{x}) \right\| > \epsilon$$

where ϵ is the desired accuracy of the network outputs. The distance $\delta(t)$ is the scale of resolution that the network is fitting at the tth input presentation. This is similar to the radius R in the VS (which remains constant throughout training), however in the RAN, δ changes upon the arrival of each new point. Whenever the error is larger than ϵ, a new unit is allocated in the RAN, whereas errors smaller than ϵ are fixed using gradient descent. This differs from the VS since in the RAN, the network output at the current input point $\mathbf{x}$ is taken into account, whereas in VS, only the first condition is used (no gradient descent is used). The main difference between the two, however, is that VS carves up the input space for densities, whereas in the RAN the target is to interpolate

a function. More details are available in the original paper (Platt (1991)).

4.4 Extensions to VS

In its general form, the algorithm is a fast effective multi-class classifier. There are plenty of extensions however, that could be considered for different application settings, some of which will be discussed in this section.

4.4.1 Kernelised VS

Given that the Euclidean distance between two points in our VS algorithm is:

$$d(\mathbf{x}, \mathbf{x}') = \left\|\mathbf{x} - \mathbf{x}'\right\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{x}', \mathbf{x}' \rangle - 2 \langle \mathbf{x}, \mathbf{x}' \rangle},$$

we can project our data into a higher dimensional feature space by replacing the dot products between our points with kernel evaluations as follows:

$$d(\mathbf{x}, \mathbf{x}') = \left\|\phi(\mathbf{x}) - \phi(\mathbf{x}')\right\| = \sqrt{k(\mathbf{x}, \mathbf{x}) + k(\mathbf{x}', \mathbf{x}') - 2k(\mathbf{x}, \mathbf{x}')}.$$

A linear solution in this kernel-induced feature space would correspond to a non-linear one in our input space (as discussed in Section 2.2). Results for VSKernel are given in Table 4.3 in Section 4.7.

4.4.2 VS with distance modification (VSDistance)

In this modification, the VS takes the distances of test points from hypersphere centres into consideration when testing. During testing, while we sum the weights of hyperspheres in which our test point falls, we divide those weights by the corresponding distances from the point to the centres of the hyperspheres:

$$T_y = \sum_{p \in P} \frac{C_p}{d(\hat{\mathbf{x}}_j, \mathbf{v}_p)}$$

where $p \in P$ is a sphere belonging to class y that $\hat{\mathbf{x}}_j$ falls in. Note that only the prediction rule has changed, training is identical to the vanilla VS. The need for doing this becomes obvious in cases where the decision is borderline, as in Figure 4.14. The incoming test point, shown as a green square, falls outside of all hyperspheres, but belongs to the blue class. The closest hypersphere of either class is retrieved, and under the normal VS, the point would be classified as red since the red hypersphere carries a larger vote. However, this would be incorrect, and the test point is clearly closer to the blue hypersphere in input space. Dividing the counts by the distance would correct the error. We refer to

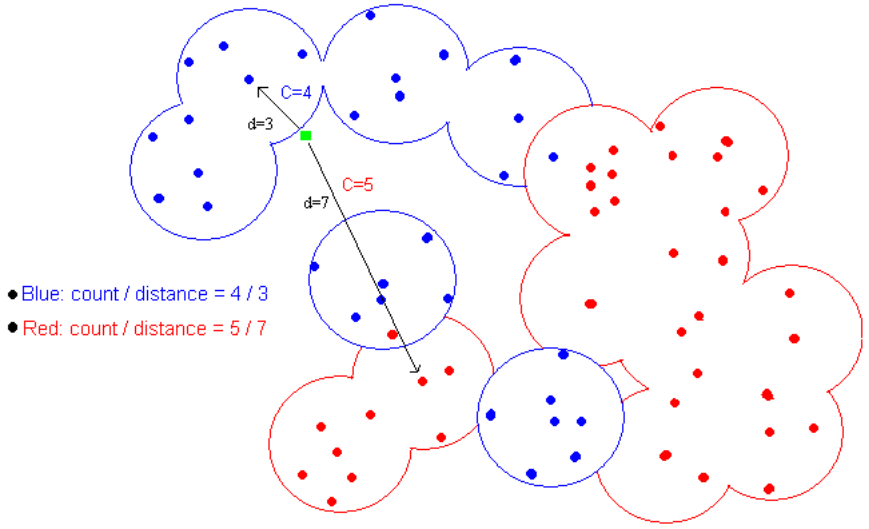


FIGURE 4.14: Plot showing a test point falling outside of the hyperspheres created by VS. The 'd's represent the Euclidean distances of the points to the hypersphere centres, while the 'C's represent the counts each hypersphere carries.

this modification as VSDistance. On the KDD '99 Cup data, this modification achieved a 94.81% generalisation accuracy, and an AUC of 0.9857 which is the highest among all other algorithms. These results are given in Table 4.3 and an ROC curve is shown in Figure 4.20.

4.4.3 VS with clustering (VSCluster)

The order of input of the training data affects all online algorithms to some extent. Being able to minimise this effect would be very beneficial, as different shuffles of the data would not cause major perturbations in the accuracy. To do this for the VS, one could consider shifting the centres of the hyperspheres whenever a point falls within them during training. This means that when a point $\mathbf{x}_i$ falls in a hypersphere with centre $\mathbf{v}_{k_{y_i}}$, then that point is added to A, where A is the set of points that fall into $\mathbf{v}_{k_{y_i}}$. The centre is then redefined as the average of all points in A:

$$\mathbf{v}_{k+1_{y_i}} = \frac{\sum_{a \in A} \mathbf{x}_a}{|A|}.$$

We refer to this variant as VSCluster, as the averaging process in effect moves the origins of the hyperspheres toward the centre of dense clusters of points. Note that we use an incremental update, thus avoiding the need to store any data points. Since the centre $\mathbf{v}_{k_{y_i}}$ is the arithmetic average of all the points that have fallen inside it, and the number

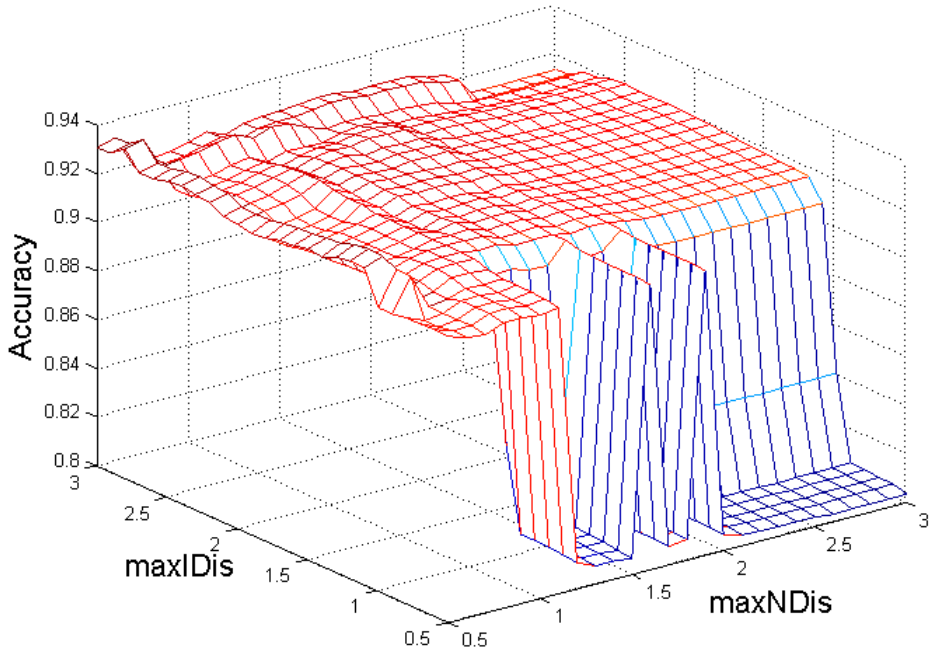


FIGURE 4.15: Plot showing the generalisation accuracy of VSCluster on the 10,000 random points from the KDD '99 data, where maxIDis is the maximum hypersphere radius for the intrusion class, and maxNDis for the normal class

of those points is simply the vote $C_{k_{y_i}}$, we perform the following update:

$$\mathbf{v}_{k+1_{y_i}} = \frac{(C_{k_{y_i}} * \mathbf{v}_{k_{y_i}} + \mathbf{x}_i)}{C_{k_{y_i}} + 1}$$

A plot showing the generalisation performance of this algorithm against different values of maxIDis and maxNDis (trained using 10,000 points from KDD '99 training data, and tested on 10,000 points from the corresponding test set) is given in Figure 4.15. Results of five-fold CV are given in Table 4.3, and an ROC curve is given in Figure 4.16.

4.4.4 VS with dynamic radius (VSMod)

Another modification was to allow the radius of individual spheres to assume any value between 0 and R_y (the maximum radius of class y). This means that when a training point $\mathbf{x}_t$ is presented to the algorithm, all the spheres P of class y_p that this point can possibly fit in are retrieved (i.e. $d(\mathbf{x}_t, \mathbf{v}_p) < R_{y_t}$). The radii are then recalculated to be the smallest value that incorporates the new point: $R_p = \min(d(\mathbf{x}_t, \mathbf{v}_p), R_y)$. Figure 4.17 shows a plot of the prediction accuracy versus the radii. It is clear that varying the radius in this manner did not produce a stable classifier. An explanation is that when the (fixed) radius of the hyperspheres is chosen correctly, most test points will lie inside

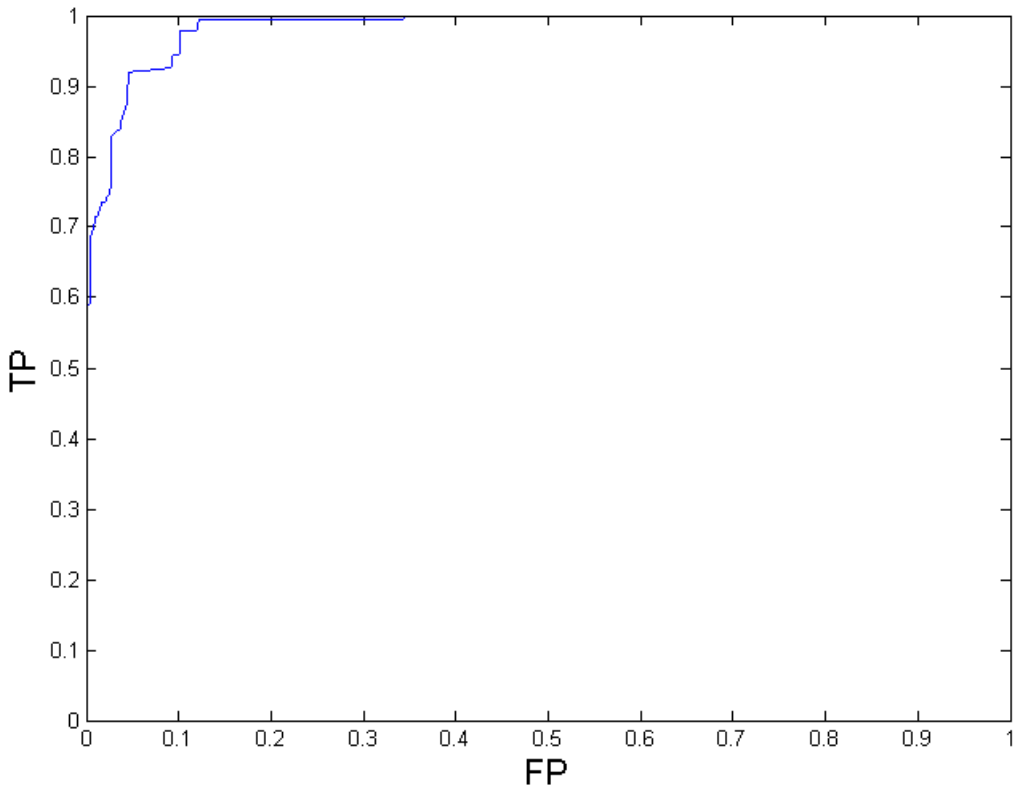


FIGURE 4.16: A ROC curve for VSCluster on the entire 10% KDD subset. The AUC was 0.9827.

hyperspheres, even if no training points lie close to the surface. This was shown to be correct by our experiments, as out of 311,029 test points, usually around 100 lie outside the spheres. However, when the right hyperparameters are chosen, this modification would still be a powerful classifier. We will explore different ways to modify the radius in future work.

4.4.5 VS for the multi-class case (VSMulti)

This is a natural extension of any binary classification algorithm, however in our case the Voted Spheres is already a multi-class classifier. This version was mainly used on the UCI data set of Section 4.6, achieving better prediction accuracies than the published results on the same data. We also run experiments on the multi-class version of the KDD '99 Cup data, using cross validation to obtain the optimal parameter values before applying the VS on the unseen test set. The generalisation performance obtained was 93.55%, which is higher than the winner of the KDD '99 Cup. The VS also easily outperformed Pfahringer (2000) in terms of the computational time required, as the former took five minutes while the latter required over a day. A detailed comparison of

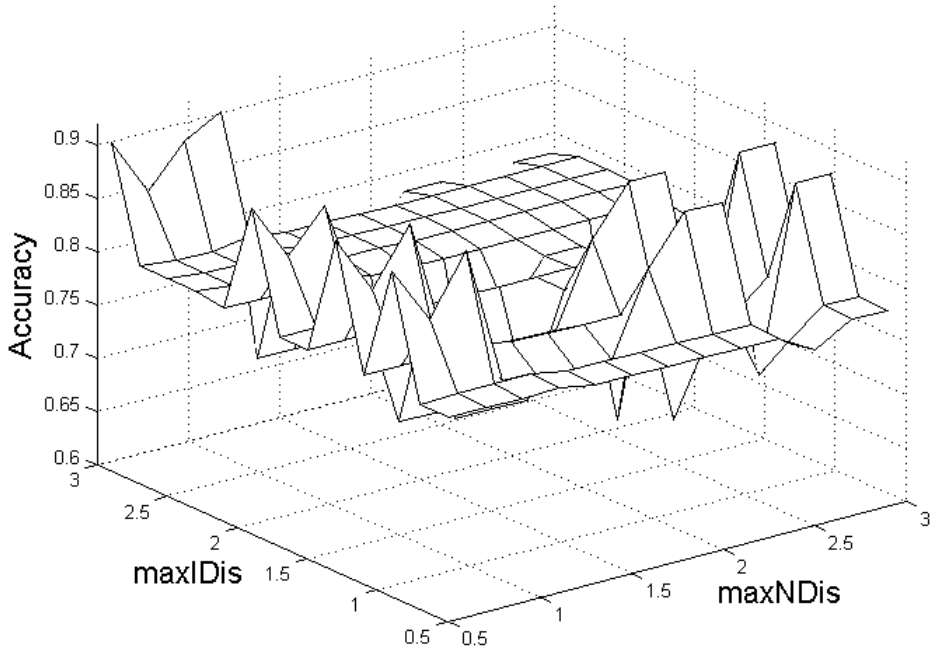


FIGURE 4.17: Generalisation performance of VS with dynamic radius (see Section 4.4.4) for different values of maxIDis and maxNDis on the 10,000 random subset (of the KDD '99 data). Note the unstable behaviour when the radius is modified during training.

VSMulti with other techniques in the literature that have used the multi-class KDD '99 data is given in Section 4.7.

4.4.5.1 On the Letter Data Set

As the letter data set is widely used in the literature, we used it to test our algorithm's generalisation ability for the multi-class case, before applying it on the large KDD '99 data. It has $20,000$ examples, of which the first $16,000$ are used for training, and the last $4,000$ are used for testing. Each example has 16 attributes, followed by a label ranging from A to Z. To obtain a range for the radius, we randomly subsampled 2,000 points, and calculated the average pairwise distance of those points, which returned a value around 12. After using this value and performing five-fold CV (in the range of {2,20}), we obtained a value of $R = 3.5$ and ran the algorithm using the test data. We obtained an accuracy of 83.12%, while the best achieved by other researchers was reported as 'a little over 80%' on the data set's website[4]. Our one-pass Voted Spheres took only 15 seconds to both train and test, and gave state-of-the-art results using only one radius value for all 26 classes.

[4]http://www.cs.toronto.edu/~delve/data/letter/letterDetail.html, last accessed 20/03/2010

4.4.6 VS for the multi-class case with imbalanced data

Here we modify the algorithm to see whether the imbalance in the KDD '99 data between normals and intrusions (1:4 ratio, respectively) affects our algorithm. Before testing, we divided the weight of each class by the number of points in that class. For example for the normal points ($97,277$ in the 10% KDD '99 Cup training data), we divided each count of each hypersphere belonging to the normal class by $97,277$. If VS was sensitive to the imbalance in the data, then this modification would reduce the effect of classes that contain a lot of points, and would therefore prevent large hyperspheres from consuming all the other points. From our experiments, VS is very stable and attempting to account for this imbalance in the data only degrades performance. As an illustration, assume there are two hyperspheres belonging to two different classes that are close to one another. Assume also that an intrusion test point arrived that is equidistant from the centres of both of the aforementioned hyperspheres. If the intrusion hypersphere's weight is twice as large as the normal's, then the regular VS algorithm would correctly classify the point as an intrusion, under the assumption that since the intrusion sphere weighs more, then this must be a more intrusion-dense region. When we modify for imbalance as described above, we would have divided the weight of the normal hypersphere by $97,277$, and the intrusion one by $396,743$, which are the numbers of normal and intrusion points in the 10% KDD '99 data subset respectively. This would have caused our test point to be classified incorrectly as a normal.

4.4.7 Outlier Removal

Another possible modification for the VS is outlier removal. Naturally for an algorithm that attempts to carve the input space, outliers can affect performance negatively. If an outlier from class 1 is deep in a class 2 region in input space, the VS will fit a hypersphere around that point and treat the region around it as if it belongs to class 1. This does not necessarily have to degrade performance significantly, but we can devise pathological cases where it could seriously lower generalisation performance. There are several intuitive ways to do this for the VS. One method could involve removing spheres that have a count of 1 or 2 (i.e. only the centre point is in the sphere or 1 other point, respectively). Another method could involve keeping the m hyperspheres that contain 95% or 99% of the data belonging to its class. There are other ways one could do this, and taking outliers into account can greatly decrease the number of misclassifications. We leave this for future work.

4.5 Results on Gunnar Rätsch's Benchmark Data sets

In this section, we compare the VS with SVMLight by Joachims (1999) (using both RBF and linear kernels) and the Sparse Pseudo-input Gaussian Process Classifier (SPGPC) developed by Naish-Guzman and Holden (2008) on small machine learning data sets. These data were used because they have been widely used in the literature for model selection, for example by Rätsch et al. (2001), Mika et al. (1999), and Cawley and Talbot (2003) among others. They are used for benchmarking purposes, enabling researchers to clearly compare methods with each other. MATLAB implementations of the VS, VSCluster, and VSDistance were used, and we only report the best performing result obtained on a PC running Microsoft Windows XP, with an Intel Core 2 6400 processor running at 2.13GHz with 3.25 GB of RAM. For each data set, we use a small subset of the data to calculate the average pairwise distances. After obtaining this value, the same setup as Rätsch et al. (2001) was used: five-fold cross validation on each of the first five splits, and the median of the optimal parameters from those five splits are used to obtain the generalisation accuracy on the 100 splits. The complex SPGPC model achieved higher prediction accuracies than the VS on the majority of the data used, however the latter is a much simpler online algorithm, whose real benefit arises in its application on massive data sets. Nevertheless, the VS outperformed SVMLight(linear) on five out of the seven data sets used (see Table 4.1), while running significantly faster on all the sets.

4.6 Results on UCI data sets

In this section we use data sets from the UCI repository to further compare the VS to other techniques in the literature that use the same data sets. On data sets where there were no published results, we employed SVMLight using either a polynomial or Gaussian kernel, using cross-validation to obtain the optimal hyperparameters, and reported the best results. If results were reported from the literature, then they were obtained from each data set's description page[5] and the same setup as the authors was used, otherwise for experiments using SVMLight, we shuffled and split the data into 70% training and 30% testing subsets[6]. Both the VSMulti and the VSMultiDistance were used, and only the best results are reported. The results are given in Table 4.2, and show that the VSMulti performed very well against the methods published in the literature using the same data sets, in a fraction of the time. This further proves the robustness of the VS technique, as we have used the results published by other researchers that have conducted their own hyperparameter tuning and testing. The VS also performed competitively

[5] http://archive.ics.uci.edu/ml/datasets.html

[6] Note that since the results of the data were obtained from the website, they may or may not be the state-of-the-art

Data set	Algorithm	Generalisation performance(%)	CPU Time (secs)
Banana	SVMLight(lin)	55.23 ± 1.09	2.4
	SVMLight(RBF)	88.84 ± 0.6	2.63
	VSCluster	87.52 ± 0.68	**0.766**
	SPGPC	**89.3**	N/A
Breast cancer	SVMLight(lin)	71.45 ± 4.6	1.62
	SVMLight(RBF)	$\mathbf{73.04 \pm 4.78}$	0.19
	VSCluster	72.79 ± 4.47	**0.03**
	SPGPC	71.9	N/A
Flare solar	SVMLight(lin)	$\mathbf{67.63 \pm 1.78}$	0.88
	SVMLight(RBF)	67.48 ± 1.72	1.71
	VS	59.07 ± 6.15	**0.07**
	SPGPC	66.2	N/A
Image	SVMLight(lin)	84.66 ± 0.84	4.33
	SVMLight(RBF)	96.84 ± 0.52	1.82
	VSDistance	95.84 ± 0.7	**1.03**
	SPGPC	**96.9**	N/A
Heart	SVMLight(lin)	$\mathbf{82.89 \pm 2.96}$	0.6830
	SVMLight(RBF)	77.72 ± 3.49	0.1866
	VSCluster	78.43 ± 3.68	**0.063**
	SPGPC	82.8	N/A
Ringnorm	SVMLight(lin)	75.24 ± 0.69	4.71
	SVMLight(RBF)	98.34 ± 0.12	3.33
	VSCluster	94.12 ± 14.12	**0.801**
	SPGPC	**98.6**	N/A
Thyroid	SVMLight(lin)	89.80 ± 2.56	0.713
	SVMLight(RBF)	95.08 ± 2	1.10
	VSCluster	91.89 ± 3.10	**0.016**
	SPGPC	**96.3**	N/A

TABLE 4.1: Results of five-fold Cross-Validation experiments on VS algorithms, SVM-Light (using a linear kernel), and SPGPC of Naish-Guzman and Holden (2008) . Only the best performing version of VS and the best achieved results of the SVM are reported (best values are in bold font). The data used were the benchmark data sets used by Rätsch et al. (2001)

against the SVM using RBF and polynomial kernels on the remaining data sets, although requiring much less time.

4.7 Results on the KDD '99 Data

Given the success of VS and its variations on the many data sets used in Section 4.5 and Section 4.6, we move on to employ the VS on the large 10% subset of the KDD '99 data, and the results are given in Table 4.3. Five-fold CV was used to obtain the hyperparameter values, and the results were reported in the same fashion as in the literature on the KDD '99 data. The time reported is for training and testing on

Data	Method	# Data	# Train	# Test	Accuracy
Abalone	**VSMulti**	4177	3133	1044	**24.14**%
	C4.5				21.5%
	LDA				0.0%
	$k = 5$ NN				3.57%
Shuttle landing	**VSMulti**	15	11	4	**100**%
	SVMLight(RBF)				75%
Statlog(Heart)	**VSMultiDistance**	270	189	81	**70.37**%
	SVMLight(RBF)				65.43%
Teaching Asses. Ex.	**VSMultiDistance**	151	106	45	**54.35**%
	SVMLight(RBF)				39.13%
Vowel	**VSMultiDistance**	990	528	462	**55.19**%
	Single Layer Perc.				33%
	Multi Layer Perc.				51%
	RBF				53%
	Gauss. Node Network				55%
Parkinsons	**VSMulti**	195	137	58	**93.22**%
	SVMLight(RBF)				83.05%
Pima Indians	**VSDistance**	768	538	230	**72.29**%
	SVMLight(RBF)				66.67%
Wine(normalised)	**VS**	178	177	1*	**100**%
	RDA				**100**%
	QDA				99.4%
	LDA				98.9%
	$k = 1$ NN				96.1%
Yeast	**VSMulti**	1484	1039	445	**50.31**%
	SVMLight(RBF & poly)				8.90%
Zoo	**VSMulti**	101	71	30	**86.67**%
	SVMLight(poly)				63.33%

TABLE 4.2: Results on different data sets from the UCI repository. *Obtained using Leave-One-Out method

a PC running Microsoft Windows XP, with an Intel Core 2 6400 processor running at 2.13GHz with 3.25 GB of RAM, unless the results were obtained from published papers. We compare our VS method with published results in which the experiments were conducted on either the 10% subset or the entire training data, as many authors resort to biased sampling on the KDD '99 data. When sampling in a biased manner, the data from the rarest classes (PROBE, U2R, and R2L) are always selected, such as by Beghdad (2008) and Wang et al. (2010), which unfairly gives them an advantage as these classes are the most expensive to misclassifiy (as seen in Table 1.4). We evaluate our VS technique on the two-class 10% KDD '99 subset in Section 4.7.1, on the multi-class 10% KDD subset in Section 4.7.2, on the entire ~ 5 million KDD data in Section 4.7.3, and on the binarised KDD data in Section 4.7.4, and show that the VS achieves prediction accuracies at least as high as the state-of-the-art published results in a fraction of the time.

4.7.1 On the Two-class 10% Subset

Initially we tested the simple nearest neighbour technique, which took close to two hours to run despite using a fast implementation by Auton Lab of Carnegie-Mellon University[7]. It did, however, achieve a prediction rate higher than the CB-SVM and the KBHC-CBSVM (see Table 4.3). In general, the NN method is very slow, as it needs to loop through the entire training data for each test point, and is not suitable for use on large scale data.

Our implementation of the CVM showed similar levels of generalisation to those reported in Tsang et al. (2005), but displayed a somewhat unstable behaviour. Significantly different prediction rates were achieved during each run of the algorithm, due to the random subsampling of 59 points as opposed to using intelligent sampling (i.e. non-random). This is shown in Table 4.4, where the CVM has the highest uncertainty for the AUC. Our experiments also showed that the CVM was significantly faster than other SVM implementations, and could be further sped up by using caching and warm start for the QP problem. Despite this subsampling, the CVM is easily outperformed by the VS in terms of computational efficiency. A plot showing the time required by the CVM and VS to run versus the sample size is given in Figure 4.18. We can see that as the

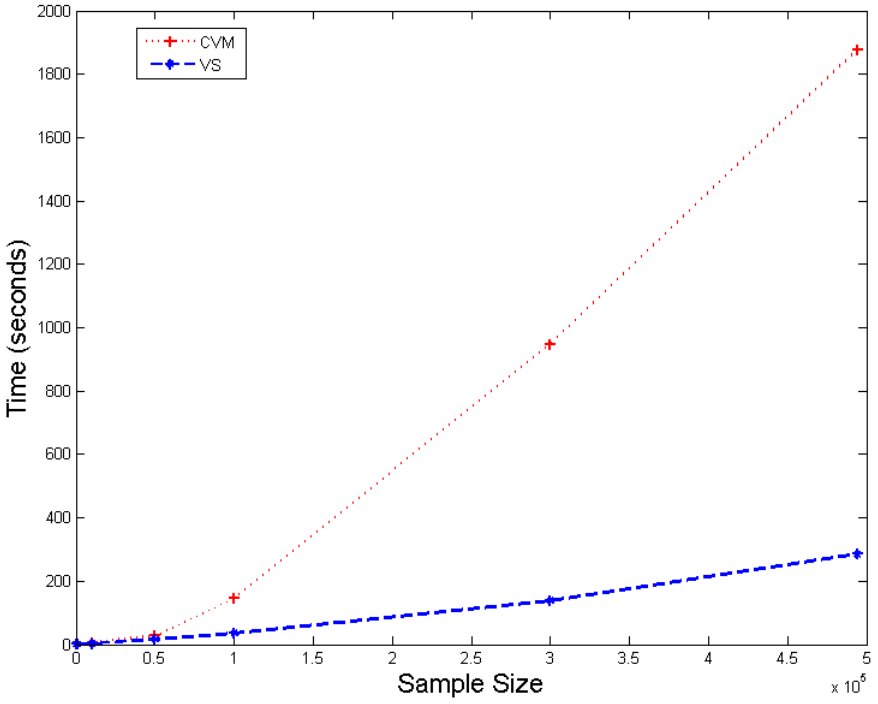


FIGURE 4.18: Time required to train and test on subsets of the KDD '99 data for the CVM and VS. The testing set was the same size as the training set.

[7]available from http://www.autonlab.org/autonweb/10522, last accessed 21/03/2010

Algorithm	Optimal parameters	Prediction Acc.(%)	AUC	Time (seconds)
NN	$k = 9$	92.10	0.9514	6122
CVM	$\sigma = 0.5$,$C = 10^6$	93.01	0.9786	2136.22
MEB	$\sigma = 1$,$C = 10^4$	92.33	0.9621	3759.8
Voted Perc (VP)	$T = 1$	91.89	0.9015	266.19
KVP*	$\sigma = 1$,$T = 1$	92.90	0.9507	4060
VP w/ Memory*	$T = 1$	92.72	0.9618	312.10
OCBudget	ker=RBF, $\sigma = 0.5$	92.45	N/A	1525.5
ETBudget*	ker=RBF,$\sigma = 0.5$	92.13	N/A	321.22
OCBudget*	ker=RBF,$\sigma = 0.5$	92.33	N/A	20.76
CBSVM[1]	B=30 T1=6.16 T2=6.295	78.75	N/A	220
KBHC-CBSVM[2]	ker=RBF B=30 T1=5.2×10^{-5} T2=6.2×10^{-4} σ=0.004 Tol=0.0001	91.78	N/A	373
SVM[3]	C = 1 ker=RBF	92.07	N/A	5172
VS	maxIDis=2.6 maxNDis=0.6	94.70	0.9815	110.11
VS Distance	maxIDis=1.7 maxNDis=0.6	94.81	**0.9857**	**100.23**
VSCluster	maxIDis=2.3 maxNDis=0.6	94.55	0.9827	409.25
VSMulti	maxIDis=0.7 maxNDis=0.3	93.55	N/A	301.77
VSKernel(RBF)	maxIDis=1.4 maxNDis=0.6 $\sigma = 0.5$	94.40	0.9559	379.22
VSBinaryDistance(Ham)	maxIDis=1 maxNDis=4	91.93	0.8611	132.39
VSBinary(Ham)	maxIDis=1 maxNDis=6	92.18	0.8870	103.72
VSBinary(Tan)	maxIDis=0.7 maxNDis=0.2	**94.96**	0.9619	186.19

TABLE 4.3: Results obtained by training on the two-class KDD '99 training data, and testing on the corresponding test set. The best values are in bold font. * indicates algorithms that were run on a random 10,000 subset and not the 10% KDD training set. [1]Result reported in Asharaf et al. (2006), which was trained on 5049×38 points only. [2]Result reported in Asharaf et al. (2006),which was trained on 4961×38 points only. [3] Reported in Asharaf et al. (2006) using LIBSVM (Chang and Lin (2001)) version 2.8. The VS achieves results competitive with the state-of-the-art in a fraction of the time.

sample size increases, the VS has a steady linear increase in the time required. There is no issue with the increase in hyperspheres, since we show in Section 4.2.2 that the number of hyperspheres created grows sublinearly with the sample size. On the other hand, the CVM appears to have a linear increase in the time required as well, albeit with a much higher slope than the VS. On the 10% KDD subset, it achieves the second highest prediction accuracy after VS (and all its modifications). An issue is that for some values of the hyperparameters (C and σ for the RBF kernel), the algorithm never converges, which was also discussed by Canu and Loosli (2006). The same can be said about the MEB, which fails to converge when incorrect hyperparameters are chosen. The MEB achieves a competitive AUC compared to the CVM, and a slightly lower prediction accuracy, despite not making use of any intrusion data.

The Voted Perceptron (VP) achieved a 91.89% prediction accuracy (shown in Table 4.3). Figure 4.20 gives a plot comparing the ROCs of the VP and the CVM, where the latter achieved a better curve and higher AUC. Despite having a lower AUC and prediction rate than the CVM and the NN, the VP is an online algorithm that we can force to be one-pass, and does not perform any subsampling or make use of any optimisation routines. Intuitively, modifying the VP to include memory (described in Section 2.3) increased the performance. This is because hard to classify points could be presented to the algorithm early on during training, causing an otherwise good separating hyperplane to be put aside. Incorporating a non-linear kernel produces the best results for the VP. This comes at a significantly higher computational cost. One run of the kernelised VP took 4,060 seconds to run on a standard desktop PC, and is not applicable in practice on very large data sets.

We then tested the OCBudget, which was discussed in detail in Section 2.3. The hyperparameter β was set to 0.01, which was the arbitrary value used by Crammer et al. (2003) on all the data sets in their paper, and produced good results. The algorithm required 1525.5 seconds to run, and returned 513 support vectors. A RBF kernel was used with $\sigma = 0.5$, which is the same value used for the CVM and the VSKernel. We used the Spider toolbox's[8] implementation, and the results are given in Table 4.3. The OCBudget gave a higher prediction accuracy than the VP, but was outperformed by both the kernelised VP, and the memory-modified VP.

The ETBudget[9] required a lot more memory than the OCBudget algorithm, causing MATLAB to run out of memory on the 10% KDD subset. Instead, we used a random subsample of 10,000 points. Since we need to set the maximum number of support vectors, we ran the OCBudget on the 10,000 points (using $\beta = 0.0$, the arbitrary value used by Weston et al. (2005) on several data sets), which returns 116 SVs. Since we want to compare the ETBudget and OCBudget together, the absolute hyperparameter values are not crucial, as long as they are the same for both algorithms. We set the maximum

[8] Available from http://www.kyb.tuebingen.mpg.de/bs/people/spider/main.html
[9] Available from http://www.kyb.mpg.de/bs/people/weston/budget/

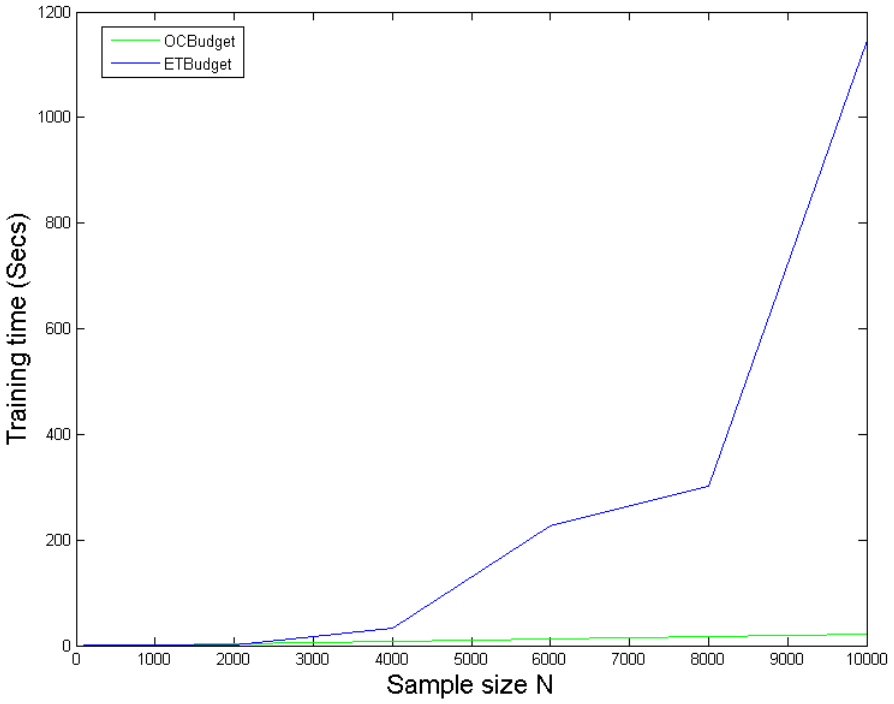


FIGURE 4.19: Training time versus N for the ETBudget and OCBudget on subsets of the KDD '99 data.

number of SVs to 116 for the ETBudget (with $\beta = 0.0$ to obtain a fair comparison), and use the same RBF kernel with $\sigma = 0.5$ as the OCBudget, again for a fair comparison. The ETBudget required 321.22 seconds and gave a 92.13% prediction accuracy, whereas the OCBudget achieved 92.33% accuracy in 20.76 seconds. These are given in Table 4.3. As discussed in Section 2.3, the update rule of the ETBudget is significantly more expensive to compute, and requires all previously seen examples to be stored for the calculation of the error rate. This means that once a point has been seen, it cannot be discarded as we do in the VS case. A plot of training time versus sample size is given in Figure 4.19. This clearly shows the difference in computational complexity between the two algorithms; the OCbudget's training scales linearly with N, and can be scaled to the 10% KDD subset, which we were not able to do for the ETBudget.

We then tested our VS algorithm along with the modifications to see how they compared to the other algorithms that explicitly take into account the large training set. The VS is very fast (takes under two minutes to train and test on $494,020$ and $311,029$ points, respectively), and achieves the highest prediction performance among all the methods tested. Given the difficulty of conducting significance tests with published results for the KDD data, we claim that the VS achieves state-of-the-art performance in a small fraction required by other methods. Also, each of the modifications tested from Section 4.4 achieved prediction accuracies higher than Pfahringer (2000), who required over a

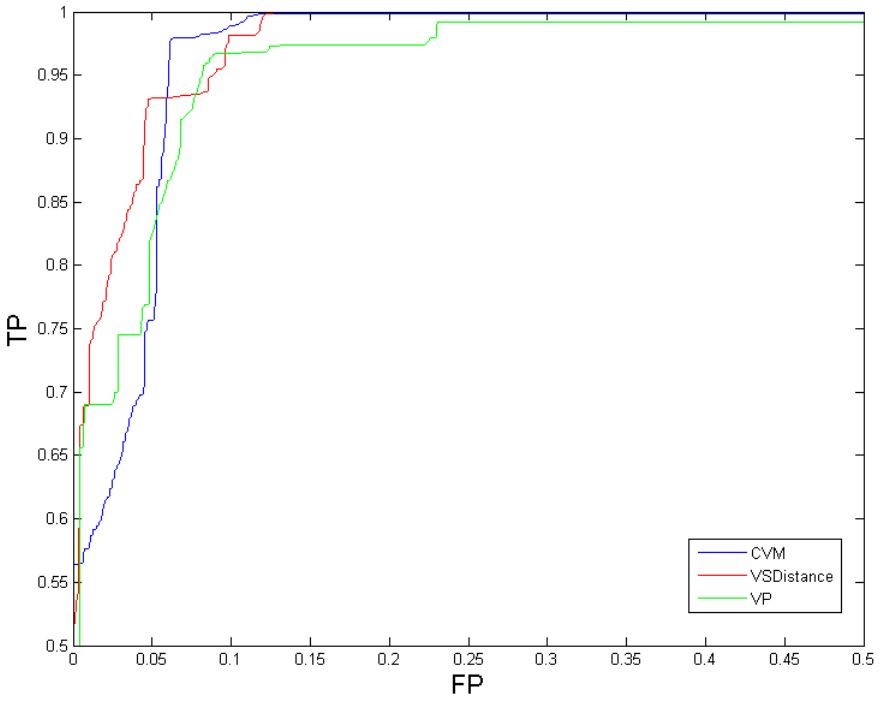


FIGURE 4.20: ROC curves for the VSDistance, CVM, and VP on the 10% KDD '99 data, where the AUCs are 0.9857, 0.9786, and 0.9730 respectively.

day to train his model whereas the VS techniques needed a few minutes. ROC curves for the VSDistance, CVM, and VP are given in Figure 4.20, where the AUCs are 0.9857, 0.9786, and 0.9730 respectively. VSDistance's ROC curve is clearly the best, as it is closest to the top left corner.

We also included in Table 4.3 published results from the literature on the two-class KDD '99 data. This comparison was possible since the same 10% subset was used for training in our empirical study, and the same test set was used for testing. The algorithms were the Clustering-Based SVM (CB-SVM), its kernelised counterpart (KBHC-CBSVM), and the SVM. Note that the time comparison is not fair, since the experiments were run on different machines, and no information was given on implementation. These algorithms were run on the 38 numerical features, which were normalised to one by dividing each feature by its maximum value. The CB-SVM is the weakest performing since it is a linear classifier, and the published result in Asharaf et al. (2006) resembles that of a random classifier since the test set is roughly 80% intrusions and 20% normals. Verifying this is made harder due to the fact that no AUC was published by the authors. The KBHC-CBSVM with an RBF kernel significantly improved on its linear counterpart, which is expected as the KDD '99 data is not linearly separable. From the optimal parameters reported in Asharaf et al. (2006), shown in Table 4.3, we can see the real benefit of comparing our method with published results as opposed to our own implementations

only: the hyperparameters have been adjusted to the authors' satisfaction. Despite this, the KBHC-CBSVM still achieves a lower performance than the VS and its modifications, and requires more time. Also reported by Asharaf et al. (2006) are results using an SVM with an RBF kernel. This achieved a 92.07% prediction accuracy, which is slightly higher than the KBHC-CBSVM, and lower than VS. The time required by the SVM was also high, despite using a fast C++ implementation called LIBSVM by Chang and Lin (2001).

Algorithm	Classification Acc.(%)	AUC
VS	$\mathbf{94.22} \pm 0.69$	0.9688 ± 0.00
VSDistance	94.17 ± 0.48	$\mathbf{0.9853} \pm 0.00$
VP	89.86 ± 1.41	0.9626 ± 0.04
MEB	92.58 ± 0.68	0.9628 ± 0.04
CVM	92.86 ± 0.82	0.9623 ± 4.8

TABLE 4.4: Results on five different 10% subsets drawn from the entire KDD '99 data set

Furthermore, we ran experiments using VS, VSDistance, VP, MEB and CVM on five different random 10% subsets obtained from the full KDD '99 Cup data (containing 4,898,424 points). This is, to the best of our knowledge, the only set of such results on the KDD '99 Cup data, as authors generally use the publicly available 10% subset for training, and the results published are similar to the way Table 4.3 is presented. The results of these experiments are given in Table 4.4. For the VS and VSDistance, the hyperparameters obtained on the aforementioned 10% subset were used, and the results show the stability of the VS with respect to its hyperparameters. It also indicates that each 10% subset was representative of the data distribution, and that VS was correctly capturing the distribution of the data in input space. Again, the VS and VSDistance outperformed the other methods tested, both in terms of the generalisation performance and the AUC. Also, note the high standard deviation of the CVM's AUC, which can be attributed to the fact that the random subsampling was generating different results for each run.

4.7.2 On the Multi-class 10% Subset

We also compare the multi-class Voted Spheres with some of the most recent algorithms in the literature that have used the multi-class KDD '99 data, including the competition winner. This is shown in Table 4.5. The multi-class VS is among the best performing methods, with very little parameter tuning. It also has the advantage of being one-pass, making it much less computationally expensive than the remaining multi-pass algorithms in the table. Some techniques have an unfair advantage, such as the ADITS and the ADAT, which are algorithms that need to take feedback from the system operator when

false predictions are identified. This is impractical, since it requires the system operator to be present on his machine during the entire training process, as both algorithms pass suspicious predictions to him to be verified. Another drawback is that these methods assume that such a person exists with the domain knowledge required to judge these connections, making it difficult for the majority of researchers to implement and apply them.

Algorithm	Accuracy(%)
VSMulti	93.55
1-NN (Elkan (2000))	92.33
KDD Winner (Pfahringer (2000))	92.71
KDD Runner-up(Levin (2000))	92.92
MC-SLIPPER (Yu and Tsai (2004))	92.59
ADITS (Yu et al. (2007))	**94.49**
ADAT (Yu et al. (2008))	**94.37**
Genetic clustering (Liu et al. (2004))	79
Hierarchical SOM (Sarasamma et al. (2005))	90.04-93.46
RSS-DSS (Song et al. (2005))	89.2-94.4
Adaboost-Based (Hu et al. (2008))	90.04-90.88

TABLE 4.5: A comparison of VS with published results on the multi-class KDD '99 data

We give the confusion matrix that VSMulti achieved on the KDD '99 data in Table 4.6, and the confusion matrix achieved by the competition's winner in Table 4.7. This

Predicted → Actual ↓	Normal	Probe	DoS	U2R	R2L
Normal	58600	201	1677	24	91
Probe	204	2868	1080	0	14
DoS	1105	135	228072	44	497
U2R	15	137	42	26	8
R2L	10612	10	4042	126	1399

TABLE 4.6: Confusion matrix generated by VSMulti on the KDD '99 data

Predicted → Actual ↓	Normal	Probe	DoS	U2R	R2L
Normal	60262	243	78	4	6
Probe	511	3471	184	0	0
DoS	5299	1328	223226	0	0
U2R	168	20	0	30	10
R2L	14527	294	0	8	1360

TABLE 4.7: Confusion matrix generated by Pfahringer (2000) on the KDD '99 data

helps show where our algorithm outperformed Pfahringer (2000), and where the latter

algorithm outperformed VS. For instance in Table 4.8 we see that VSMulti achieved lower PDs and higher FARs than Pfahringer (2000) and the runner up on the majority of the categories. Despite this, VSMulti achieved a higher overall classification rate, and a lower average cost per example (which was calculated using Table 1.4). From

		Probe	DoS	U2R	R2L	Avg. cost per example
Pfahringer (2000)	PD	0.833	0.971	**0.132**	0.084	0.2331
	FAR	0.006	**0.003**	**0.00003**	**0.00005**	
Levin (2000)	PD	**0.845**	0.975	0.118	0.0732	0.2356
	FAR	0.216	0.731	0.364	0.017	
VSMulti	PD	0.6884	**0.9923**	0.1140	**0.0864**	**0.1956**
	FAR	**0.0016**	0.0843	0.00062	0.0021	

TABLE 4.8: VSMulti compared with the top results on the KDD '99 Cup data, using PD and FAR, and average cost-per-example

the confusion matrices, we see that Pfahringer (2000) misclassified 3,915 more R2Ls as normal connections than did VSMulti, which is the misclassification that incurs the highest cost. Also, despite the fact that VSMulti had lower PDs than the competition's winner in two out of the four attack categories, the number of DoS points correctly classified by the former technique exceeds the latter by 4,846 points. Also from the confusion matrices, we are able to judge where the VS could be improved. For instance to further improve results, we can use a different radius for each of the five different categories of connections to avoid costly misclassifications.

4.7.3 On the Entire $\sim 5 \times 10^6$ KDD '99 Data

In order to show the scalability of our algorithm, we provide details of the performance of VS (C++ implementation) when run on the full KDD '99 training data. We used the hyperparameter values that gave the best results on the 10% subset, and on the entire KDD '99 data (comprised of $4,898,424$ training points and $311,029$ test points) the VS achieved an accuracy of 93.67%, which is comparable to the best results for this task. An interesting point to note was that 672 hyperspheres were created during training for the 10% data, while only $1,451$ were created for the entire $4,898,424$ data set (training and testing in just over five minutes). This shows that the hyperspheres created for the 10% subset were representative of the distribution of the data in input space. It further proves that the number of hyperspheres created does not grow linearly with N; if that were the case, we would have hyperspheres of the order of 7000. Note that choosing parameters on the subset and not the whole data indicates the stability of parameter selection. Results are given in Table 4.9, comparing the vanilla VS with the three algorithms in the literature that were applied on the entire KDD '99 data, and not just the 10% subset. The ASVM in Table 4.9 is an SVM with selective sampling or active learning. In regard to what was said earlier about authors tweaking hyperparameters, we

see contradictory results for the same technique reported by different authors. Yu et al. (2003) report that they achieved 93.27% accuracy on the entire KDD '99 Cup data, while Asharaf et al. (2006) report that Yu et al. (2003)'s CB-SVM achieved 78.75% accuracy on the 10% subset. Nevertheless, VS outperforms both of these results, in a fraction of the time.

Algorithm	Accuracy(%)	Time (secs)
VS	**93.67**	**310.9**
CB-SVM*	93.27	4753.1
ASVM*	93.04	94192.2
CVM	85.43	2192.4

TABLE 4.9: Results on the entire KDD '99 data. * Results reported by Yu et al. (2003).

4.7.4 Binarising the 10% KDD '99 Data

Some features of the KDD '99 Cup were continuous and some were symbolic. Of the 41 features of the data, three of them were symbolic, with values reaching up to 67 for the third feature. When calculating distances in Euclidean space, this difference between the symbolic features and the continuous ones (with a maximum value of 1) could potentially give less accurate results. This is because these symbolic features could dominate in the distance calculation. Thus far, the VS has outperformed the published results on the KDD '99 data, so this combination of features are obviously not significantly harmful to our classification. However, we binarised the data to discover whether an improvement in results was achievable. We obtain the maximum value of each feature (for both training and testing sets), and based on that value we know how many bits will be required to represent it in binary form. These values are given in Table 4.10 for all the features. For example for feature 2, since it can only take values in $\{0, 1, 2\}$, we replace each value with its binary representation as follows:

- $0 \rightarrow 0\ 0$
- $1 \rightarrow 0\ 1$
- $2 \rightarrow 1\ 0$

We do the same for features 3 and 4, and are now left with a 51 feature binary data set. For the remaining continuous features, the values were set to 0 if they were less than or equal to the feature mean, and 1 otherwise. We apply two different measures. The first is the Tanimoto similarity coefficient (Tanimoto (1958)), which is the most widely used similarity coefficient in the chemoinformatics field, where the data are represented as long binary strings. Trotter (2006) used this metric in a kernel classifier framework for drug discovery, and has demonstrated superior results compared to others, which was

Feature	Max value	Number of bits required
1	1	1
2	2	2
3	67	7
4	10	4
5→41	1	1

TABLE 4.10: Number of bits required to express the KDD '99 features in a binary fashion. Note the continuous features have been already normalised to $[0, 1]$ following Yu et al. (2003); Tsang et al. (2005)

also shown by Tuna and Niranjan (2010) and Tuna and Niranjan (2009). In this section, we use the Tanimoto similarity metric to define hyperspheres around our binarised data. To define this similarity, assume we have two binary vectors $\mathbf{x}$ and $\mathbf{x}'$, and that a and b are the number of 1s in $\mathbf{x}$ and $\mathbf{x}'$ respectively, and that c is the number of corresponding 1s in both $\mathbf{x}$ and $\mathbf{x}'$ (i.e. 1s in the same features), then the Tanimoto similarity is defined as:

$$T = \frac{c}{a + b - c}.$$

The range of this coefficient is between 0 and 1 (inclusive), with the former indicating no similarity whatsoever, and the latter indicating that the two vectors are identical. It represents the ration of the common number of bits set to one to the total number of bits set to one for both vectors. The denominator serves as a normalisation factor to reduce the effect of the vectors' sizes. We used the VS (using cross-validation for the radius as in Section 4.2.1) with the Tanimoto similarity instead of a Euclidean measure (using $1 - T$ as the distance), and obtained an accuracy of 94.96% (see Table 4.3). This result, although better than the one achieved by VSDistance, is not significantly higher. Because of this, we can argue that the mix of symbolic and continuous features did not significantly degrade the performance of our classifiers, and little benefit is achieved by binarising the data and using the Tanimoto similarity.

The other measure we used for our binary data is the widely-used Hamming distance. The Hamming distance between two binary vectors is the number of positions for which the corresponding bits are different. The range of this distance measure is between 0 and d (the length of the vector), where the former implies identical vectors, and the latter means they are completely different. Using the VS with the binary KDD '99 data, we applied cross validation to obtain the optimal radii. The prediction rate was 92.18% which is lower than the regular VS yet still competitive with the other techniques in Table 4.3.

These results indicate that the performance of the VS did not degrade with the mix of symbolic and continuous features, and that the Tanimoto similarity measure outperformed the Hamming distance on our intrusion detection task.

4.8 Data-dependent bounds

One characteristic of the VS is that the number of hyperspheres produced is often a small fraction of the data set (0.030% of the KDD's $4,898,424$ points were hyperspheres), as discussed in Section 4.2.2. This allows us to bound the generalisation error of the classifier using risk bounds based on data-dependent compression schemes proposed by Laviolette et al. (2005). This means that by using the training set only, we can theoretically guarantee the worst case prediction error on the unseen test set before we run our algorithm.

Laviolette et al. (2005) describe an algorithm A as a sample-compression learning algorithm if it takes as an input a training set $S = \{\mathbf{z}_1, ..., \mathbf{z}_N\}$ of size N, and returns a classifier $A(S)$ that is described entirely by two complementary sources of information: a subset $\mathbf{z_i}$ of S (the compression set), and a message string σ which represents the extra information needed to obtain a classifier from the compression set. Given a training set S, the compression set $\mathbf{z_i}$ is defined by a vector $\mathbf{i}$ of indices $\mathbf{i} \stackrel{\text{def}}{=} (i_1, i_2, \cdots, i_{|\mathbf{i}|})$ with $i_j \in \{1, \cdots, N\} \, \forall j$ and $i_1 < i_2 < \cdots < i_{|\mathbf{i}|}$ where $|\mathbf{i}|$ indicates the number of indices in the set $\mathbf{i}$. Also, $\bar{\mathbf{i}}$ denotes the set of indices not present in $\mathbf{i}$, making $S = \mathbf{z_i} \cup \mathbf{z}_{\bar{\mathbf{i}}}$ for any vector $\mathbf{i} \in \mathcal{I}$ where $\mathcal{I}$ denotes the set of 2^N possible realisations of $\mathbf{i}$.

In order for the classifier $A(S)$ to be described solely by a compression set and a message string, there needs to be a reconstruction function $\mathcal{R}$ associated with A that returns a classifier $\mathcal{R}(\sigma, \mathbf{z_i})$ when given an arbitrary compression set $\mathbf{z_i} \subseteq S$, and a message string σ from the set $M(\mathbf{z_i})$ of all distinct messages that can be supplied to $\mathcal{R}$ with the compression set $\mathbf{z_i}$. Assuming each example $\mathbf{z}$ is drawn according to a fixed but unknown distribution D on $\mathcal{X} \times \mathcal{Y}$, the risk $R(f)$ of any classifier f is defined as the probability that it misclassifies an example drawn according to D:

$$R(f) \stackrel{\text{def}}{=} \Pr_{(\mathbf{x},y)\sim D}\left(f(\mathbf{x}) \neq y\right) = \mathbf{E}_{(\mathbf{x},y)\sim D} I\left(f(\mathbf{x}) \neq y\right)$$

where $I(a) = 1$ if the predicate a is true, and 0 otherwise. Given a training set $S = \{\mathbf{z}_1, \cdots, \mathbf{z}_N\}$ of N examples, the empirical risk $R_S(f)$ on S of any classifier f is defined as:

$$R_S(f) \stackrel{\text{def}}{=} \frac{1}{N} \sum_{i=1}^{N} I\left(f(\mathbf{x}_i) \neq y_i\right) \stackrel{\text{def}}{=} \mathbf{E}_{(\mathbf{x},y)\sim S} I\left(f(\mathbf{x}) \neq y\right)$$

Let $\mathbf{Z}^N$ denote the collection of N random variables whose instantiation gives a training sample $S = \mathbf{z}^N = \{\mathbf{z}_1, \cdots, \mathbf{z}_N\}$. Denote $\Pr_{\mathbf{Z}^N \sim D^N}(.)$ by $\mathbf{P}_{\mathbf{Z}^N}(.)$. Then, for any compression set-dependent distribution of messages $P_{\mathcal{M}_{(\mathbf{z}_i)}}$ satisfying:

$$\sum_{\sigma \in \mathcal{M}_{(\mathbf{z}_i)}} P_{\mathcal{M}_{(\mathbf{z}_i)}}(\sigma) \leq 1 \; \forall \mathbf{z}_i$$

and any prior distribution $P_{\mathcal{I}}$ of vectors of indices satisfying:

$$\sum_{\mathbf{i}\in\mathcal{I}} P_{\mathcal{I}}(\mathbf{i}) \leq 1$$

we get the risk bound:

Theorem 4.1 (Quoted from Laviolette et al. (2005))**.** *For any reconstruction function $\mathcal{R}$ that maps arbitrary subsets of a training set and message strings to classifiers, for any prior distribution $P_{\mathcal{I}}$ of vectors of indices, for any compression set-dependent distribution of messages $P_{\mathcal{M}(\mathbf{z_i})}$, and for any $\delta \in (0, 1]$, we have:*

$$\boldsymbol{P}_{\boldsymbol{Z}^N}\Big\{\forall_{\mathbf{i}} \in \mathcal{I}, \forall_{\sigma} \in \mathcal{M}(\boldsymbol{Z}_{\mathbf{i}}) : R(\mathcal{R}(\sigma, \boldsymbol{Z}_{\mathbf{i}})) \leq \frac{1}{A}\left[\ln\binom{N-d}{k} + \ln\left(\frac{1}{P_{\mathcal{I}}(\mathbf{i})P_{\mathcal{M}(\mathbf{z_i})}(\sigma)\delta}\right)\right]\Big\} \geq 1-\delta$$

where $d = |\mathbf{i}|$ is the sample compression set size of classifier $\mathcal{R}(\sigma, \boldsymbol{Z}_{\mathbf{i}})$, $k = |\bar{\mathbf{i}}|R_{\mathbf{z}_{\bar{\mathbf{i}}}}(\mathcal{R}(\sigma, \boldsymbol{Z}_{\mathbf{i}}))$ is the number of training errors that this classifier makes on the examples that are not in the compression set and $A = N - d - k$.

The risk bound of a classifier is quite small when both the set size and the number of training errors are small. For the VS algorithm, we see that the set of hypersphere centres forms a compression set, and the message string in this case is the weight for each sphere. From this information, one can easily reconstruct the decision rule of the algorithm. It is also clear that the above bound depends on the prior distributions chosen over the possible message strings $P_{\mathcal{M}_{(\mathbf{z}_i)}}$ and indices $P_{\mathcal{I}}$. Unfortunately, there is no systematic way to set these, and one can naively set the distributions over the indices and the possible messages (which are integer weights) to be uniform distributions. Hence we pay a log penalty for each integer weight. One may be able to obtain a tighter bound however, as indices are more likely to be from earlier data points (see Section 4.2.2 for a discussion), and messages are not uniform (for a data set of size N the maximum weight of a centre at time t is $N - t$).

As an example, from a run of VS on the 10% KDD subset, we obtain:

- $N = 494,020$ - Training data size.
- $d = 105$ - Number of hyperspheres created
- $k = 25,822$ - Number of errors on non-hypersphere training points.
- $P_{\mathcal{M}(\mathbf{z_i})} = \frac{1}{N^d} = \frac{1}{494020^{105}}$ - Distribution of messages; since there are N^d possible combinations of messages and we assume they are all equally likely.

- $P_{\mathcal{I}}(\mathbf{i}) = \binom{N}{|\mathbf{i}|}^{-1}(N+1)^{-1} = \binom{494020}{105}^{-1}(494021)^{-1}$ - Probability distribution of vectors of indices. Each of the $N+1$ possible values of $|\mathbf{i}|$ is equally likely, hence we divide by $\binom{N}{|\mathbf{i}|}$.

Therefore we obtain

$$\boldsymbol{P}_{\boldsymbol{Z}^N}\Big\{\forall_{\mathbf{i}} \in \mathcal{I}, \forall_{\sigma} \in \mathcal{M}(\boldsymbol{Z}_{\mathbf{i}}) : R(\mathcal{R}(\sigma, \boldsymbol{Z}_{\mathbf{i}})) \leq 0.2216\Big\} \geq 0.9$$

which means that if the training and testing data were i.i.d., then with at least 90% confidence the generalisation error is $\leq 22.16\%$.

4.9 Conclusion

In this chapter, we present a fast, novel, non-linear, multi-class classifier called Voted Spheres. The algorithm is online, requiring one pass through the data, enabling it to scale to very large problems. For the full KDD '99 Cup data with close to 5 million points, VS took five minutes on average to both train and test while achieving state-of-the-art performance. Apart from its very low computational cost, we have demonstrated other important advantages of VS:

- It is very stable with respect to its hyperparameter.
- The order of data input does not significantly affect the performance of the VS, highlighting its stability.
- There is no need to re-train from scratch upon the arrival of new training data; each new training point is inserted at $O(m)$ cost.
- The number of hyperspheres grows sublinearly with the number of training points, further allowing the VS to scale to very large data.
- Achieves state-of-the-art performance.
- Flexible framework, with many possible modifications.

We then identified and discussed the similarities and differences of the VS to two algorithms, the edited k-NN and the RAN. The former creates a reference set by eliminating points from the training set, whereas in VS, the eliminated points are important for correct classification and density preservation. We discuss this thoroughly in Section 4.3 using the Peterson-Barney vowel data set. The RAN, despite its clear differences to VS, has a similar fundamental idea: to allocate a new unit whenever a training point deviates enough from the current state of the algorithm.

We then present seven modifications to VS in Section 4.4, each of which solves a different problem, and conduct a thorough empirical study of VS compared with other state-of-the-art methods in the literature. We use Gunnar Rätsch's Benchmark data sets in Section 4.5, which are widely used in the literature for benchmarking purposes, along with ten data sets from the UCI repository and demonstrate that VS performed competitively with the best published results on these data sets. We demonstrate the scalability of the VS algorithm by applying it on the 10% KDD subset, and compare it with the state-of-the-art methods using both our implementations and published results. The VS achieved results comparable to the published state-of-the-art results on both the binary and multi-class KDD '99 data, while making significant memory and computational savings. This is advantageous since the published results eliminate the possibility of any criticism regarding incorrect implementation or insufficient parameter tuning. To further show scalability, we compared the performance of VS on the entire KDD '99 Cup data, and not just the 10% subset, with three methods in the literature that were run on the entire KDD data. The VS outperformed the other techniques in a fraction of the time. Finally, given that the VS can be regarded as a compression scheme (as in Section 4.8), we showed how its generalisation error on unseen test data can be bounded using risk bounds.

Chapter 5

Adapting the VS Framework for Distribution Shift

Usually, systems that are built using machine learning techniques face difficulties when applied in real life. The conditions under which the systems were developed will almost certainly be different than those in which they will be used. This distribution shift is generally never taken into account in machine learning algorithms. They assume that the training and test data are drawn from the same distribution, or that it does not matter if they are not. In this chapter, we investigate the effect of a distribution shift on both a parametric model and a non-parametric one, and show that the existence of a shift in the data causes the classifiers' performances to degrade. To correct for this, we implement and incorporate the two most recent techniques from the literature into our classifiers: an importance weighting scheme by Kanamori and Shimodaira (2009) to downweight large portions of the training distribution with less importance in the test data, and kernel mean matching (KMM) by Gretton et al. (2009), which matches the input distributions in a high dimensional feature space. The authors demonstrate these on parametric models, while we extend their use to our non-parametric VS. We compare the two weighting methods with respect to performance ability, speed, and how well they adapt to large data sets. Our results on artificial data and subsets of the KDD '99 data show that explicitly accounting for distribution shift during training increases the generalisation ability of both classes of algorithms.

5.1 Introduction

There are plenty of examples of domains that contain a distribution shift, such as email spam detectors and computer intrusion detection systems. A sophisticated IDS built today will most likely not work in a few years time and will need to be adapted. The reason for this is that data changes over time; new intrusions emerge, normal data

changes, and intruders in general constantly look for ways to bypass IDSs. This problem can be seen as a case of data set shift, where the joint distribution of inputs and outputs differs between the training and testing stage. Textbook predictive machine learning models, however, work by ignoring these differences. They assume that the test and training data distributions match, or that it makes no difference if they do not. This problem has only recently gained popularity, especially after the NIPS 2006 workshop entitled "Learning when test and training inputs have different distributions," with a book based on the workshop published by Quionero-Candela et al. (2009).

As mentioned throughout this thesis, the KDD '99 data set has been widely used for testing and evaluating algorithms for intrusion detection purposes, and for large scale tasks in general. In this chapter, we will take advantage of the fact that it contains a distribution shift between the training and test phases. The shift occurs because the training and test data were collected at separate time intervals, and because new attack data types were added to the test set that were not present in the training set. The existence of a shift was verified using a Kolmogorov-Smirnov statistical test in MATLAB (see section 5.1.1). A very broad range of machine learning techniques were applied on this data set, with methods ranging from supervised rule generation to anomaly detection using normal data only. SVMs have been used by Fugate and Gattiker (2002) for anomaly detection, by Kim and Park (2003) and Lee et al. (2002) to construct host and network-based IDS, and by S. Mukkamala (2002) for network-based IDS, which was shown to outperform neural networks. Multi-layer Perceptrons (Sabhani and Serpen (2003)) were also used on the KDD '99 Cup data, along with different types of clustering (Ertoz et al. (2003)) techniques for unsupervised learning. Other techniques applied on the KDD '99 data were decision trees (Levin (2000)), Parzen windows (Parzen (1962)) and boosting among others. The latter technique (boosting) was used in conjunction with decision trees by Pfahringer (2000) to win the KDD '99 competition, achieving a 92.71% classification rate, as discussed in Section 1.1.3. A common issue when we take a closer look at the methods applied on this data, is that none of them explicitly account for this distribution shift, despite being fully aware of its existence.

Using this data set, we aim to explore whether accounting for distribution shift improves generalisation results for both parametric and non-parametric methods. The non-parametric method we use is our one-pass Voted Spheres, while the parametric model we use is logistic regression (LR) trained using gradient descent (see Section 5.3.2). In Section 5.2, we describe several ways the data set shift problem is tackled in the literature. Specifically, the shifting Perceptron algorithm (SPA) by Cavallanti et al. (2006) is discussed. Also, a technique by Kanamori and Shimodaira (2009) is described which is used to downweight large parts of the input space with less importance in the test distribution. The final method we discuss for dealing with distribution shift is kernel mean matching, which works by matching the input distributions of the training and test data in a high dimensional feature space. Then, in Section 5.3, we incorporate both

weighting schemes into the VS and the LR, and perform experiments on toy data and subsets of the KDD '99 data to show that a distribution shift in the data puts classifiers that assume the data are i.i.d. at a disadvantage. Explicitly accounting for this shift corrects the problem in both the parametric and non-parametric class of algorithms. We conclude the chapter with a discussion of the results and the conclusions drawn.

5.1.1 Kolmogorov-Smirnov test on the KDD '99 Cup Data

In order to verify the existence of a distribution shift between the training and test sets for the KDD '99 Cup data, we used the non-parametric Kolmogorov-Smirnov (K-S) test for two independent samples. The aim of this test is to check whether the two independent samples were drawn from the same population. The Kolmogorov-Smirnov statistic quantifies a distance between the empirical distribution functions of two samples. The null hypothesis is that the two samples are drawn from the same underlying distribution, and the alternative hypothesis is that they are drawn from different underlying distributions. The empirical distribution function F_N for N i.i.d. observations x_i is defined as:

$$F_N(x) = \frac{1}{N} \sum_{i=1}^{N} I_{x_i \leq x}$$

where $I_{x_i \leq x}$ is the indicator function equal to 1 if $x_i \leq x$ and 0 otherwise. The Kolmogorov-Smirnov statistic to test for the independence of two one-dimension probability distributions is:

$$D_{N,N'} = \sup_x |F_{1,N}(x) - F_{2,N'}(x)|,$$

where $F_{1,N}(x)$ and $F_{2,N'}(x)$ are the empirical distribution functions of the first and second samples respectively, and $\sup S$ is the supremum of set S. The supremum of a set S of real numbers is defined to be the smallest real number that is greater than or equal to every number in S. The null hypothesis is rejected at the α level if:

$$\sqrt{\frac{NN'}{N+N'}} D_{N,N'} > K_\alpha$$

where K_α is the critical value of the test.

To the best of our knowledge, this is the first use of a formal statistical test to verify the KDD '99 data's distribution shift. For this, we use MATLAB's built-in *kstest2* function. The null hypothesis, as mentioned above, is that the two data samples are from the same distribution, and the alternative hypothesis is that they are drawn from different distributions. The syntax is:
h = kstest2(**x1**,**x2**),
where **x1** and **x2** are the two data vectors, and the value returned in h is 0 if the null hypothesis is accepted and 1 otherwise (at the 5% significance level).

Since our data is multi-dimensional, and the *kstest*2 above takes two vectors as input, we cannot simply pass our 41-dimensional data to it, and the need for a different approach arises. When we look closely, we realise the 'shift' in such data means that the distribution of a feature i is different between the training and testing data. Therefore, what we should test for is whether corresponding features from the training and testing distributions are drawn from the same underlying distribution. If this is the case for all the features, then we conclude that the training and testing data follow the same distribution, otherwise, the data contains a shift. Also, the K-S test requires the input data to be either continuous or ordered nominal data. Since nine of the KDD '99 features (see Section 1.1) are either binary or symbolic (and not ordered), we remove them before performing the K-S tests. We are left with 32 continuous features.

To verify our approach, we check whether it correctly identifies that two samples from the training data (with no repetitions) follow the same distribution. We randomly shuffle the 10% KDD '99 Cup training sample to remove any bias, and split the data in half. We now have two samples, *sample*1 and *sample*2, with $247,010$ points and 32 dimensions each. We used the Kolmogorov-Smirnov technique to test feature by feature, comparing each feature i of *sample*1 with the corresponding feature in *sample*2. This means that we pass two vectors of size $247,010$ to *kstest*2 32 times. Out of the 32 tests conducted between *sample*1 and *sample*2, none of them followed different distributions as *kstest*2 returned a value of 0 for all tests, which is correct since they are both drawn from the training distribution. However, between *sample*1 and *test* (a random subsample of size $247,010$ from the test set), 22 out of the 32 features were flagged as belonging to different distributions. We conclude by these tests that the KDD '99 training and test sets do indeed follow different underlying distributions. Note that there could be statistical techniques to verify the distributions for symbolic data, however since there are a large number of continuous features that belong to different underlying distributions, we need not perform any unnecessary tests.

5.2 Shifting Distributions

Distribution shift is a largely ignored topic within the machine learning community. However, interest in this area has increased in the past few years. In this section, we discuss recent developments to deal with distribution shift, which we split into two sections to emphasise their difference: time-varying data and distribution shift between the training and test sets.

5.2.1 Time-varying Data

Cavallanti et al. (2006) study a slightly different shifting distribution problem. The problem they deal with is time-varying data sets, as opposed to data that has a sudden

distribution shift between training and test phases (like the KDD '99 data). Although we leave this for future work, we will include a review of their work for completeness.

The authors use a Perceptron algorithm to study two important aspects related to online learning: tracking ability and memory boundedness. The need to track change in the data arises because in certain real-world tasks, such as categorization of text generated by a news feed, using a fixed classifier does not make sense. The algorithm (see Algorithm 10) has a positive input parameter λ which determines the weight decay. The algorithm also maintains a weight vector $\mathbf{w}$, and two more variables: a mistake counter k, and a time-changing decaying factor λ_k (initialised to 1). When a mistake is made on some example $(\mathbf{x}_t, y_t)$, the signed instance vector $y_t\mathbf{x}_t$ is added to the old weight vector, just like in the original Perceptron update rule. However, the difference is that the SPA scales down the old weight before adding $y_t\mathbf{x}_t$, to diminish the importance of early update stages. The scaling factor $(1 - \lambda_k)$ changes with time, since $\lambda_k \to 0$ as more mistakes are made.

If, however, the algorithm is kernel-based, then any mistake it makes results in the corresponding point being stored. This means that the number of support vectors can grow unboundedly, and therefore many variants of the Perceptron algorithm that incorporate a budget have been proposed (see Crammer et al. (2003), Weston et al. (2005), described in Section 2.3). Once the budget limit is reached, these algorithms evict a support vector every time a new point is to be added to the cache. Therefore, the authors proposed a new randomised budget Perceptron that removes a random point from the cache when the limit is reached (see Algorithm 11). They compared their algorithm's random eviction policy with other policies in the literature such as the one by Dekel et al. (2006) on shifting data sets they obtained from the Reuters Corpus Volume 1[1]. These experiments showed the effectiveness of budget algorithms on time-changing data sets, and the robustness of the random policy compared to deterministic ones.

Algorithm 10 The shifting Perceptron algorithm by Cavallanti et al. (2006)

Input: $\lambda > 0$, labeled training data $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$
Initialise: $\mathbf{w}_0 = \mathbf{0}$,$\lambda_0 = 1$,$k = 0$

for $t = 1, 2, \cdots, N$ **do**
  Get instance vector $\mathbf{x}_t \in \mathbb{R}^d$,$\|\mathbf{x}_t\| = 1$
  Predict with $\hat{y}_t = \text{sign}(\mathbf{w}_k^T \mathbf{x}_t) \in \{-1, +1\}$
  Get label $y_t \in \{-1, +1\}$
  if $\hat{y}_t \neq y_t$ **then**
    $\mathbf{w}_{k+1} = (1 - \lambda_k)\mathbf{w}_k + y_t\mathbf{x}_t$, $k \leftarrow k + 1$, $\lambda_k = \frac{\lambda}{\lambda + k}$
  end if
end for

Output: $H(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x})$

[1] http://about.reuters.com/researchandstandards/corpus

Algorithm 11 The Randomised Budget Perceptron by Cavallanti et al. (2006)

Input: Budget $B \in \mathbb{N} \geq 2$, labeled training data $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$
Initialise: $\mathbf{w}_0 = \mathbf{0}$,$s = 0$,$k = 0$

for $t = 1, 2, \cdots, N$ **do**
  Get instance vector $\mathbf{x}_t \in \mathbb{R}^d$,$\|\mathbf{x}_t\| = 1$
  Predict with $\hat{y}_t = sign(\mathbf{w}_k^T \mathbf{x}_t) \in \{-1, +1\}$
  Get label $y_t \in \{-1, +1\}$
  if $\hat{y}_t \neq y_t$ **then**
    if $s < B$ **then**
      $\mathbf{w}_{k+1} = \mathbf{w}_k + y_t \mathbf{x}_t$, $k \leftarrow k + 1$, $s \leftarrow s + 1$
    else
      Let $\mathbf{q}_k$ be a random support vector of $\mathbf{w}_k$ and perform the assignment $\mathbf{w}_{k+1} = \mathbf{w}_k + y_t \mathbf{x}_t - \mathbf{q}_k$, $k \leftarrow k + 1$
    end if
  end if
end for

Output: $H(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x})$

5.2.2 Distribution shift between training and test phases

This section describes the two most recent techniques in the literature that deal with covariate shift between the training and test phases. Both of the discussed methods will be incorporated into our VS framework, and the LR in Section 5.3.

5.2.2.1 Importance Weighting (IW)

Kanamori and Shimodaira (2009) study learning algorithms under the covariate shift. In the literature, covariate shift occurs when the data is assumed to be generated according to a model $P(y|\mathbf{x})P(\mathbf{x})$, where $P(\mathbf{x})$ changes between training and test distributions. So when an algorithm is trained on the plain training distribution, the expected error on the test set will be higher than the case with no shift. To account for this, the authors propose an importance weighting (IW) scheme to push the learning algorithm towards more important regions of the input space (i.e. where more test points lie). To illustrate their work, the authors use a naive estimator (the maximum likelihood estimator), which yields an estimation bias when the assumed statistical model is misspecified. To adjust for this, they introduce the maximum weighted log-likelihood estimator (MWLE). The idea of the MWLE is to downweight large parts of the training data with less importance in the test distribution. This is achieved by weighting each training point $\mathbf{x}$ by

$$w(\mathbf{x}) = p_{te}(\mathbf{x})/p_{tr}(\mathbf{x}),$$

where $p_{te}(\mathbf{x})$ is the density of $\mathbf{x}$ in the test distribution, and $p_{tr}(\mathbf{x})$ is its density in the training distribution. It is clear that portions of the input space where the test distribution is denser would yield a higher value for w, thus guiding the learning algorithm

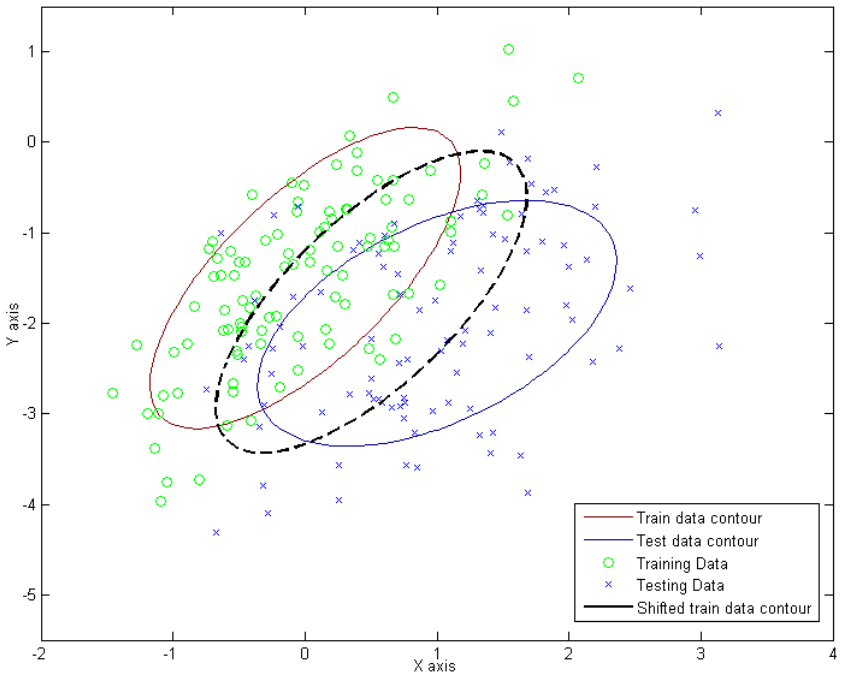


FIGURE 5.1: Plot showing how the training distribution's mean has shifted towards the mean of the test distribution by using KMM. $B = 0.5$, and an RBF kernel was used.

towards this more important region which in turn minimises the expected loss on the test data. A drawback with this method, which is clearly demonstrated in Section 5.3, is that a sufficiently large amount of data is required to obtain a good estimate of the training and test densities. When insufficient data are used, the method could yield incorrect weights and ultimately degrade the performance of the algorithm into which it was incorporated.

5.2.2.2 Kernel Mean Matching (KMM)

Gretton et al. (2009)'s kernel mean matching (KMM) is an approach that aims to find a transformation that shifts the examples of the training set towards those of the test set, matching the means of the two. This reweighted training data will more closely resemble the test data, which is what we would like our learning algorithm to generalise on. The matching is achieved in a high dimensional feature space induced by a kernel function, rather than in the input space of the data. Unlike the IW, there is no density estimation involved, which means that not much data is needed for this algorithm to work.

More formally, the idea of the KMM is to find suitable values of the weights ($\beta \in \mathbb{R}^{n_{\text{tr}}}$) by minimising the discrepancy between the means of the distributions (in feature space) subject to the constraints $\beta_i \in [0, B]$ and $\left|\frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{tr}} \beta_i - 1\right| \leq \epsilon$. The former constraint limits the scope of discrepancy between P_{tr} and P_{te}, and increases robustness by limiting

the influence of individual examples. The latter constraint ensures that $\beta(\mathbf{x})P_{\text{tr}}(\mathbf{x})$ is close to a probability distribution. The objective function is given by the difference of the two empirical means:

$$\left\| \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} \beta_i \Phi(\mathbf{x}_i^{\text{tr}}) - \frac{1}{n_{\text{te}}} \sum_{i=1}^{n_{\text{te}}} \Phi(\mathbf{x}_i^{\text{te}}) \right\|^2 .$$

Using

$$K_{ij} = k(\mathbf{x}_i^{\text{tr}}, \mathbf{x}_j^{\text{tr}})$$

and

$$\kappa_i = \frac{n_{\text{tr}}}{n_{\text{te}}} \sum_{j=1}^{n_{\text{te}}} k(\mathbf{x}_i^{\text{tr}}, \mathbf{x}_j^{\text{te}}),$$

the objective function becomes

$$\left\| \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} \beta_i \Phi(\mathbf{x}_i^{\text{tr}}) - \frac{1}{n_{\text{te}}} \sum_{i=1}^{n_{\text{te}}} \Phi(\mathbf{x}_i^{\text{te}}) \right\|^2 =$$

$$\frac{1}{n_{\text{tr}}^2} \beta^T K \beta - \frac{2}{n_{\text{tr}}^2} \kappa^T \beta + \text{const.}$$

Putting everything together, the quadratic problem to find suitable β becomes:

$$\text{minimise}_\beta \frac{1}{2} \beta^T K \beta - \kappa^T \beta \text{ subject to}$$

$$\beta_i \in [0, B] \text{ and } \left| \sum_{i=1}^{n_{\text{tr}}} \beta_i - n_{\text{tr}} \right| \leq n_{\text{tr}} \epsilon.$$

As an illustration, we applied the KMM to a toy problem (shown in Figure 5.1) involving two 2D-Gaussian distributions. The training and testing distributions (shown in green 'o' and blue 'x', respectively) are both generated from Gaussian distributions with slightly shifted means and covariances ($\mu_{\text{tr}} = [0, -1.5], C_{\text{tr}} = [0.5\ 0.5; 0\ 1]$ and $\mu_{\text{te}} = [1, -2], C_{\text{te}} = [1\ 0.5; 0\ 1]$ respectively). The corresponding contour plots of the distributions are also displayed. After applying the KMM and obtaining weights for the training points, the new weighted mean was obtained for the training data by multiplying each point $\mathbf{x}_i$ by it's corresponding weight β_i:

$$\mu_{\text{Shift}} = \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} \beta_i \mathbf{x}_i$$

The new mean (μ_{Shift}) was used to plot a Gaussian contour and is represented in the figure with a dashed line. The figure clearly shows how the training distribution has shifted towards the testing one in the input space.

5.3 Experiments

In the original VS algorithm, all hyperspheres that a training point fell into had their counts increased by one. This, however, is not the optimal choice when the test distribution is different from the training one, as all input points are incorrectly assumed to carry the same weight. To correct this, we change step 7 in Algorithm 8 (Section 4.2) from:

$$C_{\mathbf{v}} \leftarrow C_{\mathbf{v}} + 1$$

to

$$C_{\mathbf{v}} \leftarrow C_{\mathbf{v}} + p_{te}(\mathbf{x}_t)/p_{tr}(\mathbf{x}_t).$$

Similarly, we change step 13 in Algorithm 8 from:

$$C_{\mathbf{v}} \leftarrow 1$$

to

$$C_{\mathbf{v}} \leftarrow p_{te}(\mathbf{x}_t)/p_{tr}(\mathbf{x}_t).$$

By incrementing the counts of hyperspheres in the new fashion, we force these hyperspheres to have a greater impact in the region where the test points lie. This is because spheres in regions with more test data will have a larger contribution towards votes, and therefore, large parts of the input space with less importance in the test distribution are downweighted. We call this modification VSShift(IW).

As mentioned earlier, the probability densities ($p_{te}(.)$ and $p_{tr}(.)$) can either be known in advance, or estimated from the data presented to the algorithm. In practice, the distributions are rarely known in advance, and so we take the latter approach. There are many density estimation algorithms in the literature, such as the Parzen windows by Parzen (1962), but these are generally very expensive to compute, especially for large data sets. We therefore take a faster (yet less accurate) approach based on histogramming. We first partition each of the features j (e.g. 41 in the KDD '99 case) of the training and test sets into $n = 4$ bins, and increment the values of these bins depending on the feature value, exactly as we would for a histogram. Note that a higher value for n is more accurate than a smaller value but is slower; we choose $n = 4$ as a tradeoff between the two. At the end of this process each feature will have n bins, the sum of the frequencies of these bins totaling N (the number of training samples). Later, when training point $\mathbf{x}$ arrives, we calculate the probability of this point occurring (in either the training or test set) by multiplying the probabilities of each component x_j occurring in its corresponding feature j. For example if x_j fell into bin number two (for the test set), then $p_{te,j}(x_j) = \frac{F_j(2)}{N}$, where $F_j(2)$ is the frequency of values in the jth feature of the test data that fell into the second bin. To tackle the problem of having zero

probabilities, we add a pseudocount $c = 2$ to our calculations as follows:

$$p_{te,j}(x_j) = \frac{F_j(2) + c}{N + n * c}$$

We make this clearer through an example in Section 5.3.1.

In a similar fashion to the way we incorporate the IW to the VS framework, we modify the VS to incorporate the KMM by changing step 7 of Algorithm 8 from:

$$C_{\mathbf{v}} \leftarrow C_{\mathbf{v}} + 1$$

to

$$C_{\mathbf{v}} \leftarrow C_{\mathbf{v}} + \beta_i.$$

where β_i is the weight obtained from the KMM optimisation (in section 5.2) for training point $\mathbf{x}_i$. Similarly, we change step 13 in Algorithm 8 from:

$$C_{\mathbf{v}} \leftarrow 1$$

to

$$C_{\mathbf{v}} \leftarrow \beta_i.$$

5.3.1 IW and KMM on a 2D Artificial Data Set

Before we test on the KDD '99 data, we perform experiments on an artificial 2D-Gaussian data set, to verify our implementation using data we can plot. We will also show the process of calculating the probability of an example $\hat{\mathbf{x}}$ using the IW in detail. All the data generated were 2D-Gaussians using the covariance matrix

$$C = \begin{pmatrix} 0.5 & 0.5 \\ 0 & 1 \end{pmatrix}$$

and the means as follows:

- Training class 1: [0,-1.5]
- Training class -1: [1,1]
- Shifted test class 1: [1,-3]
- Shifted test class -1: [3,0].

The two classes were balanced in the training and test sets, and both sets contained 200 points each. A plot of the data is given in Figure 5.2, clearly showing how the training data has shifted from the testing data.

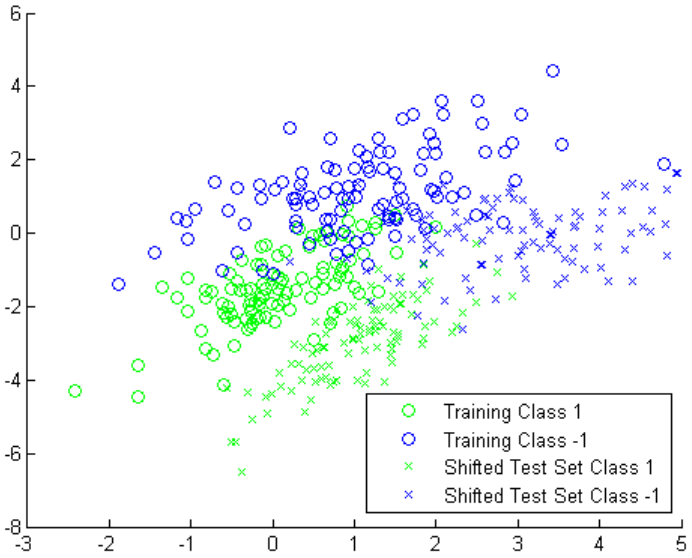


FIGURE 5.2: Plot showing 2D-Gaussian distributions, where the mean of the training data has clearly shifted from its test set counterpart.

We use the point $\hat{\mathbf{x}} = [-0.551, -1.999]$ to illustrate the process of calculating the weight of a point for the IW. After the binning process on the above training set, we have four bins for each of the two features. The frequencies are

$$F_1 = [21, 128, 47, 4]$$

$$F_2 = [20, 86, 74, 20].$$

The corresponding frequencies for the testing data are

$$F_1 = [40, 79, 51, 30]$$

$$F_2 = [8, 60, 67, 65].$$

The cutoff values to create the four bins for feature 1 of the training data are

$$\{[-2.4239, -0.6199], (-0.6199, 1.1842], (1.1842, 2.9882], (2.9882, 4.7923]\}.$$

The value of $\hat{x}_1 = -0.551$ means that it falls into the second bin. This gives a frequency of 128. Given that $N = 200$ and $c = 2$, the probability of -0.551 occurring in feature 1 of the training data is

$$p_{tr,1}(-0.551) = \frac{F_1(2) + c}{N + n * c} = \frac{128 + 2}{200 + 4 * 2} = 0.625.$$

The probability of $\hat{x}_2 = -1.999$ occurring in the second feature of the training data is 0.4231. By multiplying these values together, the probability of $\hat{\mathbf{x}}$ occurring in the training data is 0.2644. Using the same calculations, the probability of $\hat{\mathbf{x}}$ occurring in the testing data is 0.067. Therefore point $\hat{\mathbf{x}}$ contributes $w(\hat{\mathbf{x}}) = \frac{0.067}{0.2644} = 0.2534$ to the hyperspheres it falls within for the VS.

Also as part of our experiments on this data, we test on a sample that has been independently drawn from the training distribution. The results of VS, VSShift(IW), and VSShift(KMM) are given in Table 5.1, where $R = 1$ (chosen arbitrarily) for all the algorithms and every shuffle. Five random shuffles were used to obtain the uncertainties. It is clear that the shift has affected performance, as the prediction accuracy has significantly dropped between the sets drawn from the training distribution (column 1 and column 2) and the test set (column 3). We see that when we test on the data we used for

	Data 1(%)	Data 2(%)	Data 3(%)
VS	87.8 ± 1.4	87.9 ± 0.9	63.1 ± 3.6
VSShift (IW)	87.8 ± 1.4	87.8 ± 0.7	69.3 ± 1.8
VSShift (KMM)	86.7 ± 0.7	89 ± 0.0	81.7 ± 1.6

TABLE 5.1: Prediction accuracy of VS and its weighted counterparts on three data sets: first is the 200 points used for training. Second, on the 200 points independently sampled from the training distribution, and third on the 200 points from the shifted test distribution.

training, the IW does not affect the performance since the weight of each point would be 1. The KMM however, decreased the prediction accuracy. When we test on the independent sample from the training distribution (column 2), we get similar results to column 1, since the VS has correctly partitioned the input space to capture the training distribution. On the shifted test set (column 3), we observe that the IW improves the VS by $\sim 6\%$, whereas the KMM improved the VS by $\sim 18.6\%$. Since the IW requires more data than the KMM to accurately capture the data's density, the KMM is able to outperform it on smaller data, as no density estimation is required. ROC curves for the VS, VSShift(IW), and VSShift(KMM) are given in Figure 5.3. The VSShift(KMM) clearly has the best curve (AUC=0.9234), followed by VSShift(IW) (AUC=0.8972), and finally the VS (AUC = 0.8491). To make a decisive conclusion, we perform experiments on the noisier KDD '99 data set in the following sections, which will pose a challenge to the weighting schemes in a tougher real-world setting.

5.3.2 VS and Logistic Regression using IW on the KDD '99 Data

First, we demonstrate that the shift we showed in Section 5.1.1 causes a decrease in an algorithm's performance ability. When the VS is used on the 10% KDD '99 training and test sets provided, a generalisation performance of $94.03\% \pm 0.0085$ is achieved. However, when the training and test sets are concatenated $(494,020 + 311,029)$, randomly shuffled,

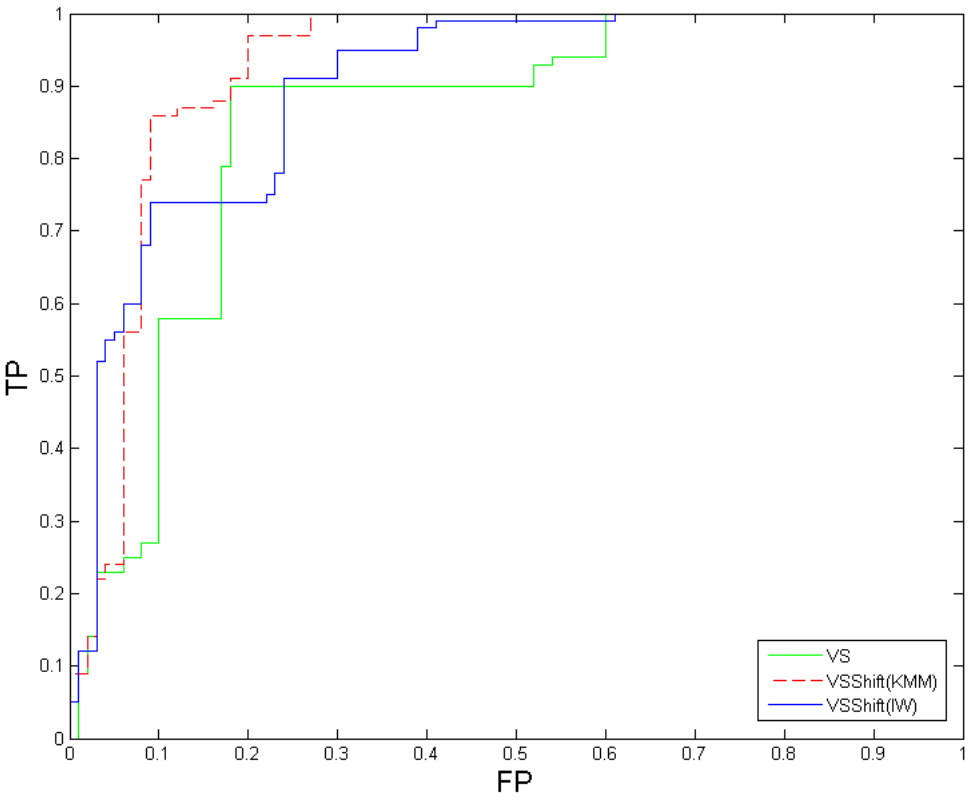


FIGURE 5.3: ROC curves for VS, VSShift(IW), and VSShift(KMM) on an artificial 2D-Gaussian data set.

and then split back into two groups of $494,020$ and $311,029$ as before, the generalisation performance rises to $97.74\% \pm 0.0036$. This shows that VS is now able to capture the rarer intrusion classes that were not in the training set to begin with, and further proves that there is a shift between these data sets. This is also because the shifted normal class is also better captured now.

In this section, we show experimental results on subsets of the KDD '99 data. To verify our results, we train our methods using 20,000 points sampled from the training set, and test on:

- the 20,000 points used for training
- 20,000 points independently sampled from the training data
- 20,000 points randomly sampled from the test distribution.

We expect the prediction accuracy to decrease as we move down the list. This is because for the first subset our learning algorithm has already seen the data, and for the second subset performance will be less since the testing data would be unseen, yet higher than the third since this data is sampled from a different underlying distribution. We shuffle the above data five times to obtain uncertainties.

Initially we experimented with logistic regression trained using gradient descent to see whether the ratio weighting would improve a parametric learning algorithm. The logistic regression is used for the prediction of the probability of an event happening by fitting data onto a logistic curve, and is preferable to ordinary linear regression when the dependent variable is binary. Pseudocode for this is given in Algorithm 12, where the gradient is

$$\frac{\partial E}{\partial w_j} = 2\sum_{i=1}^{N} \frac{e^{-\mathbf{w}^T\mathbf{x}_i} \cdot x_{ij}}{(1+e^{-\mathbf{w}^T\mathbf{x}_i})^2} \left(\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}_i}} - y_i \right).$$

Algorithm 12 Logistic regression trained using gradient descent

Input: labeled training set $\langle(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\rangle$, where $\mathbf{x}_i \in \mathbb{R}^d$, η - learning rate.
Output: a weight vector $\mathbf{w}$.

```
for i = 1, ..., d do
    w_i ← rand(-0.5,0.5)
end for
REPEAT
for i = 1 ..., d do
    Δw_i ← 0

    for t = 1, ..., N do
        Δw_i ← Δw_i + ∂E(x_t)/∂w_i
    end for
end for
for i = 1, ..., d do
    w_i ← w_i − ηΔw_i
end for
UNTIL CONVERGENCE
```

To incorporate the importance weighting technique into the LR training, we change (in Algorithm 12):

$$\Delta w_i \leftarrow \Delta w_i + \frac{\partial E(\mathbf{x}_t)}{\partial w_i}$$

to

$$\Delta w_i \leftarrow \Delta w_i + \left(\frac{p_{te}(\mathbf{x}_t)}{p_{tr}(\mathbf{x}_t)} \right) \frac{\partial E(\mathbf{x}_t)}{\partial w_i}.$$

This change will guide the logistic regression's search towards regions of the space where more of the test data is found. For our empirical study, we trained the logistic regressions (both weighted and unweighted) using the $20,000$ points mentioned above. To conduct a fair comparison between the two versions, we passed the randomly initialised weight vector from the logistic regression to the weighted version. This removes the random element and allows a clear and objective comparison, and the results are shown in Table 5.3. By testing on the three aforementioned subsets, we observe that a higher generalisation accuracy was achieved when testing on the second subset than on the third. This is expected for any algorithm, as the distribution from which the former was sampled is the same as the training data, whereas the latter was sampled from a shifted

maxIDis	maxNDis	VS Acc(%)	VSShift(IW) Acc(%)
2.2	0.6	93.65	**94.01**
2.2	0.8	92.13	**93.56**
2.2	0.9	93.70	**94.56**
2.2	1.0	94.04	**94.54**
2.3	0.6	93.20	**93.41**
2.3	0.9	93.55	**94.14**
2.3	1.0	93.64	**94.32**
2.4	0.6	93.28	**93.40**
2.4	0.9	93.48	**94.13**
2.4	1.0	93.59	**94.31**
2.5	0.6	93.40	**93.85**
2.5	0.9	93.33	**94.09**
2.5	1.0	93.31	**94.28**
2.6	0.6	93.23	**93.71**
2.6	1.0	93.28	**94.26**
2.7	0.6	93.21	**94.01**
2.7	1.0	93.30	**94.23**
2.8	0.6	93.09	**93.93**
2.8	1.0	93.36	**94.16**
2.9	0.6	92.71	**93.55**
2.9	1.0	92.89	**94.07**

TABLE 5.2: Prediction accuracy of VS and VSShift for different random values of the radii in the range $\{0.5, 3\}$ indicated by the plots of Section 4.2. Results obtained by training on 20,000 training points, and testing on 20,000 points from the (shifted) test set subsampled from the KDD '99 data.

data distribution. It is quite clear that accounting for distribution shift significantly improved the logistic regression.

To demonstrate the improvement achieved by weighting the hyperspheres, the VS and VSShift(IW) algorithms were run for different random values of the radii (in the range discussed in Section 4.2) on subsamples of size 20,000 (drawn from the KDD '99 data), and the results presented in Table 5.2. It is clear that the VSShift(IW) algorithm outperforms VS every time. In Table 5.3 (radii chosen arbitrarily as maxIDis =2.9 and maxNDis=1.0 from Table 5.2), as we expected, we observe that the shift in the data causes the algorithms' performance levels to drop, since the performance drops between column 1 and column 3. Using the ratio weighting for the hyperspheres in the VS corrected the problem. This trend is also visible in the LR case; the performance of the algorithm declines due to the shift, which is taken care of by directing the search towards more test-data-dense regions using the IW scheme during training. The increase in performance we observe in the second column when accounting for shift (Table 5.3), is somewhat unusual. This is because, theoretically speaking, there should not be a shift in the data, yet we observe an improvement when we use the ratio weighting. This can be attributed to the fact that we are empirically estimating the distributions from

a small sample of the data. Had we known the actual distributions of the training and test data, the values in this column would have remained constant, just like in the first column.

	Data 1 (%)	Data 2(%)	Data 3(%)
VS	$\mathbf{94.59 \pm 0.11}$	94.35 ± 0.11	93.08 ± 0.41
VSShift (IW)	$\mathbf{94.59 \pm 0.11}$	$\mathbf{95.82 \pm 0.14}$	$\mathbf{94.30 \pm 0.44}$
LR	88.13 ± 7.4	78.51 ± 24.5	56.60 ± 15.11
Weighted LR	88.13 ± 7.4	85.36 ± 13.0	80.07 ± 0.54

TABLE 5.3: Generalisation accuracy of VS, VSShift(IW), LR, and WLR on three data sets: First is on the 20,000 points used for training. Second, on 20,000 points randomly sampled from the training set, and third on 20,000 points randomly sampled from the test data. The radii were: maxIDis = 2.9, and maxNDis = 1.

5.3.3 IW versus KMM on the KDD '99 Data

Finally, we compare the IW with the KMM. Since the latter requires the solution of a QP of order $O(N^3)$, we were unable to run it on data samples of size $20,000$. Instead, we train on 400 points from the training set, and test on:

- The 400 points used for training
- 400 points randomly drawn from the KDD '99 training distribution
- 400 points randomly drawn from the KDD '99 testing distribution.

	Data 1(%)	Data 2(%)	Data 3(%)
VS	$\mathbf{95.75 \pm 0.0}$	92.44 ± 0.31	92.94 ± 0.37
VSShift (IW)	$\mathbf{95.75 \pm 0.0}$	87.50 ± 0.2	90.88 ± 0.85
VSShift (KMM)	95.06 ± 0.24	$\mathbf{92.75 \pm 0.46}$	$\mathbf{93.75 \pm 0.20}$

TABLE 5.4: Generalisation accuracy of VS and its weighted counterparts on three data sets: first is on the 400 points used for training. Second, on 400 points randomly sampled from the training set, and third on 400 points randomly sampled from the test data. The data were shuffled five times, with maxIDis=2.9 and maxNDis=1.0.

Using training data of this size enables us to compare the IW and KMM in situations where not much data are available for density estimation. It also allows us to test the KMM's ability to match the means of noisy real-world data. It also poses a challenge to the vanilla VS, as it will have to carve the input space appropriately using a very small fraction of the available data. Given that the VS algorithm has no random element and that we pass exactly the same data and radii to all three algorithms, the results shown in Table 5.4 are a direct comparison between the KMM and the IW. We also did not perform any cross-validation to obtain the radii, we simply use maxIDis=2.9 and

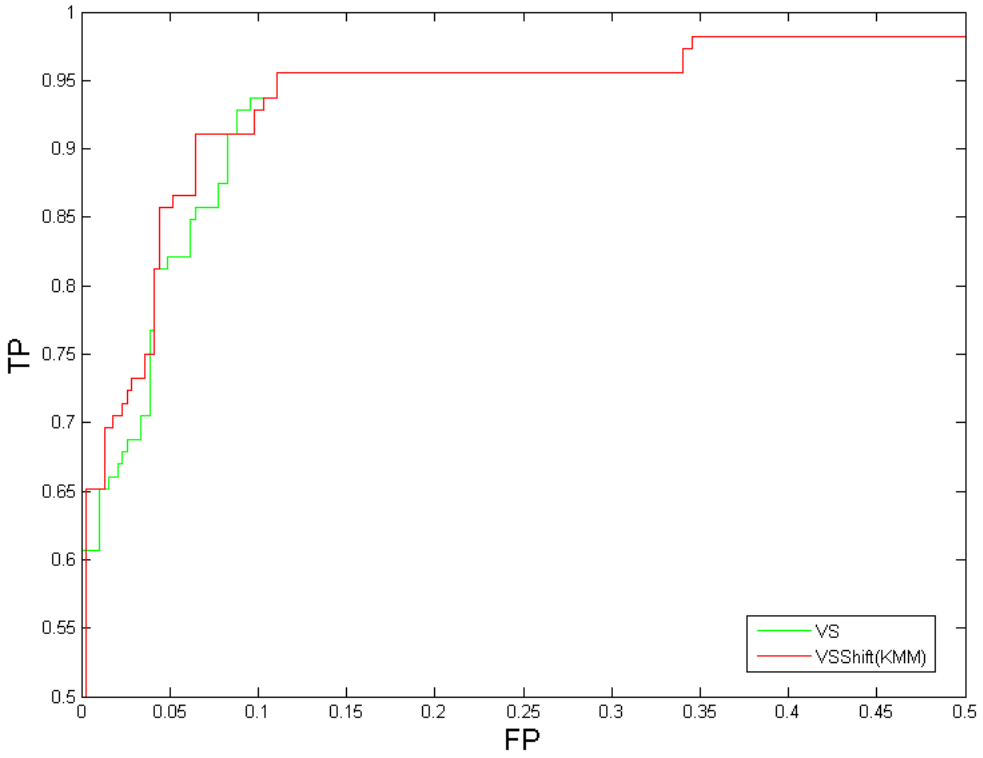


FIGURE 5.4: Plot showing ROC curves for VS and VSShift(KMM) on a subset of the KDD '99 data. The latter clearly has a better curve, as it is closer to the top left corner. The VS had a classification rate of 91.4% and an AUC of 0.9612, while the KMM version had an accuracy of 92.80% and an AUC of 0.9628.

maxIDis=1.0 as we did above. The reason is that we only want to compare the KMM and IW together, the absolute prediction accuracy is irrelevant. As can be seen from the results in the table, the KMM outperforms both the VS and the VSShift(IW) on the shifted test data. This is because the IW needs a large sample size to obtain an accurate estimate of the probability distributions, and 400 points are simply not enough to capture the distribution density of the KDD '99 Cup data. When this is the case, the incorrect weights returned by the IW actually decrease the performance of the algorithm, which is clearly shown in column 3 of Table 5.4. For the artificial data sets of Section 5.3.1, the IW improved the VS despite only using 200 points, however those simpler data had only two dimensions and were far less noisy than the KDD '99 data. We also observe, as we did in Section 5.3.1, that the KMM decreases the performance of the VS on the training data that does not contain a shift. Figure 5.4 gives two ROC curves for the VS and VSShift(KMM) on the test data. The KMM version has a better curve and a higher AUC, achieving 0.9628 compared to VS' 0.9612. Another plot showing a comparison of ROC curves between VSShift(IW) and VSShift(KMM) is given in Figure 5.5. This ROC curve clearly shows the difference between the IW and the KMM when used on small data: the IW decreased the VS' performance whereas the KMM improved it significantly. Figure 5.6 gives a plot of generalisation accuracy versus sample size for the VS, VSShift(IW), and VSShift(KMM). It clearly shows the trends observed in the

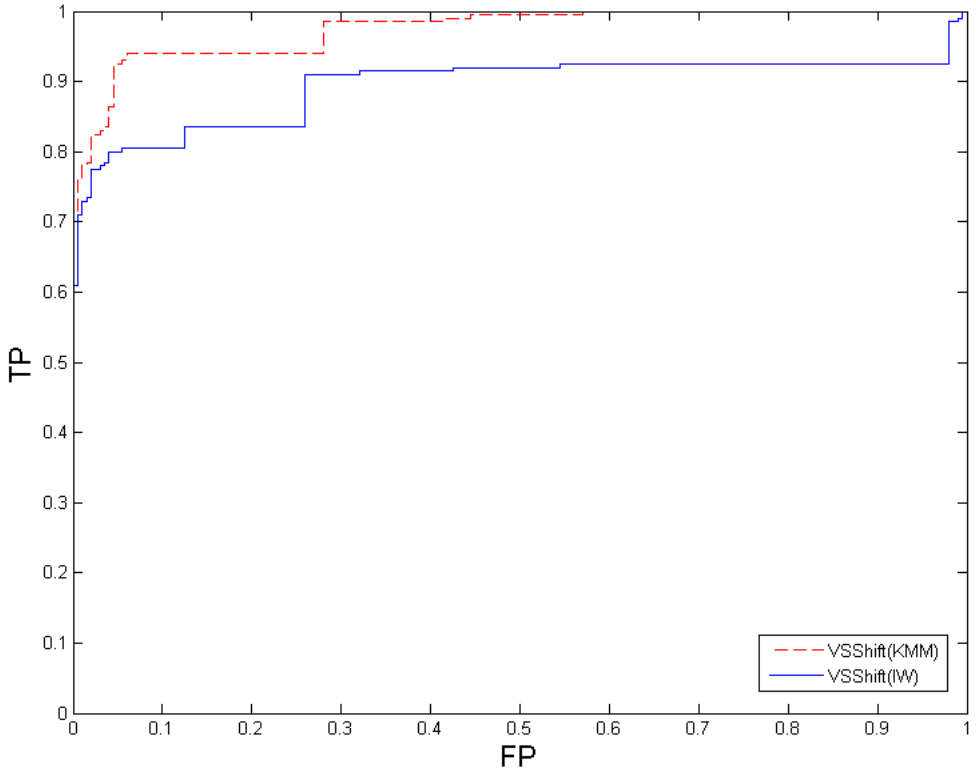


FIGURE 5.5: Plot showing ROC curves for VSShift(KMM) and VSShift(IW) on a subset of the KDD '99 data. The former had an AUC of 0.9736 while the latter achieved 0.8938

chapter, where the VS is the base algorithm. We observe that the IW clearly improves the overall generalisation accuracy of VS, except on very small sample sizes where the IW actually worsens the VS due to the incorrect weights. As mentioned earlier, the IW needs a lot of data to estimate the input densities accurately, and this is clearly visible in the figure; the larger the sample size, the lower the uncertainty of the IW is. When the sample size is small, we have higher uncertainties. Also when the data set is small, the KMM outperforms the VS and the VSShift(IW). Figure 5.6 only has two points plotted for the KMM since we were only able to run it on the subsets of size 100 and 1000; 5000 points proved to be too much for the KMM to handle in MATLAB using a standard desktop PC. Therefore, we recommend the use of the KMM on smaller sample sizes, and the IW when sufficient data are available to produce a good estimate of the densities (or when the densities are known in advance).

5.4 Conclusion

In this chapter, we tackle the problem of large scale learning when the data contains a distribution shift between the training and testing phases. Using artificial data and subsets of the KDD '99 data, we demonstrated how the IW and KMM techniques can be adopted into our one-pass Voted Spheres classifier to deal with the disparities between

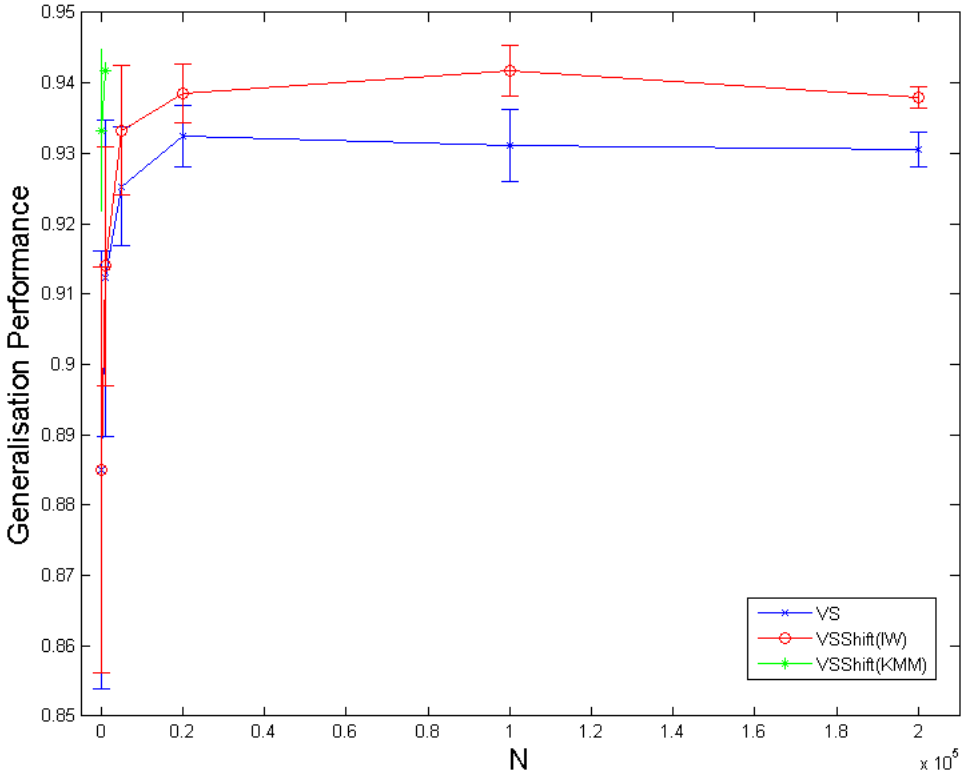


FIGURE 5.6: Plot showing the generalisation performance vs the sample size for VS, VSShift(IW) and VSShift(KMM) on subsets of the KDD '99 data. 10 random shuffles were used to obtain the points and uncertainties.

the training and testing data sets. We achieved significant improvements using both methods, especially the IW, since the KMM cannot scale to large data sets given the quadratic programming it requires. The IW requires a sufficiently large amount of data to work, which is not a problem in our case since we explicitly aim to tackle the issues of massive data sets and shifting distributions simultaneously. However, the KMM is better suited to small data, as no density estimation is required. We also recommend its use on data that is confirmed to have a shift, as we discovered that the KMM decreases the performance on data that does not contain a shift. This is shown in the first columns of Table 5.1 and Table 5.4.

Chapter 6

Applications on Brain-Computer Interfaces Data

Brain-Computer Interfaces enable the human brain and computers to interact via a new communication channel. It transforms mental intentions into control commands by analysing the brain's bioelectrical activity (Quigguo et al. (2007)). However, the way data are collected causes an inherent distribution shift in the data, which is ignored by researchers in the field. They generally apply traditional batch machine learning algorithms on their data, such as SVMs and k-nearest neighbours. We take a different approach in this chapter by applying our one-pass VS and VSShift (including both IW and KMM) methods on data from the BCI Competition III to compare against SVMs (linear and RBF kernels) and all the published results from the competition submissions. This is, to the best of our knowledge, the first time a distribution-shift-adapted one-pass algorithm has been applied on BCI data, which is a contribution to the BCI field in its own right. Our results show that the VS-based methods outperformed the SVM on the BCI data, and explicitly accounting for the shift further improved the results, especially when testing on the independent test sets for which there is the greatest shift. The VSShift also performed competitively with the published results of the competition.

6.1 Introduction

Generally for BCI tasks the data set sizes are small, especially in comparison to the KDD '99 data we used in the rest of the thesis. Because of this, researchers in the field were not concerned with online algorithms, instead resorting to the more common and established batch methods. Also inherent in the BCI field are distribution shifts between the training and test sets. This is unavoidable however, since this shift arises in the way data are collected: training data are collected by making a subject perform some actions, while the test data are collected some time later, from the same subject

performing the same actions. The subject could be for instance tired or demotivated, which directly affects the brain's electrical signals. This shift is not accounted for by the traditional machine learning algorithms applied in the BCI domain, and so a gap exists in the field which our work can fill.

Research into BCI has been very active over the past decade, and is still a very fast growing field (Wolpaw et al. (2000); Dornhege et al. (2007)). For us, it is essentially a familiar classification problem, as it involves recording and analyzing EEG signals and associating them with their corresponding mental states. For instance, a subject imagining the movement of their left hand will produce an EEG signal that will be recorded. To learn the mapping of this EEG signal with the mental state of the action performed, supervised classification techniques are used. Generally, linear classifiers such as Fisher's linear discriminant have been preferred (Müller et al. (2004), Blankertz et al. (2006) and Lotte et al. (2007)). But more recently, other classifiers have been used, such as k-NN by Blankertz et al. (2002) and SVMs by Lee et al. (2005). For the SVMs, Rakotomamonjy et al. (2005) argue that linear kernels are best to use on such data because the difficulty of BCI data (and other biosignal classification problems) arises from the variability of the data sets, and so a linear kernel will prevent any overfitting of the training data. Quigguo et al. (2007) also argue in favour of this, claiming that linear classifiers are more robust and have better generalisation performance than non-linear classifiers when finite training samples are available. Basically, this is the community's attempt to solve the distribution shift: avoid overfitting the training data (since there's a shift) by fitting a linear classifier. This is visible from the techniques applied on the BCI Competition III, the majority of which are linear. However as Lotte et al. (2007) pointed out, evaluating these classifiers is not straightforward as the experimental setup, pre-processing, and feature selection differ from study to study. This can be seen from the methods applied on the BCI Competition III (shown in Table 6.7 and Table 6.13), where the majority of classifiers are linear, and the difference in performance between methods arises due to the preprocessing.

In this chapter, we apply our one-pass VS method to the BCI domain and compare it with the state-of-the-art SVM, and also use the two weighting schemes from Chapter 5 to tackle the issue of having intra- and inter-subject variability in the produced EEG signals. Despite there being many published results using SVMs, we perform our own SVM experiments since the performance of classifiers applied on such data relies heavily on the preprocessing the data has gone through. Since every published result has undergone different preprocessing, our SVM experiments enable a direct classifier comparison with the VS techniques. Finally, we show how VSShift compares to the results of all the participants of the BCI competition III.

6.2 BCI Competition III

This competition[1] was set up on December 2004, and ran until May 2005, to give researchers the opportunity to validate signal processing and classification methods for Brain-Computer Interfaces. Despite the fact that such a competition is far from allowing practitioners to validate BCI systems as a whole, the competition organisers were looking for interesting contributions that could ultimately progress the BCI field.

There were several tracks to the competition, each with its own goal, data sets, and performance measure. Each data set came in two parts: a labeled training set, and an unlabeled test set, where all of the data sets consisted of single-trials of spontaneous brain activity. The goal, as with any such competition, was to predict the labels of the test set such that the given performance measure was maximised. We use two data sets that contain distribution shifts in our experiments, since the other data sets from the competition dealt with other issues such as multi-class classification and non-stationary data. These sets, which are described in more detail below, are data set I and data set IVa. The former is ECoG data and involves session-to-session transfer (explained in Section 6.2.1), while the latter involves small training sets to be solved by subject-to-subject transfer (explained in Section 6.2.2). Given that the competition is over and the true test labels have been published, we are able to compare our algorithms with published results.

6.2.1 Data set I

The way this data set (Lal et al. (2005)) was generated reflects the real situation in which BCI researchers collect their data, and shows why their data are plagued with distribution shifts. The training and test data were recorded from the same subject performing the same task, but on two different days with around a week in between. Factors such as the patient being in a different state with respect to motivation or fatigue would directly affect brain electrical activity, which gives rise to the distribution shift. Furthermore, in the gap week, elements of the recording system might have changed slightly, such as the electrode positions and impedances. The competition organisers were hoping that contestants would design classifiers that are capable of classifying the shifted test set without the need for re-training.

The data were generated by having the subject perform imaginary movements of either their left pinkie finger or their tongue. The electrical brain–activity's time series was recorded during these experiments using an 8x8 ECoG platinum electrode grid (0.016-300Hz) which was placed on the right motor cortex. All recordings were performed with a sampling rate of 1000Hz. After the recorded potentials were amplified, they were stored

[1] http://www.bbci.de/competition/iii/

as microvolt values. Every trial consisted of either an imaginary tongue or an imaginary finger movement and was recorded for a duration of 3 seconds. The resultant training and test sets had 278 and 100 trials respectively, and the preprocessing performed on them is described in Section 6.2.3.

6.2.2 Data set IVa

This data set was created to address a specific issue in the BCI domain: the challenge of learning with small amounts of training data. Before data can be collected, a user has to perform time-consuming and boring calibration measurements. An important objective in BCI research is reducing the time needed for this initial measurement. An approach to dealing with this could be to reuse information from other subjects' measurements to adapt to a new subject. This can be seen as a form of distribution shift. This data set was of interest to us since it contains a distribution shift, and yet only has a small amount of data which poses a challenge for our weighting schemes.

During this experiment, five healthy subjects sat in a comfortable armchair with their hands on the rests. Only data from the four initial sessions without feedback were included in the data set. Visual cues indicated for 3.5 seconds which of the following three motor imageries the subject should perform:

- Left Hand (L)
- Right Hand (R)
- Right Foot (F).

Between each presentation of a target cue, there was a random period of time (between 1.75 seconds and 2.25 seconds) for the subject to relax. There were two different types of visual stimulation:

1. where targets were indicated by letters appearing behind a fixation cross, and
2. where a randomly moving object indicated targets.

For subjects *al* and *aw*, two sessions of both types were recorded, while from the other subjects three sessions of type (2) and one session of type (1) were recorded. The number of training and testing points for each subject are given in Table 6.1, and the preprocessing is described in Section 6.2.3. More details are given on the competition's website. The fact that there are far fewer training samples than testing ones would be a test for the VS, to see whether it can partition the input space in a way to achieve good performance using very little data.

Subject	# training points	#test points	Features
aa	168	112	8
al	224	56	8
av	84	196	8
aw	56	224	8
ay	28	252	8

TABLE 6.1: Number of training and test points for the five subjects of data set IVa

6.2.3 Preprocessing

Our focus in this chapter is that of using our one-pass VS and VSShift on the small, shifted BCI data, and not that of signal processing. Because of this, the preprocessing of the data in this chapter was done by Dr CQ Chang[2] of the University of Hong Kong, and we were supplied with the final preprocessed data. We briefly discuss the process below.

For data set I, the data were preprocessed as in Quigguo et al. (2007). Initially, they were downsampled using decimation (from 1000Hz to 100Hz). For feature set 1, the downsampled data were filtered using a low-pass filter [0-3]Hz. The filter was Chebyshev type I filter. Then, Common Spatial Pattern (CSP) analysis (introduced by Koles et al. (1990)) was applied, returning four features. Using the low-pass filtered data used for feature set 1, feature set 3 was generated using the mean magnitude of 18 channels (as described by Quigguo et al. (2007)), giving us 18 features which were normalised to $[-1, +1]$. The channels used were 12,14,18,21–24,29–32,37–40, and 46–48.

For feature set 2, the original downsampled data above was passed through a band-pass filter [8-30]Hz, followed by CSP to generate six features.

For data set IVa, only the signals collected during the 0.5–2.5 second window after the presentation of the visual cue were kept, as Thomas et al. (2009) argue that this period contains the most information. After that, each input signal was passed to a filter bank, with bandwidth frequencies: [8-12]Hz, [14-18]Hz, [18-22]Hz, and [20-24]Hz.

For each filtered EEG signal, do CSP to obtain eight features in total.

6.2.4 Kolmogorov-Smirnov test on the BCI data sets

In this section, we perform the Kolmogorov-Smirnov test for two independent samples on a dimension by dimension basis in the same spirit as in Section 5.1.1. This is to test whether the training and test sets of data I and data IVa were drawn from the same distribution. Table 6.2 shows the number of features that follow different distributions between the training and test sets for data I.

[2] http://www.eee.hku.hk/people/cqchang.html#

Feature Set	# dimensions	#dimensions with different distributions
1	4	3
2	6	5
3	18	18

TABLE 6.2: Number of dimensions that follow different distributions between the training and test sets as indicated by the Kolmogorov-Smirnov test. The data are the three feature sets of data I.

For data set IVa, five similar experiments were conducted, one for each of the five subjects. Table 6.3 shows the number of dimensions that had a shift between the training and test phases of data IVa, as indicated by the Kolmogorov-Smirnov test.

Subject	# dimensions	#dimensions with different distributions
aa	8	6
al	8	4
av	8	2
aw	8	8
ay	8	8

TABLE 6.3: Number of dimensions that follow different distributions between the training and test sets as indicated by the Kolmogorov-Smirnov test. The data are the five subjects of data IVa.

We conclude from the results of the tests in this section that these data sets do indeed suffer from a distribution shift, in accordance to what was mentioned by the competition's organisers.

6.3 Experiments

In this section, we present an empirical comparison of several algorithms on data sets I and IVa of the BCI competition III data, which we proved via a statistical test to have training and test data from different underlying distributions. These data sets are small, especially when compared to the large scale data we tackle in this thesis. Nevertheless, data generated from BCI systems contain a shift, and naturally with time more data will be generated as technology advances. Here, we perform experiments to verify whether using online algorithms can be effective in this domain, and whether accounting for this shift using the two methods described in Chapter 5 enhances the VS on small BCI data. For each of the two data sets used, we:

- trained our algorithms on the training data, and tested on the training data itself,
- split the training data in half, used one half for training and the other half for testing,

- mixed the training and test sets, shuffled them, then split back into their original proportions for training and testing,
- trained on the training data, and tested on the test data.

This way, we can verify the validity of our results, as we expect performance to degrade as we move down the list. Training and testing on the same sample is trivial, and would yield the best results since the algorithm is testing on previously seen data. Training and testing on different subsets drawn from the training data would perform slightly worse, since it is testing on unseen data, but would perform better than on the third set above since the two sets would be from the same underlying distribution. Shuffling the training and test sets would further worsen the performance of the algorithms because the training set would have data from two different distributions, making it more difficult for a learning algorithm to capture either of them. Finally, we expect to get the least generalisation performance when we train on the training set, and test on the independent test set. This is because the test data is previously unseen, and in our case, also contains a shift. These trends are visible in all the results of data set I and data set IVa. For the SVMs, we set $C = 0.01$, and $\sigma = 0.5$ when a RBF kernel was used (chosen via five-fold cross validation). For the VS-based techniques, we followed the procedure described in Section 4.2.1 to set the radii: we used a small subsample of the training data to calculate the average pairwise distances, and performed five-fold CV on the training data to set the final radii. These radii were then used for testing. For each value in our tables, the uncertainties were obtained by shuffling the data five times.

Despite the fact that SVMs with linear kernels are generally used for BCI tasks, we use a non-linear RBF kernel in our experiments (on data set I) as well as the linear one. We observe through our results in Table 6.4, Table 6.5, and Table 6.6 that linear kernels are sufficient for this task, in accordance with what researchers do in the literature. For instance, in Table 6.5, the RBF kernel actually decreases the performance of the SVM on the training data, hardly improves the results on the independent training data and the shuffled test data, while not improving the performance on the test data at all. This could mean, as researchers in the BCI field claim, that the RBF kernels are overfitting the training data and are underperforming on the shifted test set. This is clear in the third column of the results on data set I, where the linear SVM achieves comparable results to the RBF kernel, albeit in much less time. A component-by-component plot of the four dimensions of feature set 1's training data is given in Figure 6.1. These plots are not accurate depictions of the data in input space, as there is a component missing in each plot, but help demonstrate the separability of the two classes. We can see from the plots that, overall, the two classes of the data are not highly overlapping. Using a linear classifier such as a soft-margin SVM would not return a perfectly separating hyperplane, but would try to separate the two classes as cleanly as possible. In column one of Table 6.4, we see how well the classifiers separated the data. The VS correctly captured the density of the data, almost perfectly separating the data. It is clear that

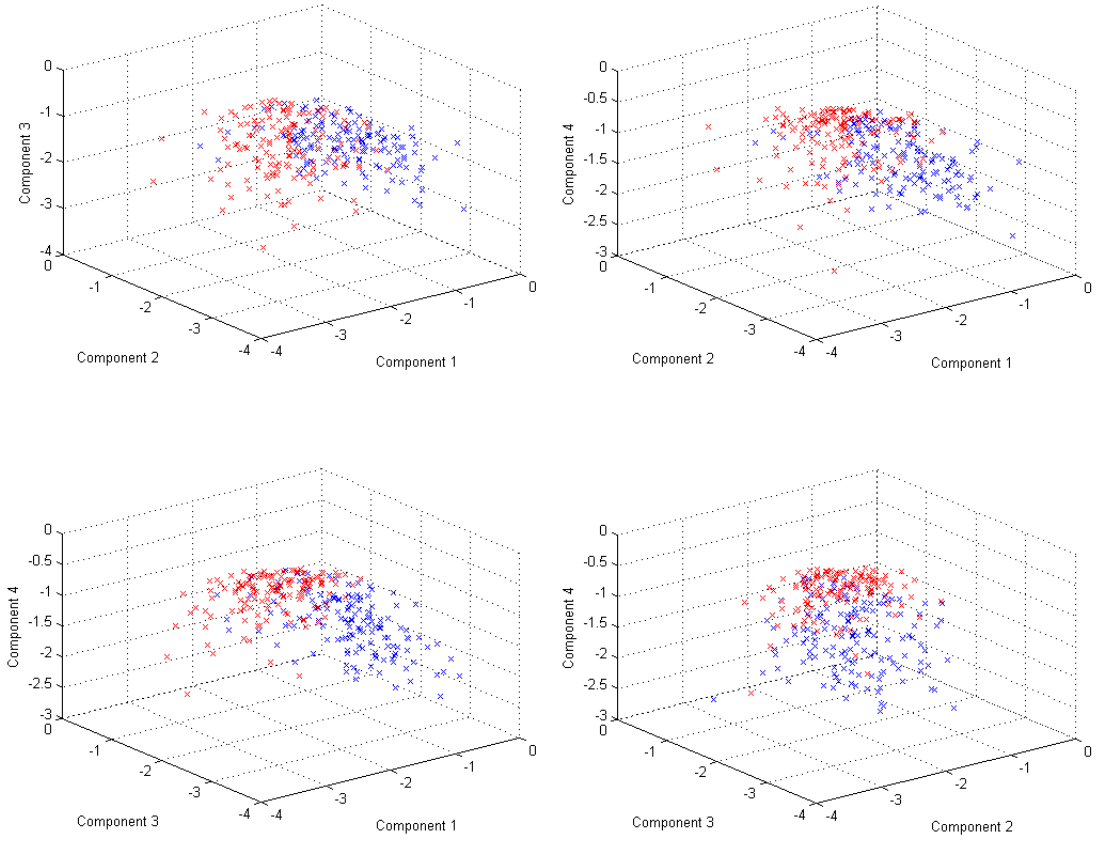


FIGURE 6.1: Component-wise plot of the four-dimensional feature set 1 of data I. The red data are class 1, and the blue data are class -1.

the voting scheme was effective in classifying points in overlapping regions. Because of this, we only run a linear kernel SVM on data set IVa. We observe from the three tables of data set I, that the KMM and IW constantly improve the generalisation accuracy of the VS, especially in the last two columns (which contain the shift), and are always the top performing among the rest. The vanilla VS also outperforms the SVM on this BCI task, without the need to account for distribution shift. For instance, in Table 6.6, the VSKernel(RBF) achieves a 12% increase in classification accuracy over an SVM with an RBF kernel. Given that we know there is a shift in the data, incorporating the IW into VS increased the improvement to 14%. Finally, in Table 6.7, we compare the performance of VSShift to the 27 competition submissions[3] on data I. The VSShift on feature set 2 outperformed the top result, which was a linear SVM by 3%. The VSShift on feature sets 1 and 3 also performed well, ranking 15th and 20th respectively. It is worthy to note that the preprocessing step plays a key role in the performance of the classifier. For example our run of the linear SVM on feature set 2 gave an 87% accuracy, whereas plenty of researchers applied linear SVMs and obtained different results (as shown in Table 6.7). This is also clear from Table 6.4, Table 6.5, and Table 6.6 where we use linear SVMs as well, but perform different preprocessing on the data.

[3]http://www.bbci.de/competition/iii/results/index.html#tuebingen

	Accuracy(%)			
Algorithm	Train data	Indep. train data	Test data(shuffled)	Test data
VS	82.95 ± 1.7	85.61 ± 1.1	82.20 ± 2.4	66
Linear SVM	**85.61** ± 0.0	83.88 ± 4.3	77.60 ± 3.1	62
VSKernel(RBF)	83.09 ± 1.3	86.19 ± 2.7	80.60 ± 3.2	69
SVM (RBF)	85.25 ± 0.0	82.16 ± 6.4	78.00 ± 3.7	63
VSShift(IW)	82.95 ± 1.7	**86.33** ± 1.4	81.60 ± 2.7	68
VSShift(KMM)	81.94 ± 1.7	86.19 ± 0.94	**82.60** ± 1.1	**72**

TABLE 6.4: Table showing generalisation accuracy using five random shuffles on Feature Set 1 derived from data set I. This feature set contains four dimensions

	Accuracy(%)			
Algorithm	Train	Indep. train	Test (shuffled)	Test
VS	90.5 ± 1.0	92.66 ± 2.9	92.40 ± 1.1	71
Linear SVM	**92.08** ± 0.0	88.63 ± 2.3	88.00 ± 1.2	87
VSKernel(RBF)	90.22 ± 1.4	**94.39** ± 0.60	**94.00** ± 1.4	90
SVM (RBF)	91.73 ± 0.0	90.94 ± 1.3	88.40 ± 1.7	87
VSShift(IW)	90.5 ± 1.0	91.94 ± 0.60	**94.00** ± 0.2	91
VSShift(KMM)	90.5 ± 1.0	92.37 ± 1.6	92.80 ± 2.3	**94**

TABLE 6.5: Table showing generalisation accuracy using five random shuffles on Feature Set 2 derived from data set I. This feature set contains six dimensions

	Accuracy(%)			
Algorithm	Train	Indep. train	Test(shuffled)	Test
VS	**100** ± 0.0	**68.06** ± 2.8	67.00 ± 2.9	58
Linear SVM	83.45 ± 0.0	47.34 ± 0.60	44.40 ± 3.9	52
VSKernel(RBF)	**100** ± 0.0	63.45 ± 2.5	63.20 ± 2.6	63
SVM (RBF)	**100** ± 0.0	47.05 ± 1.4	48.00 ± 1.2	51
VSShift(IW)	**100** ± 0.0	63.88 ± 3.4	63.20 ± 1.9	**65**
VSShift(KMM)	73.60 ± 2.0	66.33 ± 2.0	**67.80** ± 4.0	61

TABLE 6.6: Table showing generalisation accuracy using five random shuffles on Feature Set 3 derived from data set I. This feature set contains 18 dimensions. Features were normalised to $[-1, 1]$

Rank	Method	Contributor	Accuracy(%)
1.	**VSShift(KMM) w/ Feat. set 2**	**	**94**
2.	Linear SVM	Qingguo Wei	91
3.	Regularised LR	Paul Hammon	87
4.	LR	Michal Sapinski	86
4.	Mahalanobis inter-class dist.	Mao Dawei	86
4.	Hyperplane	Alexander D'yakonov	86
4.	LDA	Liu Yang	86
8.	LDA	Florian Knoll	84
8.	LDA	Zhou Zongtan	84
10.	Semi-Supervised (quadratic classifier)	Jianzhao Qin	83
11.	Cluster Mean	Matthias Krauledat	82
12.	Linear SVM	Kiyoung Yang	81
12.	N/A	Martin Hieden	81
14.	Ensemble of NNs and Fisher Discriminant	Archis Gore	79
14.	Robust Discriminant	Elly Gysels	79
15.	**VSShift(KMM) w/ Feat. set 1**	**	**72**
17.	FDA	Xiaomei Pei	69
18.	Bayes class. and SVM	Ehsan Arbabi	67
19.	Sparse model from 'finite prediction error'	Florian Popescu	66
20.	**VSShift(IW) w/ Feat. set 3**	**	**65**
20.	Linear SVM	Hyunjin Yoon	65
20.	Probabilistic classifier	Guido Nolte	65
23.	N/A	Timothy Uy	60
24.	Discriminative biorthogonal bases	Wit Jakuczun	59
25.	Linear model with Tikhonov regularisation	Ken Wong	58
26.	Attribute Clustering Network via Nearest Neighbours	Xi-Chen Sun	54
27.	NN (single layer FF) with ELM training	Nanying Liang	50
28.	RBF SVM	Bin An	48
29.	1-NN	Miharu Nishino	44
30.	NN and SVM	Yan Ning	22

TABLE 6.7: Comparison of the results achieved by VSShift with all 27 submissions on data set I (http://www.bbci.de/competition/iii/results/index.html#tuebingen). Overall accuracy is 50% for a random classifier.

	Accuracy(%)			
Algorithm	Train data	Indep. train data	Test data(shuffled)	Test data
VS	$\mathbf{99.05} \pm 0.33$	94.05 ± 5.9	79.64 ± 1.7	66.07
Linear SVM	98.21 ± 0.0	95.48 ± 2.4	81.61 ± 4.7	58.93
VSShift(IW)	$\mathbf{99.05} \pm 0.33$	$\mathbf{97.38} \pm 1.8$	$\mathbf{84.64} \pm 2.9$	66.96
VSShift(KMM)	$\mathbf{99.05} \pm 0.33$	96.67 ± 2.8	84.46 ± 2.9	**68.75**

TABLE 6.8: Table showing classification accuracy on subject "AA" from data set IVa. Train data means the data used for training, the Indep. Train data means that half of the training data was used for training and the other half for testing, and the Test data means the independent test set provided was used for testing. Uncertainties obtained by using five random shuffles of the data.

Since there was no benefit in applying an RBF kernel over a linear one for the SVM on these BCI data, we only conduct experiments using a linear kernel on data IVa. On the five subjects of this data set, we observe similar trends to the ones discussed above. The results are given in Table 6.8, Table 6.9, Table 6.10, Table 6.11, and Table 6.12. In the last two columns for all five subjects, the VSShift(IW) and VSShift(KMM) are, without fail, the top performing algorithms. This is expected, as we are explicitly assigning higher weights to regions that have higher concentrations of test data than training data. The linear SVM is, however, competitive with the VS on the training data (first two columns). The SVM outperforms the VS on the independent training data four times out of five, which could indicate that the SVM is possibly overfitting the training data, given that it performs much worse on the test data sets. Also worth noting is that out of the eight tables in this chapter (for data I and data IVa), the KMM outperforms the IW in five of them. This is also consistent with Chapter 5, where we recommend using the KMM on smaller data sets, and the IW when larger data sets are available for a better estimate of the densities. Finally in Table 6.13, we compare the results obtained by VSShift in this chapter with the 14 submissions[4] made by researchers during the competition. We can see that our method performs well, with the VSShift(KMM) placing in 8th place, and the IW in 10th place. As on data set I, the difference in algorithmic performance can be largely attributed to the preprocessing performed on the data. For instance, we used a single filter bank for each of the five subjects, whereas Yijun Wang heavily fine-tuned their preprocessing steps for each subject. For subjects *aw* and *ay*, which have the least number of training samples, they went even further by using an adaptive approach to adopt former classified test samples as extended training samples.

[4]http://www.bbci.de/competition/iii/results/index.html#berlin1

	Accuracy(%)			
Algorithm	Train data	Indep. train data	Test data(shuffled)	Test data
VS	**99.64** $\pm$ 0.65	97.50 $\pm$ 2.2	98.21 $\pm$ 1.8	91.07
Linear SVM	99.11 $\pm$ 0.0	98.93 $\pm$ 0.75	97.86 $\pm$ 2.0	91.07
VSShift(IW)	**99.64** $\pm$ 0.65	**99.11** $\pm$ 0.0	**99.64** $\pm$ 0.8	**92.86**
VSShift(KMM)	**99.64** $\pm$ 0.65	98.93 $\pm$ 0.75	99.29 $\pm$ 0.98	**92.86**

TABLE 6.9: Table showing classification accuracy on subject "AL" from data set IVa. Train data means the data used for training, the Indep. Train data means that half of the training data was used for training and the other half for testing, and the Test data means the independent test set provided was used for testing. Uncertainties obtained by using five random shuffles of the data.

	Accuracy(%)			
Algorithm	Train data	Indep. train data	Test data(shuffled)	Test data
VS	**98.10** $\pm$ 0.65	88.10 $\pm$ 4.8	58.16 $\pm$ 3.6	53.06
Linear SVM	94.05 $\pm$ 0.0	85.24 $\pm$ 11	58.06 $\pm$ 5.4	52.04
VSShift(IW)	**98.10** $\pm$ 0.65	**94.29** $\pm$ 4.9	61.12 $\pm$ 2.7	**57.14**
VSShift(KMM)	**98.10** $\pm$ 0.65	92.86 $\pm$ 3.8	**63.78** $\pm$ 4.4	55.61

TABLE 6.10: Table showing classification accuracy on subject "AV" from data set IVa. Train data means the data used for training, the Indep. Train data means that half of the training data was used for training and the other half for testing, and the Test data means the independent test set provided was used for testing. Uncertainties obtained by using five random shuffles of the data.

	Accuracy(%)			
Algorithm	Train data	Indep. train data	Test data(shuffled)	Test data
VS	**100** $\pm$ 0.0	89.29 $\pm$ 6.2	56.43 $\pm$ 3.7	50
Linear SVM	**100** $\pm$ 0.0	**100** $\pm$ 0.0	51.79 $\pm$ 3.1	50.89
VSShift(IW)	**100** $\pm$ 0.0	97.86 $\pm$ 3.2	**62.86** $\pm$ 2.5	**54.02**
VSShift(KMM)	**100** $\pm$ 0.0	97.86 $\pm$ 3.2	62.77 $\pm$ 3.2	53.57

TABLE 6.11: Table showing classification accuracy on subject "AW" from data set IVa. Train data means the data used for training, the Indep. Train data means that half of the training data was used for training and the other half for testing, and the Test data means the independent test set provided was used for testing. Uncertainties obtained by using five random shuffles of the data.

	Accuracy(%)			
Algorithm	Train data	Indep. train data	Test data(shuffled)	Test data
VS	**100** $\pm$ 0.0	75.71 $\pm$ 13	53.81 $\pm$ 3.1	**55.56**
Linear SVM	**100** $\pm$ 0.0	**100** $\pm$ 0.0	50.79 $\pm$ 4.0	48.81
VSShift(IW)	**100** $\pm$ 0.0	95.71 $\pm$ 6.4	59.05 $\pm$ 4.0	48.40
VSShift(KMM)	**100** $\pm$ 0.0	90.00 $\pm$ 11	**60.40** $\pm$ 1.5	**55.56**

TABLE 6.12: Table showing classification accuracy on subject "AY" from data set IVa. Train data means the data used for training, the Indep. Train data means that half of the training data was used for training and the other half for testing, and the Test data means the independent test set provided was used for testing. Uncertainties obtained by using five random shuffles of the data.

			Accuracy(%)					
Rank	Method	Contributor	Overall	aa	al	av	aw	ay
1.	LDA	Yijun Wang	94.17	95.5	100.0	80.6	100.0	97.6
2.	Extended EM	Yuanqing Li	85.12	89.3	98.2	76.5	92.4	80.6
3.	Linear w/ energy penalisation	Liu Yang	83.45	82.1	94.6	70.4	87.5	88.1
4.	LDA	Zhou Zongtan	72.62	83.9	100.0	63.3	50.9	88.1
5.	nu-SVM	Michael Bensch	69.17	73.2	96.4	70.4	79.9	50.8
6.	SVM	Cedric Simon	68.57	83.0	91.1	50.0	87.9	54.4
7.	SVM or DA	Elly Gysels	67.86	69.6	96.4	64.3	69.6	61.9
8.	**VSShift(KMM)**	**	**65.27**	**68.75**	**92.86**	**55.61**	**53.57**	**55.56**
9.	LDA based Kalman filtering	Carmen Viduarre	64.05	66.1	92.9	67.3	68.3	50.4
10.	**VSShift(IW)**	**	**63.88**	**66.96**	**92.86**	**57.14**	**54.02**	**48.40**
11.	N/A	Le Song	63.69	66.1	100.0	63.3	64.3	54.4
12.	Bayesian Classifier	Ehsan Arbabi	62.74	70.5	94.6	56.1	63.8	56.3
13.	Linear SVM	Cyrus Shahabi	59.52	57.1	76.8	57.7	64.3	54.0
14.	Linear SVM	Kiyoung Yang	53.93	52.7	85.7	61.2	51.8	43.7
15.	Linear + non-linear SVM	Wang Feng	51.90	50.9	53.6	54.6	56.2	46.0
16.	Linear SVM	Hyunjin Yoon	51.19	50.0	67.9	52.6	52.7	45.6

TABLE 6.13: Comparison of results of VSShift with the 14 entries on data set IVa (http://www.bbci.de/competition/iii/results/index.html#berlin1). Overall accuracy is 50% for a random classifier.

6.4 Conclusion

In this chapter we compared the performance of our presented techniques from Chapter 4 and Chapter 5, the VS and VSShift respectively, on a completely new problem domain: Brain-Computer Interfaces. Since the data sets are usually very small, it poses a challenge to the weighting schemes used by VSShift, especially the IW. Also, batch algorithms are generally used in the BCI domain, and applying a one-pass technique like VS (and accounting for distribution shift) is a contribution to the BCI field. We used small data sets from the BCI Competition III, and compared our techniques with the state-of-the-art SVM, using both linear and RBF kernels. Since our data samples are small, the KMM generally performs better, as it outperforms the IW four out of six times on data set I (when considering the shuffled test data and the independent test data). Another observation is that the KMM degrades the performance of the classifier when used on data that contains no shift. This is evident in the three tables for data set I in the training data column. These observations are consistent with the findings in Chapter 5 regarding the use of KMM and IW. From our empirical study, we observe that the SVM is always outperformed by a VS-based method, either the standard VS, VSShift, or the VSKernel, which further proves the robustness of the VS method on other application domains. We also contrasted the performance of the VSShift method with the competition submissions, showing that when the properly preprocessed features are passed to VSShift, it is competitive with the best results achieved on the data.

Chapter 7

Conclusions and Future Work

7.1 Summary of work

This thesis deals with the issues that arise when machine learning techniques are applied on very large and shifting data sets. We argue that while algorithms based on subsampling and caching can scale to some extent, they each have their drawbacks which make them impractical for use on tasks such as computer intrusion detection. We therefore propose the use of one-pass algorithms as they see all of the data while still being able to scale well to large scale problems.

First, we discuss clustering techniques and their applicability on large scale data. A common drawback among the vast majority of clustering algorithms is the multiple passes they require through the data. For very large data sets, a novel online clustering method is required that can effectively scale to large problems, while not degrading the quality of the clusters produced. We present the SHPC, which is a novel, fast, hierarchical clustering method that scales well to large data. The idea is to construct an initial tree using any method in the literature using a small random subset of the data, and to sequentially insert the rest of the points where they fit best in the hierarchical tree. Then, by cutting this tree at different levels, we can obtain as many clusters as we want which can be used as-is, or for classification. We demonstrate the effectiveness of this method on bioinformatics tasks, and subsets of the KDD '99 data.

We demonstrate, in Chapter 4, that a one-pass algorithm could be designed to achieve prediction performances competitive with the state-of-the-art machine learning methods. Our novel approach is fast, and can handle very large data with ease; a test performed on $\sim 5 \times 10^6$ training and $311,029$ testing points needed five minutes on average on a standard desktop PC. It works by carving the input space into local regions using hyperspheres, while modeling the density of points in the region of the hypersphere using votes. We initially demonstrated that it achieves good performance on 18 benchmark data sets, which were comparable to the results reported by other authors on the same

problems. We demonstrate this by using the KDD '99 data, which has close to five million points. The VS achieved higher prediction accuracies (in a fraction of the time) than the few published results on this set, since not many authors were able to scale their methods to such large data. Using the publicly available 10% subset of this data, the VS also outperformed the winner of the KDD '99 challenge, along with all the published results reported in the thesis on both the binary and multi-class versions of the data. We then demonstrate the flexibility of our framework by presenting several variants of the VS algorithm, which included VSKernel, VSDistance, VSCluster, VS with adaptive radius, VSMulti with imbalanced data, and VSMulti with distance factor. We conclude the chapter by showing that since VS is a compression scheme, we can bound it's generalisation error on unseen data using risk bounds.

This thesis further considers the issue of shifting distributions, which is a new topic of interest in the community. We performed a Kolmogorov-Smirnov two-sample test to show that the KDD '99 Cup's training and testing distributions do indeed follow different underlying distributions. While the use of the data by other authors simply assumes this to be true, to the best of our knowledge, we are the first to formally confirm this and explicitly account for it in modeling the data. We use the two most recent techniques in the literature to tackle the distribution shift: an importance weighting scheme and kernel mean matching. We evaluate and compare both techniques together, highlighting their advantages and disadvantages, and observe that accounting for the shift using these methods increased the generalisation performances of both our novel non-parametric VS technique, and a parametric Logistic Regression model.

Finally, in Chapter 6, we applied our VS and VSShift methods on the expanding and important field of BCI. Distribution shift is never taken into account in the batch algorithms used in the BCI field, and successfully applying the VSShift on the EEG and ECoG data is a contribution in its own right. We used two data sets from the BCI Competition III, whose distribution shifts were verified using a formal statistical test. We compared our method with the SVM that is generally used in the BCI domain with a linear kernel, and we achieved better results in a fraction of the time, even when the SVM was used with a RBF kernel. Also, since distribution shifts are not taken into account in the BCI domain, we show that explicitly reweighting our training data to match the test data further improved the ability of VS to map the EEG signals to their respective states of mind. This is consistent with the results and findings in Chapter 5, further demonstrating the ability of our VS and its covariate-shift adapted framework to be discriminant and effective in problem areas that do not require fast online algorithms. We also contrasted the performance of the VSShift method with all of the competition submissions for both data sets. On one data set the VSShift was the top performing, and on the other, it outperformed half of the published submissions.

7.2 Future work

There are several directions in which we can proceed with our research.

For the SHPC, we can investigate better ways to set the novelty threshold θ. Currently, θ has one value for all levels of the tree. We aim to investigate ways to automatically set different values of θ at different levels of the tree. Although this is more computationally costly, it could potentially significantly increase the quality of clusters generated by SHPC.

An important issue in machine learning that was not explored in the thesis is outlier detection. After the training phase is complete, and before the testing phase starts, we can have an intermediate step that removes any sphere that is considered an outlier. An example of such a technique is the AdaBoost, described in Section 2.2. There are several straightforward ways this could be done for the VS. One way could involve removing spheres that only have a count of 1. Having a sphere with such a count could indicate that it was created by accident, especially if it is in a region that is dense with data from other classes. Another possible way is to keep the spheres that contain 95% of the total input points, with the spheres representing the remaining 5% discarded. This could potentially increase the training error, but would avoid overfitting on the training data and hence give better generalisation performance. A deep analysis on the effect of this on the VS would be a clear continuation of the discussion started in Section 4.3.1.

Another hidden and important part of VS that was not investigated in this thesis is the use of the training phase as a one-pass clustering technique. Given the speed at which the training phase of the VS completes (can take as little as 1.4 seconds on the $494,020 \times 41$ KDD '99 data), this could be extremely useful for clustering large data sets, such as those described in Chapter 3. A thorough empirical study of how this would compare against standard clustering techniques, and the SHPC, would be a useful addition to the clustering literature. Even if this clustering does not outperform the state-of-the-art, it can still have a place in domains where we can afford to trade off some cluster quality for speed. Indeed, in some cases such as bioinformatics and intrusion detection, the use of traditional clustering methods such as k-means is already infeasible today, and other techniques that will trade off cluster quality for the ability to scale will be the only way forward. This study can also be taken further, in that the training phase of VS can be used as a preprocessing step before applying other state-of-the-art techniques like SVMs, in the same spirit as Yu et al. (2003). The hypersphere centres can be used as the training points to the SVM, reducing the training set size from millions to several thousands or possibly hundreds. A more sophisticated way could be to use the counts of the hyperspheres as well, treating them like weights in the classifier they are passed to.

Novelty detection is an important issue in machine learning, especially in fields where

abnormal data are hard to obtain. We aim to investigate the VS framework's ability to be used as a novelty detector. This means that the VS only takes points from one class, which is considered 'normal', and fits hyperspheres around them without the need to store any counts. Later, during testing, input points are checked to see whether they fit into any hypersphere, and if they do not they are considered anomalous. Current novelty detectors, such as the Parzen windows, are computationally intensive, as they attempt to estimate the data's density. Investigating how VS compares to other anomaly detectors in the literature is an important future direction to pursue.

The VS algorithm carves up the input space into overlapping hyperspheres, and has mainly used Euclidean distances to determine if data points fall into these. The approach thus does not take into account local correlations in the data. We aim to address this in the future by storing local covariance matrices and computing Mahalanobis distances instead. The difference between such an approach and the discriminant approach of one-pass training of a multilayer Perceptron, the Sequential Input Space Partitioning (SISP) algorithm by Shadafan and Niranjan (1994), remains to be explored.

Finally, we aim to extend the idea of using VSShift to include time-varying data, as opposed to covariate shift only. There is a lack of such techniques in the literature, as was mentioned by Cavallanti et al. (2006). Therefore, in a similar fashion to what was done by Cavallanti et al. (2006), we can use a decay parameter to downweight older spheres as training progresses. Naturally, we can compare this method with the algorithm proposed in the aforementioned paper, and with the OCBudget with which they compare their algorithm. Should this be successful, the VS framework would deal with both time-varying data and covariate shift, which affect many real world problem tasks.

Bibliography

E. Achtert, C. Bohm, H.-P. Kriegel, and P. Kroger. Online hierarchical clustering in a data warehouse environment. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 10–17. IEEE Computer Society, 2005.

S. Asharaf, S. K. Shevade, and M. N. Murty. Scalable non-linear support vector machine using hierarchical clustering. In *Proceedings of the 18th International Conference on Pattern Recognition*, pages 908–911, 2006.

R. Beghdad. Critical study of neural networks in detecting intrusions. *Computers and Security*, 27(5–6):168 – 175, 2008.

B. Blankertz, G. Curio, and K. R. Müller. Classifying single trial EEG: towards brain computer interfacing. In *Advances in Neural Information Processing Systems 14*, pages 157–164, 2002.

B. Blankertz, K.-R. Müller, D. Krusienski, G. Schalk, J. R. Wolpaw, A. Schlögl, G. Pfurtscheller, J. Millan, M. Schröder, and N. Birbaumer. The BCI2000 Competition III: validating alternative approaches to actual BCI problems. *IEEE Transactions on Neural Systems and Rehabilitation Engineering.*, 14:153–159, 2006.

B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, volume 5, pages 144–152, 1992.

J. Bradley and R. E. Schapire. In *FilterBoost: Regression and classification on large datsets*. 2008.

T. Brugger. Kdd cup '99 dataset considered harmful. Position Letter, KD Nuggets newsletter, 07(18), 2007.

M. J. Brusco and H. F. Köhn. Comment on "Clustering by passing messages between data points". *Science*, 319:726c, 2008.

S. Canu and G. Loosli. Comments on the "Core Vector Machines: Fast SVM training on very large datasets". Technical report, National Institute of Applied Sciences –INSA, Lyon, France, 2006.

G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Advances in Neural Information Processing Systems 13*, 2000.

G. Cavallanti, N. Cesa-Bianchi, and C. Gentile. Tracking the Best Hyperplane with a Simple Budget Perceptron. *Machine Learning*, 2/3:143–167, 2006.

G. C. Cawley and N. L. C. Talbot. Efficient leave-one-out cross-validation of kernel Fisher discriminant classifiers. *Pattern Recognition*, 36(11):2585–2592, 2003.

C. C. Chang and C. J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

R. Chinchani, A. Muthukrishnan, M. Chandrasekaran, and S. Upadhyaya. Racoon: Rapidly generating user command data for anomaly detection from customizable templates. *Computer Security Applications Conference, Annual*, 0:189–204, 2004.

L. Lo Conte, B. Ailey, T. J. P. Hubbard, S. E. Brenner, A. G. Murzin, and C. Chothia. SCOP: a Structural Classification of Proteins database. *Nucleic Acids Research*, 28 (1):257–259, 2000.

G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize oat: An analytic study of exact and approximation algorithms. *Management Science*, 23:789–810, 1977.

C. Cortes and V. Vapnik. Support-Vector Networks. In *Machine Learning*, pages 273–297, 1995.

K. Crammer, J. Kandola, and Y. Singer. Online classification on a budget. In *Advances in Neural Information Processing Systems 16*, 2003.

K. Crammer and Y. Singer. Ultraconservative online algorithms for multi-class problems. *Journal of Machine Learning Research*, 3:951–991, 2003.

O. Dekel, S. Shalev-Shwartz, and Y. Singer. The Forgetron: a kernel-based perceptron on a fixed budget. *Advances in Neural Information Processing Systems 18*, pages 259–266, 2006.

T. G. Diettrich. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, 40 (2):139–157, 2000.

C. Ding and I. Dubchak. Multi-class protein fold recognition using Support Vector Machines and neural networks. *Bioinformatics*, 17:349–358, 2001.

G. Dornhege, J. del R. Millan, T. Hinterberger, D. J. McFarland, and K.-R. Müller. *Toward Brain-Computer Interfacing*. MIT Press, Cambridge, 2007.

M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences of the United States of America*, 95(25):14863–14868, 1998.

Y. El-Sonbaty and M.A. Ismail. On-line hierarchical clustering. *Pattern Recogn. Lett.*, 19(14):1285–1291, 1998.

C. Elkan. Results of the KDD'99 classifier learning. *SIGKDD Explor. Newsl.*, 1(2): 63–64, 2000.

L. Ertoz, M. Steinbach, and V. Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of SIAM International Conference on Data Mining*, 2003.

E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabelled data. In *Applications of Data Mining in Computer Security*. Kluwer, 2002.

A. M. Fahim, A. M. Saleh, F. A. Torkey, M. A. Ramadan, and G. Saake. An efficient K-Means with good initial starting points. *Georgian Electronic Scientific Journal: Computer Science and Telecommunications*, 2(19):47–57, 2009.

D. Fasulo. An analisys of recent work on clustering algortihms. Technical report, 1999.

Y. Freund and R. Schapire. Large margin classification using the Perceptron algorithm. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 209–217, 1998.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of online learning and an application to boosting. In *EuroCOLT: European Conference on Computational Learning Theory*, 1994.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of 13th International Conference on Machine Learning*, 1996.

Y. Freund and R. E. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artifical Intelligence*, 14(5):771–780, 1999.

B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Technical report, Dept. of Statistics – Stanford University, 1998.

M. Fugate and J. R. Gattiker. Anomaly detection enhanced classification in computer intrusion detection. In *SVM '02: Proceedings of the First International Workshop on Pattern Recognition with Support Vector Machines*, pages 186–197. Springer-Verlag, 2002.

G. Fung. A comprehensive overview of basic clustering algorithms. Department of Computer Sciences, University of WisconsinMadison.

C. Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242, 2001.

G. Giacinto, R. Perdisci, M. Del Rio, and F. Roli. Intrusion detection in computer networks by a modular ensemble of one-class classifiers. *Information Fusion*, 9(1):69 – 82, 2008. Special Issue on Applications of Ensemble Methods.

A. Gretton, A. Smola, J. Huang, M. Schmittfull, K. Borgwardt, and B. Schölkopf. Covariate Shift by Kernel Mean Matching. In A. Schwaighofer J. Quiñonero Candela, M. Sugiyama and N. D. Lawrence, editors, *Dataset Shift in Machine Learning*, pages 131–160. MIT Press, 2009.

M. Hasan and J. Jue. Online clustering for hierarchical WDM networks. In *IEEE/OSA Conference on Optical Fiber Communication*, pages 1–3, 2008.

K. Hattori and M. Takahashi. A new edited k-nearest neighbor rule in the pattern classification problem. *Pattern Recognition*, 33(3):521–528, 2000.

W. Hu, W. Hu, and S. Maybank. Adaboost-based algorithm for network intrusion detection. *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, 38(2):577–583, 2008.

T. Joachims. Making large-scale svm learning practical. In A. Smola B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, Cambridge, MA, 1999.

V. Kadirkamanathan and M. Niranjan. A function estimation approach to sequential learning with neural networks. *Neural Computation*, 5(6):954–975, 1993.

T. Kanamori and H. Shimodaira. Geometry of Covariate Shift with Applications to Active Learning. In A. Schwaighofer J. Quiñonero Candela, M. Sugiyama and N. D. Lawrence, editors, *Dataset Shift in Machine Learning*, pages 87–105. MIT Press, 2009.

N. Kaplan, M. Friedlich, M. Fromer, and M. Linial. A functional hierarchical organization of the protein sequence space. *BMC Bioinformatics*, 5:196, 2004.

V. Katos. Network intrusion detection: Evaluating cluster, discriminant, and logit analysis. *Information Sciences*, 177(15):3060 – 3073, 2007.

G. H. Kayacik, N. A. Zincir-Heywood, and M. I. Heywood. A hierarchical som-based intrusion detection system. *Engineering Applications of Artificial Intelligence*, 20(4): 439–451, 2007.

S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murty. A fast iterative nearest point algorithm for support vector machine classifier design. Technical report, Indian Institute of Science, Bangalore, 1999.

D.S. Kim and J. S. Park. Network-based intrusion detection with Support Vector Machines. *Lecture Notes in Computer Science*, 2662:747–756, 2003.

Z. J. Koles, M. S. Lazar, and S. Z. Zhou. Spatial patterns underlying population differences in the background EEG. *Brain Topography*, 2(4):275–284, 1990.

T. N. Lal, T. Hinterberger, G. Widman, C. E. Elger, B. Schölkopf, and N. Birbaumer. Methods Towards Invasive Human Brain Computer Interfaces. In *Advances in Neural Information Processing Systems 17*, pages 737–744, 2005.

P. Laskov, K. Rieck, C. Schfer, and K. Müller. Visualization of anomaly detection using prediction sensitivity. In *Sicherheit*, pages 197–208, 2005.

F. Laviolette, M. March, and M. Shah. Margin-sparsity trade-off for the set covering machine. In *Proceedings of the 16th European Conference on Machine Learning (ECML 2005)*, Lecture Notes in Artificial Intelligence, 2005.

B. Lazareva-Ulitsky, K. Diemer, and P. D. Thomas. On the quality of tree-based protein classification. *Bioinformatics*, 21(9):1876–1890, 2005.

Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and K. Müller, editors, *Neural Networks: Tricks of the trade*, pages 5–50. Springer, 1998.

F. Lee, R. Scherer, R. Leeb, C. Neuper, H. Bischof, and G. Pfurtscheller. A comparative analysis of multi-class EEG classification for brain computer interface. In *Proceedings of the 10th Computer Vision Winter Workshop (CVWW)*, 2005.

J. Lee, D. S. Kim, S. Chi, and J. S. Park. Using the Support Vector Machine to detect the host-based intrusion. In *IRC International Conference.*, 2002.

I. Levin. KDD-99 classifier learning contest LLSoft's Results Overview. *ACM SIGKDD Explorations*, 1(2):67–75, 2000.

Y. Li and P. M. Long. The relaxed online maximum margin algorithm. *Machine Learning*, 46(1–3):361–387, 2002.

Y. G. Liu, K. F. Chen, X. F. Liao, and W. Zhang. A genetic clustering method for intrusion detection. *Pattern Recognition*, 37(5):927–942, 2004.

Y. Loewenstein, E. Portugaly, M. Fromer, and M. Linial. Efficient algorithms for accurate hierarchical clustering of huge datasets: tackling the entire protein space. In *ISMB*, pages 41–49, 2008.

F. Lotte, M. Congedo, A. Lecuyer, and B. Arnalsi. A review of classification algorithms for EEG-based brain-computer interfaces. *Journal of Neural Engineering*, 4:R1–R13, 2007.

J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying suspicious URLs: an application of large-scale online learning. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 681–688, 2009.

O. Mangasarian and D. R. Musicant. Active Support Vector Machine Classification. Technical Report 00–04, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, April 2000. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-04.ps.

J. G. Marin-Blazquez and G. M. Perez. Intrusion detection using a linguistic hedged fuzzy-xcs classifier system. *Soft Computing*, 13(3):273–290, 2008. ISSN 1432-7643.

J. McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, 2000.

R. Meir and G. Rätsch. An introduction to boosting and leveraging. In *Advanced Lectures on Machine Learning*, 2003.

S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K. Müller. Fisher Discriminant Analysis with kernels. In *Proceedings of IEEE Neural Networks for Signal Processing Workshop*, pages 41–48, 1999.

N. Mladenovic, J. Brimberg, P. Hansen, and J. Moreno-Perez. The p-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*, 179:927–939, 2007.

C. Molina and M. Niranjan. Pruning with replacement on limited resource allocating networks by f-projections. *Neural Computation.*, 8(4):855–868, 1996.

K.-R. Müller, M. Krauledar, G. Dornhege G. Curio, and B. Blankertz. Machine learning techniques for brain-computer interfaces. *Biomedical Technology*, 49:11–22, 2004.

N. Murata. *A statistical study on the asymptotic theory of learning.* PhD thesis, University of Tokyo, 1992.

N. Murata, M. Kawanabe, A. Ziehe, K. Müller, and S. Amari. On-line learning in changing environments with applications in supervised and unsupervised learning. *Neural Networks.*, 15(4):743–760, 2002.

A. Naish-Guzman and S. Holden. The Generalized FITC approximation. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1057–1064. 2008.

S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

F. Orabona, J. Keshet, and B. Caputo. The Projectron: a bounded kernel-based Perceptron. In *ICML '08: Proceedings of the 25th International Conference on Machine Learning*, pages 720–727, 2008.

E. Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

G. E. Peterson and H. L. Barney. Control methods used in the study of vowels. *J. Acoust. Soc. Amer.*, 24:75–184, 1952.

B. Pfahringer. Winning the KDD99 classification cup: bagged boosting. *ACM SIGKDD Explorations*, 1(2):65–66, 2000.

J. Platt. A resource-allocating network for function interpolation. *Neural Computation.*, 3(2):213–225, 1991.

J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods–Support Vector Learning*, pages 185–208, 1999.

R. W. Prager and F. Fallside. The modified kanerva model for automatic speech recognition. *Computer Speech and Language*, 3(1):61 – 81, 1989.

W. Quigguo, M. Fei, W. Yijun, G. Xiaorong, and G. Shangkai. Feature combination for classifying single-trial ECoG during motor imagery of different sessions. *Progress in Natural Science*, 17(7):851–858, 2007.

J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence. *Dataset Shift in Machine Learning*. The MIT Press, 2009.

A. Rakotomamonjy, V. Guigue, G. Mallet, and V. Alvarado. Ensemble of SVMs for improving Brain Computer Interface P300 Speller Performances. In *ICANN (1)*, pages 45–50, 2005.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. *Machine Learning*, 42(3):287–320, 2001.

K. Rieck and P. Laskov. Language models for detection of unknown attacks in network traffic. *Journal in Computer Virology*, 2(4):243–256, 2007.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407, 1958. Reprinted in Neurocomputing (MIT Press, 1988).

F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Spartan, New York, 1962.

A. H. Sung. S. Mukkamala, G. Janoski. Intrusion detection using neural networks and Support Vector Machines. In *Proc. of IEEE Int. Joint Conf. on Neural Networks*, 2002.

M. R. Sabhani and G. Serpen. Application of machine learning algorithms to KDD intrusion detection dataset within misuse detection context. In *Proceedings of International Conference on Machine Learning: Models, Technologies, and Applications*, pages 209–215, 2003.

S. T. Sarasamma, Q. A. Zhu, and J. Huff. Hierarchical Kohonenen net for anomaly detection in network security. *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, 335(2):302–312, 2005.

R. E. Schapire. A brief introduction to boosting. In *IJCAI*, pages 1401–1406, 1999.

R. S. Shadafan and M. Niranjan. A dynamic neural network architecture by sequential partitioning of the input space. *Neural Computation*, 6(6):1202–1222, 1994.

R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28:1409–1438, 1958.

D. Song, M. I. Heywood, and A. N. Zincir-Heywood. Training genetic programming on half a million patterns: An example from anomaly detection. *IEEE Transactions on Evolutionary Computation*, 9(3):225–239, 2005.

T. T. Tanimoto. An elementary mathematical theory of classification and prediction. IBM Internal Report, 1958.

M. B. Teitz and P. Bart. Heuristic Methods for Estimating Generalized Vertex Median of a Weighted Graph. *Operations Research*, 16:955–961, 1968.

K. P. Thomas, C. Guan, C. T. Lau, A. P. Vinod, and K. K. Ang. A new discriminative common spatial pattern method for motor imagery brain-computer interfaces. *IEEE Transactions on Biomedical Engineering*, 56(11 Pt 2):2730–2733, 2009.

M. E. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

M. W. B. Trotter. *Support vector machines for drug discovery*. PhD thesis, University College London, 2006.

I. Tsang, J. Kwok, and P. M. Cheung. Core vector machines: Fast SVM training on very large datasets. *Journal of Machine Learning Reseach*, 6:363–392, 2005.

S. Tuna and M. Niranjan. Cross-platform analysis with binarized gene expression data. In *PRIB*, pages 439–449, 2009.

S. Tuna and M. Niranjan. Inference from low precision transcriptome data representation. *Signal Processing Systems*, 58(3):267–279, 2010.

P. C. van Oorschot, J.-M. Robert, and M. V. Martin. A monitoring system for detecting repeated packets with applications to computer worms. *International Journal of Information Security*, 5(3):186–199, 2006.

V. Vapnik. *Statistical Learning Theory*. Wiley Inter–Science, 1998.

S. V. N. Vishwanathan, A.J. Smola, and M. N. Murty. SimpleSVM. In *Proceedings of the 20th International Conference on Machine Learning*, pages 760–767, 2003.

G. Wang, J. Hao, J. Ma, and L. Huang. A new approach to intrusion detection using artificial neural networks and fuzzy clustering. *Expert Systems with Applications*, In Press, Corrected Proof:–, 2010.

M. S. Waterman and T. F. Smith. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

E. Welzl. Smallest enclosing disks (balls and ellipsoids). In *New results and new trends in computer science*, pages 359–370. Springer-Verlag, 1991.

J. Weston, A. Bordes, and L. Bottou. Online (and Offline) on an even tighter budget. In *International Workshop on Artificial Intelligence and Statistics*, 2005.

J. Wickramaratna, S. Holden, and B. Buxton. Performance degradation in boosting. In *MCS '01: Proceedings of the Second International Workshop on Multiple Classifier Systems*, pages 11–21, 2001.

D. L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, 2:408–421, 1972.

J. R. Wolpaw, N. Birbaumer, W. J. Heetderks, D. J. McFarland, P. H. Peckham, G. Schalk, E. Donchin, L. A. Quatrano C. J. Robinson, and T. M. Vaughan. Brain-computer interface technology: a review of the first international meeting. *IEEE Transactions on Rehabilitation Engineering*, 8:164–173, 2000.

C. H. Wu, R. Apweiler, A. Bairoch, D. A. Natale, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, R. Mazumder, C. O'Donovan, N. Redaschi, and B. Suzek. The Universal Protein resource (UniProt): an expanding universe of protein information. *Nucleic Acids Research*, 34(Database issue):D187–D191, 2006.

D. Yeung and C. Chow. Parzen window network intrusion detectors. In *Proceedings of the International Conference on Pattern Recognition*, 2002.

H. Yu, J. Yang, and J. Han. Classifying large data sets using SVMs with hierarchical clusters. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

Z. Yu and J. Tsai. A multi-class SLIPPER system for intrusion detection. In *Proceedings of the 28th IEEE Annual International Computer Software and Applications Conference (COMPSAC04)*, pages 212–217, 2004.

Z. Yu, J. Tsai, and T. Weigert. Automatically tuning intrusion detection system. *IEEE Transactions on Systems, Man, and Cybernetics*, 37(2):373–384, 2007.

Z. Yu, J. Tsai, and T. Weigert. An adaptive automatically tuning intrusion detection system. *ACM Trans. Auton. Adapt. Syst.*, 3(3):1–25, 2008.

J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study. *IJCV*, 73(2): 213–238, 2007.

Y. Zhao, G. Karypis, and U. Fayyad. Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery*, 10(2):141–168, 2005.