2010 IEEE International Workshop on Machine Learning for Signal Processing (MLSP 2010)
August 29 – September 1, 2010, Kittilä, Finland

MACHINE LEARNING FOR INTRUSION DETECTION: MODELING THE DISTRIBUTION SHIFT

Bassam Farran, Craig Saunders, and Mahesan Niranjan

School of Electronics & Computer Science, University of Southampton, Southampton, UK
bf06r@ecs.soton.ac.uk, Craig.Saunders@xrce.xerox.com, mn@ecs.soton.ac.uk

ABSTRACT

This paper addresses two important issue that arise in formulating and solving computer intrusion detection as a machine learning problem, a topic that has attracted considerable attention in recent years including a community-wide competition using a common data set known as the KDD Cup '99[1]. The first of these problems we address is the size of the data set, 5×10^6 by 41 features, which makes conventional learning algorithms impractical. In previous work, we introduced a one-pass non-parametric classification technique called Voted Spheres, which carves up the input space into a series of overlapping hyperspheres. Training data seen within each hypersphere is used in a voting scheme during testing on unseen data. Secondly, we address the problem of distribution shift whereby the training and test data may be drawn from slightly different probability densities, while the conditional densities of class membership for a given datum remains the same. We adopt two recent techniques from the literature, density weighting and kernel mean matching, to enhance the Voted Spheres technique to deal with such distribution disparities. We demonstrate that substantial performance gains can be achieved using these techniques on the KDD cup data set.

1. INTRODUCTION

A computer intrusion is an incident of unauthorised access to data or an automated information system[2]. Intrusion Detection Systems (IDS) began in the 1980s as a promising paradigm for detecting and preventing such attacks [1]. Over the past two decades, two major types of IDS have been developed: Anomaly-based and signature-based. The former models 'normal' behaviour, and flags any behaviour that is considered deviant as an intrusion. The latter models specific attack types, and can only detect intrusions for which it has seen previous data. However, as time goes by and technology improves, developing efficient IDS becomes an increasingly difficult task. One major concern is the amount of data generated by modern networks, which is in the order of gigabytes per minute. Another is that normal behaviour changes with time, and intruders constantly look for ways to bypass IDS, an issue we refer to as distribution shift.

A good example of a very large data set containing a distribution shift is the KDD '99 cup data. 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. Now, this data set is considered the de facto for intrusion detection [2, 3, 4], 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, with 41 dimensions), contains a distribution shift between training and test sets, has new intrusions in the test set not present in the training set, the results of the competition's winners are available for comparison purposes, and it is probably the best intrusion detection data set among the very few publicly available ones.

In this paper, we aim to tackle the issue of learning under the covariate shift. We first present a novel one-pass, non-linear, multi-class classifier called Voted Spheres. We then present two techniques to tackle distribution shift: A likelihood weighting scheme and the kernel mean matching. We incorporate these into both our VS framework and a Logistic Regression, and demonstrate that explicitly accounting for distribution shift increases the performance of parametric and non-parametric methods alike.

2. VOTED SPHERES

Farran and Saunders [5] present a fast one-pass algorithm for learning on large data sets called Voted Spheres (VS). The algorithm combines the idea of voting [6] with hypersphere fitting. VS is a constructive algorithm to carve up the space into overlapping hyperspheres, which allows training to progress much faster as no extra effort is needed to explicitly disallow overlapping. Each hypersphere also has a count c associated with it, reflecting the number of training points that fall in that sphere.

During training, when an input point $\langle \mathbf{x}_t, y_t \rangle$ where $\mathbf{x}_t \in$

[1]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[2]Dictionary of Military and Associated Terms. US Department of Defense 2009

$\mathbb{R}^d$ and $y \in \mathcal{Y}$ arrives at time t, the set S of hyperspheres (that belong to y_t) that $\mathbf{x}_t$ falls within are retrieved. If S is empty, then a new hypersphere is created centred around $\mathbf{x}_t$, and the count c of the new hypersphere is set to 1. On the other hand, if S is not empty, then the counts of all hyperspheres in S are incremented by 1, and the input point $\mathbf{x}_t$ is discarded. At the end of the training phase, the algorithm will have a set of hyperspheres, along with their associated weights. This set is a small fraction of the overall data set, and along with the weights, summarises the data neatly. Each hypersphere can then be viewed as a direct estimate of the density of training points within its immediate vicinity. This data compression greatly speeds up the testing phase, as the algorithm only needs to loop over a tiny fraction of the overall data.

During testing, when a new point arrives, the set S of all hyperspheres that this point falls within is retrieved (using the same radius as in the training phase). Since the hyperspheres in S could potentially belong to different classes, the new point is classified as the class that has the highest total sum of weights in S. However, if S is empty, then the closest hypersphere from each class is retrieved, and the testing point is classified using the class of the hypersphere with the largest weight. If in the rare case that these largest weights are equal, the test point is classified using the 1-NN rule. This is shown in Algorithm 1, and flowcharts describing the training and testing phases are given in Figure 1(a) and Figure 1(b) respectively.

Algorithm 1 Pseudo-code for the Voted Spheres by Farran and Saunders [5]

```
Input: labeled training set $\langle(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\rangle$, $R_y \in \mathbb{R}$, a radius for each distinct class $y \in \mathcal{Y}$
Output: a set of pairs of sphere centers and associated weights, the predictions on the test data $\hat{\mathbf{y}}$

    Training:
 1: while Input point $\mathbf{x}$ do
 2:     Retrieve set $S$ of all hyperspheres $\mathbf{x}$ falls within
 3:     if $S$ is empty then
 4:         Create new hypersphere centred around $\mathbf{x}$, initialise count to 1.
 5:     else
 6:         Increment count of each hypersphere in $S$ by 1.
 7:     end if
 8: end while

    Testing:
 9: while test point $\hat{\mathbf{x}}$ do
10:     Retrieve $S$ containing all hyperspheres $\hat{\mathbf{x}}$ falls within
11:     if S is not empty then
12:         Sum counts of hyperspheres from each class, and make prediction
13:     else
14:         Retrieve closest hypersphere from each class
15:         if Top two weights equal then
16:             Make 1-NN prediction
17:         else
18:             Predict using highest vote
19:         end if
20:     end if
21: end while
```

3. SHIFTING DISTRIBUTIONS

3.1. Importance Weighting

Kanamori et al. [7] 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 scheme 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 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 4, is that plenty 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.

3.2. Kernel Mean Matching

Another technique to reweight the training data to better match the test data is Kernel Mean Matching (KMM)[8]. The KMM works by attempting to match the means of the input (covariate) distributions in a kernel induced feature space by solving a quadratic program. This reweighted training data will more closely resemble the test data, which is what we would like our learning algorithm to generalise on.

More formally, following [8], 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

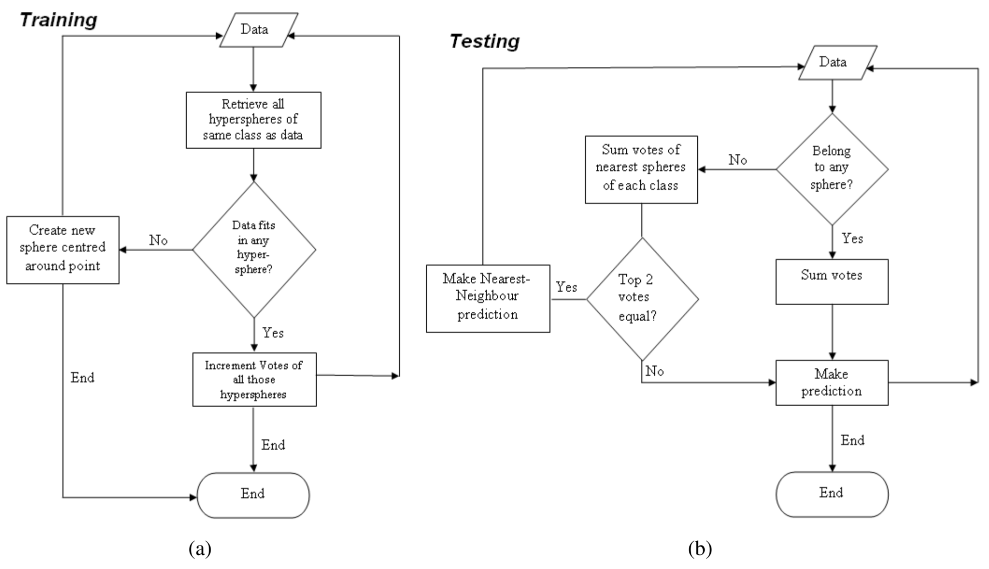

Training
Data
Retrieve all hyperspheres of same class as data
Data fits in any hyper-sphere?
No
Create new sphere centred around point
Yes
Increment Votes of all those hyperspheres
End
End
End
(a)
Testing
Data
Belong to any sphere?
No
Sum votes of nearest spheres of each class
Yes
Sum votes
Top 2 votes equal?
Yes
Make Nearest-Neighbour prediction
No
Make prediction
End
End
(b)

Fig. 1. Flowcharts showing the training (a) and testing (b) phases of the Voted Spheres approach. The training phase carves the input space into a set of overlapping hyperspheres and determines the number of votes each sphere will contribute to a test example. When a test points does not lie in any hypersphere, the classifier defaults to a 1-NN decision rule.

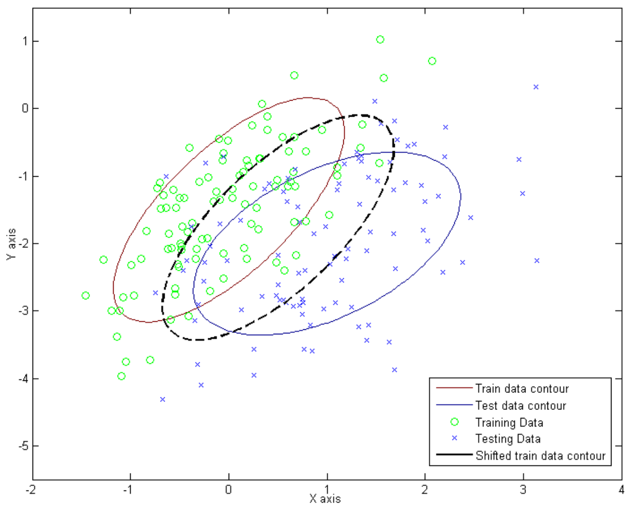


Fig. 2. Illustration of KMM on an artificial data set. The training and test distributions (bold ellipses) differ. The ellipse in dashed line is the mean-matched distribution of the test set, computed using an RBF kernel and $B = 0.5$.

$\left|\frac{1}{n_{\text{tr}}}\sum_{i=1}^{n_{tr}} \beta_i - 1\right| \leq \epsilon$. 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 with an rbf kernel to a toy problem (shown in Figure 2) 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 (using $B = 0.5$, selected using five-fold cross validation) 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 which is shown with a dashed line. The figure clearly shows how the training distribution has shifted towards the testing one in the input space. Unlike the IW, there is no density estimation involved, which means that not much data is needed for this algorithm to work.

R_{-1}	R_1	VS Accuracy (%)	VSShift(IW) Acc(%)
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	0.9	93.55	**94.14**
2.3	1.0	93.64	**94.32**
2.4	0.9	93.48	**94.13**
2.4	1.0	93.59	**94.31**
2.5	0.9	93.33	**94.09**
2.5	1.0	93.31	**94.28**
2.6	1.0	93.28	**94.26**
2.7	1.0	93.30	**94.23**
2.8	1.0	93.36	**94.16**
2.9	1.0	92.89	**94.07**

Table 1. Prediction accuracy of VS and VSShift for different values of the radii in the vicinity of the optimal radii achieved via 5-fold cross-validation. 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.

4. EXPERIMENTS

Initially we experimented with a 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. To incorporate the importance weighting technique into the LR training, we change the update step from:

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

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 report performances on three subsets of the KDD '99 data:

- the training set from which parameters were estimated;
- An unseen hold-out part of the training set;
- A subset of the test set supplied with the KDD data set.

The second and third in the above three are on data not seen by the training algorithm. The second, however, is data that has the same probability density as the training set, while the third is data that has suffered a distribution shift, the main concern we address. We used 20,000 points for each of the three partitions above, sampling a balanced set of 10,000 normal connections and an equal number from the intrusion class. This is to illustrate the benefit arising from the two different weighting techniques, and to show the effect of the weighting on our algorithm's performance with respect to the sample size. 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. Also, several runs were performed to avoid local minima during training, and the results are shown in Table 2. 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 latter was sampled has shifted from the distribution on which the algorithm was trained. It is quite clear that accounting for distribution shift significantly improved the logistic regression.

Now, we adapt the ratio weighting method to the non-parametric VS model. In the original VS algorithm, all spheres 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, instead of updating the counts of hyperspheres and initialising the hypersphere counts to 1 (in Step 4 and Step 6 of Algorithm 1), we update them by $p_{te}(\mathbf{x}_t)/p_{tr}(\mathbf{x}_t)$.

By incrementing the counts of spheres in the new fashion, we force spheres 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).

	Data 1 (%)	Data 2(%)	Data 3(%)
VS	**94.59 ± 0.1**	94.35 ± 0.1	93.08 ± 0.4
VSShift(IW)	**94.59 ± 0.1**	**95.82 ± 0.1**	**94.30 ± 0.4**
LR	88.13 ± 7.0	78.51 ± 24	56.60 ± 15
WLR	88.13 ± 7.4	85.36 ± 13	80.07 ± 0.5

Table 2. Classification accuracies of VS, VSShift(IW), LR, and weighted LR on 3 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: R_{-1} = 2.9, and R_1 = 1.

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. For our experiments on the KDD '99 cup, we took the latter approach as the densities of the train and test sets are not known. To achieve this, we first partition each of the features (41 in the KDD '99 case) of the training and test sets (separately) into $n = 4$ bins, and increment the values of these bins depending on the feature value, exactly as we would for a histogram. 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 $\mathbf{x}_i$ arrives, we calculate its probability density (in either the training or test set) by multiplying the probabilities of each feature value ($\mathbf{x}_{i,j}$) occurring together. Furthermore, as discussed in [5], the radii of the VS algorithm were chosen using cross-validation. To highlight the improvement in results achieved by weighting the spheres, the VS and VSShift(IW) algorithms were run for different values of the radii on subsamples of size 20,000 (drawn from the KDD '99 data), and the results presented in Table 1. It is clear that the VSShift(IW) algorithm outperforms VS every time. In Table 2, as we expected, we observe that the shift in the data causes the algorithms' performance levels to drop. 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.

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 use 3 samples of size 400 for testing:

- 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.

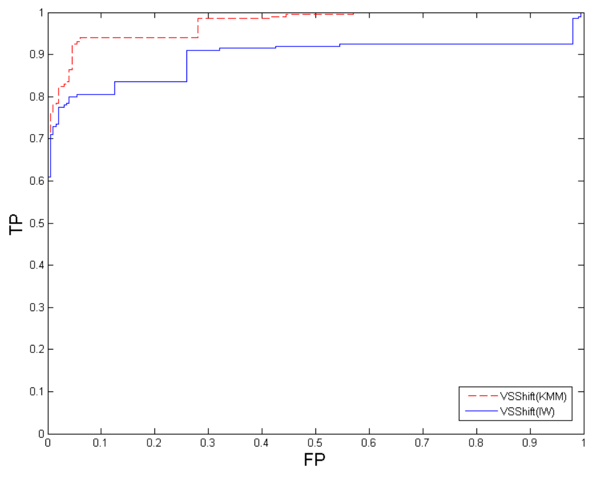
In a similar fashion to the way we incorporated the IW to the VS framework, we modify the VS to incorporate the KMM by updating counts of hyperspheres and initialising new ones to β_i instead of 1, where β_i is the weight obtained from the KMM optimisation (in Section 3) for training point $\mathbf{x}_i$.

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 3 are a direct comparison between the KMM and the IW. As can be seen from the results in the table, the KMM outperforms both the VS and the VSShift(IW). 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. A plot showing ROC curves for VSShift(IW) and VSShift(KMM) is given in Figure 3(a).

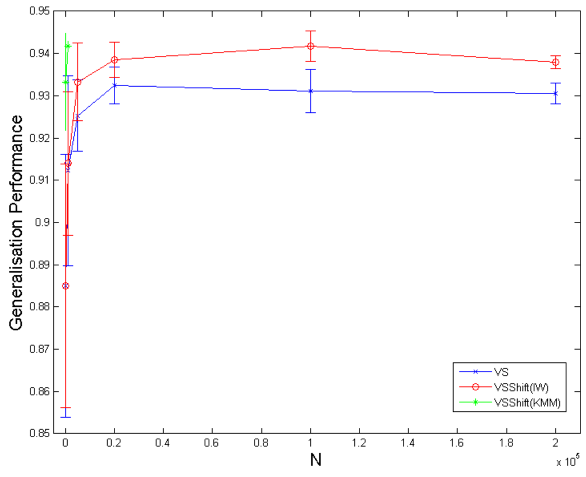
Figure 3(b) shows the trends observed in the paper, where the VS is the base algorithm. We observe that the IW clearly improves the overall generalisation accuracy of VS. As mentioned earlier, the IW needs a lot of data to estimate the input densities accurately, and this is clearly visible in the figure; the larger the sample size, the better the improvement to VS (and the smaller the standard deviation). However when the data set is small, the KMM outperforms the VS and the VSShift(IW). Figure 3(b) 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. CONCLUSION

We have shown that two techniques for dealing with disparities between training and test data sets can be adopted to enhance a non-parametric one-pass pattern classification approach. On the benchmark data set for intrusion detection, we found significant improvements by using the density weighting approach. The kernel mean matching was



(a)



(b)

Fig. 3. (a) 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. (b) Performance as a function of sample size for VS, VSShift(IW) and VSShift(KMM) on subsets of the KDD '99 data. Five random subsamples of each N were used to obtain the generalisation performance and uncertainties.

	Data 1(%)	Data 2(%)	Data 3(%)
VS	$\mathbf{95.8 \pm 0.0}$	92.4 ± 0.3	92.9 ± 0.4
VSShift (IW)	$\mathbf{95.8 \pm 0.0}$	87.5 ± 0.2	90.9 ± 0.9
VSShift (KMM)	95.1 ± 0.2	$\mathbf{92.8 \pm 0.5}$	$\mathbf{93.8 \pm 0.2}$

Table 3. 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.

also shown to improve performance, but because it involves quadratic programming, cannot be applied to very large data sets.

The VS algorithm carves up the input space into overlapping hyperspheres and uses Euclidean distances to determine if a new data falls into these. The approach thus does not take into account local correlations in the data. Our future work will address this 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 [9] remains to be explored.

6. REFERENCES

[1] D.E. Denning, "An intrusion-detection model," *IEEE Trans. Software Engineering*, vol. 13, no. 2, pp. 222–232, 1987.

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

[3] P. Laskov, K. Rieck, C. Schfer, and K. Müller, "Visualization of anomaly detection using prediction sensitivity," in *Sicherheit*, 2005, pp. 197–208.

[4] V. Katos, "Network intrusion detection: Evaluating cluster, discriminant, and logit analysis," *Information Sciences*, vol. 177, no. 15, pp. 3060 – 3073, 2007.

[5] B. Farran and C. Saunders, "Voted spheres: An online, fast approach to large scale learning," in *Proceedings of WAINA*, 2009, pp. 744–749.

[6] Y. Freund and R. Schapire, "Large margin classification using the Perceptron algorithm," in *Proceedings of the eleventh annual conference on Computational learning theory*, 1998, pp. 209–217.

[7] T. Kanamori and H. Shimodaira, "Geometry of Covariate Shift with Applications to Active Learning," in *Dataset Shift in Machine Learning*, pp. 87–105. 2009.

[8] A. Gretton, A. Smola, J. Huang, M. Schmittfull, K. Borgwardt, and B. Schölkopf, "Covariate Shift by Kernel Mean Matching," in *Dataset Shift in Machine Learning*, pp. 131–160. 2009.

[9] R. S. Shadafan and M. Niranjan, "A dynamic neural network architecture by sequential partitioning of the input space," *Neural Computation*, vol. 6, no. 6, pp. 1202–1222, 1994.