Applications of Support Vector Machines in Bioinformatics and Network Security

Rehan Akbani and Turgay Korkmaz
University of Texas at San Antonio
San Antonio, Texas, USA

1. Introduction

Support Vector Machines (SVM) were introduced by Vapnik and colleagues (Vapnik, 1995) and they have been very successful in application areas ranging from image retrieval (Tong & Chang, 2001) and handwriting recognition (Cortes, 1995) to text classification (Joachims, 1998). However, when faced with imbalanced datasets where the number of negative instances far outnumbers the positive instances, the performance of SVM drops significantly (Wu & Chang, 2003). There are many applications in which instances belonging to one class are heavily outnumbered by instances belonging to another class. Such datasets are called imbalanced datasets, since the class distributions are not evenly balanced. Examples of these imbalanced datasets include the human genome dataset and network intrusion datasets. In the human genome dataset, only a small proportion of the DNA sequences represent genes, and the rest do not. In network intrusion datasets, most of the nodes in a network are benign; however, a small number may have been compromised. Other examples include detecting credit card fraud, where most transactions are legitimate, whereas a few are fraudulent; and face recognition datasets, where only some people on a watch list need to be flagged, but most do not. An imbalance of 100 to 1 exists in fraud detection domains, and it approaches 100,000 to 1 in other applications (Provost & Fawcett, 2001).
Although it is crucial to detect the minority class in these datasets, most off-the-shelf machine learning (ML) algorithms fail miserably at this task. The reason for that is simple: Most ML algorithms are designed to minimize the classification error. They are designed to generalize from sample data and output the simplest hypothesis that best fits the data, based on the principle of Occam's razor. This principle is embedded in the inductive bias of many machine learning algorithms, such as decision trees, which favour shorter trees over longer ones. With imbalanced data, the simplest hypothesis is often the one that classifies all the instances as the majority class.
Consider the scenario where a network consists of 1000 nodes, 10 of which have been compromised by an attacker. If the ML algorithm classifies all of these nodes as uncompromised, it misclassifies only 10 out of 1000 nodes, resulting in a classification error of only 1%. In most cases, an accuracy of 99% is considered very good. However, such a classifier would be useless for detecting compromised nodes.

Therefore, for many imbalanced datasets, the ML classifier ends up classifying everything as the majority class. Artificial Neural Networks (ANNs) use gradient descent with the objective of minimizing the classification error, which usually occurs when the minority class is completely ignored. In K Nearest Neighbours, the vicinity of the test instance is more likely to be dominated by the majority class resulting in a majority class prediction. Decision Trees perform pruning to reduce the risk of over fitting. In most cases, they prune out the leaf with the minority class, leaving only a decision stump at the root with the majority class. Even Support Vector Machines (SVM) fall prey to imbalanced datasets.

The second factor that causes ML algorithms to ignore the minority class is that many of them are designed to ignore noise in the dataset. As a result, they end up treating the minority class as noise and discard them. Many algorithms modify the behaviour of existing algorithms to make them more immune to noisy instances, such as IB3 (Aha, 1992) for kNN, or pruning of decision trees, or soft margins in SVM (Vapnik, 1995). While these approaches work well for balanced datasets, they fail when dealing with highly imbalanced datasets having ratios of 50 to 1 or more (Akbani et al., 2004).

In this chapter, we highlight the reasons why SVM, in particular, fails and what can be done to overcome this. We specifically chose SVM to attack the problem of imbalanced data because SVM is based on strong theoretical foundations (Vapnik, 1995) and it performs well with moderately imbalanced data even without any modifications (Akbani et al., 2004). SVM has its strengths in that the final model is only dependent on the support vectors, whereas the rest of the instances are discarded. Its unique learning mechanism makes it an interesting candidate for dealing with imbalanced datasets, since SVM only takes into account those instances that are close to the boundary, i.e. the support vectors. This means that SVM is unaffected by non-noisy negative instances far away from the boundary even if they are huge in number. Another advantage is that even though there may be more majority class support vectors than minority class, because of Karush-Kuhn-Tucker conditions, the weights of the minority class support vectors would be higher, resulting in some offsetting.

We use the human genome dataset and the network security dataset as illustrative examples. The major difference between the properties of these datasets is that the imbalance ratio for the human genome dataset is very large, but we know what that ratio is at the time of training. Both the training and test sets are derived from the same distribution (the human genome), so the imbalance ratio in the train and test sets would be the same. For network security, however, we do not know at the time of training what the imbalance ratio in a real network would be. To make matters worse, that ratio is not expected to remain constant and would vary as attackers compromise more nodes, or are detected and removed from the network. The imbalance ratio is expected to change dynamically and any algorithm needs to adapt to that change. In this chapter, we present techniques to deal with these changes.

2. Effects of Imbalance on SVM

In order to combat the effects of imbalance, we need to understand exactly why SVM's performance deteriorates with high imbalance ratios. To do that, we need to look at how soft margin SVMs work. Given a set of labelled instances $X_{train} = \{x_i,\ y_i\}^n_{i=1}$ and a kernel function

K, SVM finds the optimal a_i for each x_i to maximize the margin γ between the hyper plane and the closest instances to it. The class prediction for a new test instance *x* is made through:

$$sign\left(f(x) = \sum_{i=1}^{n} y_i \alpha_i K(x, x_i) + b \right) \tag{1}$$

where *b* is the threshold. 1-norm soft-margin SVMs minimize the primal Lagrangian:

$$L_p = \frac{\|w\|^2}{2} + C\sum_{i=1}^{n} \xi_i \ - \sum_{i=1}^{n} \alpha_i \left[y_i \left(w \ . \ x_i + b \right) - 1 + \xi_i \right] - \sum_{i=1}^{n} r_i \xi_i \tag{2}$$

where $a_i \geq 0$ and $r_i \geq 0$ (Cristianini & Shawe-Taylor, 2000). The penalty constant *C* represents the trade-off between the empirical error ξ and the margin. In order to meet the Karush-Kuhn-Tucker (KKT) conditions, the value of a_i must satisfy:

$$0 \leq \alpha_i \leq C \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{3}$$

2.1 Reasons for performance loss with imbalanced data

1. *Weakness of Soft-Margins*. The most significant factor for the loss in performance of SVMs is the weakness of soft margin SVMS. Mathematically, we can see from eq. 2 that minimizing the first term on the right hand side $||w||^2/2$, is equivalent to maximizing the margin γ, while minimizing the second term $C \sum\xi$ minimizes the associated error. The constant *C* specifies what trade-off we are willing to tolerate between maximizing the margin and minimizing the error. If *C* is not very large, SVM simply learns to classify everything as negative because that makes the margin the largest, with zero cumulative error on the abundant negative examples. The only trade-off is the small amount of cumulative error on the few positive examples, which does not count for much. This explains why SVM fails completely in situations with a high degree of imbalance.

2. *Positive points lie further from the ideal boundary.* Wu and Chang (Wu & Chang, 2003) point out this factor as one source of boundary skew. They mention that the imbalance in the training data ratio means that the positive instances may lie further away from the "ideal" boundary than the negative instances. This is illustrated by way of example that if we were to draw *n* randomly chosen numbers between 1 to 100 from a uniform distribution, our chances of drawing a number close to 100 would improve with increasing values of *n*, even though the expected mean of the draws is invariant of *n*. As a result of this phenomenon, SVM learns a boundary that is too close to and skewed towards the positive instances.

3. *Imbalanced Support Vector Ratio*. Another source of boundary skew according to Wu and Chang (Wu & Chang, 2003) is the imbalanced support vector ratio. They found that as the training data gets more imbalanced, the ratio between the positive and negative support vectors also becomes more imbalanced. They hypothesize that as a result of this imbalance, the neighbourhood of a test instance close to the boundary is more likely to be dominated by negative support vectors and hence the decision function is more likely to classify a boundary point negative. We would like to point out however, that because of the KKT conditions in eq. 3, the sum of the *a*'s associated with the positive support vectors must be

equal to the sum of the *a*'s associated with the negative support vectors. Because there are fewer positive support vectors with correspondingly fewer *a*'s, each positive support vector's *a* must be larger than the negative support vector's *a* on average. These *a*'s act as weights in the final decision function (eq. 1) and as a result of larger *a*'s the positive support vectors receive a higher weight than the negative support vectors which offsets the effect of support vector imbalance to some extent. This shows why SVM does not perform too badly compared to other machine learning algorithms for moderately skewed datasets.

3. Applying SVM to Bioinformatics

To illustrate the degree of imbalance encountered in bioinformatics, we use the example of trying to identify parts of genes in human DNA. Human DNA consists of 23 pairs of chromosomes. The Human Genome Project sequenced these chromosomes and we now have almost the entire DNA sequence of 3 billion base pairs. However, not all of the 3 billion base pairs in DNA code for proteins. In fact, the vast majority of DNA does not code for proteins. Portions of DNA that code for proteins are called genes. Genes have several components which are illustrated in Figure 1.

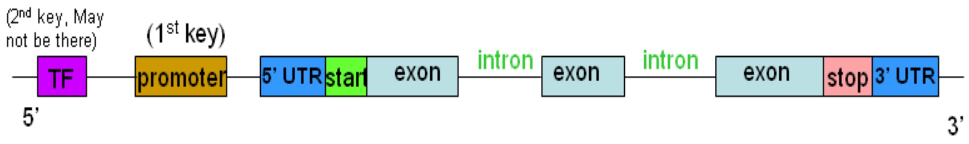


Fig. 1. Components of a typical gene

1st and 2nd keys (promoters): These regions aid in initiating gene expression (i.e. protein production). They may be absent.
5′ UnTranslated Region (UTR): This is the point where mRNA transcription begins.
Start (Translation Initiation Site – TIS): This is the position where translation begins i.e. proteins start to be coded from this site.
Exon: This is region of DNA that codes for the protein.
Intron: This region of DNA is interspersed between two exons and it does not code for proteins. It is spliced out from the mRNA before translation begins.
Donor Site (not shown): The junction where an exon meets an intron.
Acceptor Site (not shown): The junction where an intron meets an exon.
Together the donor and acceptor sites are called **splice sites**.
Stop: This is the termination site where protein synthesis stops. It always consists of one of three possible codons, TGA, TAA or TAG.
3′ UTR: This is the region after the stop codon that does not code for a protein but it forms the tail of the mRNA that is produced by the gene.

The problem of gene finding is to identify the locations of each of these components on the DNA sequence. For our example, we only try to identify the start and stop codons. Almost all start codons have the sequence ATG, while all stop codons consist of either TAA, TAG or TGA. But not all ATG sequences are start codons and not all TAA, TAG and TGA sequences are stop codons or else the problem would become trivial. We counted the number of times

ATG occurs in the entire human genome and found that it occurs approximately 104 million times. Note that if the DNA sequence was random then ATG would occur $(1 / 4^3) \times 3 \times 10^9$ = 47 million times. By contrast, there are an estimated 23,000 confirmed and unconfirmed genes that have ATG as the start codon. So assuming 23,000 genes the estimated degree of imbalance in predicting when an ATG sequence is a start codon, (henceforth called positive instances), and when it is not (negative instances) is:

ATG is a start codon : ATG is not a start codon
23,0000 : 104,000,000
1 : 4,522

Looking at the problem of identifying stop codons independently of other predictions poses an even greater degree of imbalance. We have found that there are about 300 million TAG, TAA or TGA sequences in the human genome. Only 23,000 of them are suspected to be actual stop codons. The imbalance ratio for stop codons is therefore:

23,000 : 300,000,000
1 : 13,043

Unfortunately, ordinary machine learning algorithms are incapable of handling this extremely high degree of imbalance.

4. Related Work

Several researchers have approached the problem of trying to identify the start codon, or translation initiation site (TIS). Stormo et al. (Stormo et al., 1982) use a perceptron and train it using DNA sequences obtained from E. coli. They used a feature vector containing four bit encodings of the nucleotide sequence as their training data. The window size of the feature vector was up to 101 base pairs.
Pedersen and Nielsen (P&N) (Pedersen & Nielsen, 1997) constructed their famous dataset from eukaryotic DNA data. They removed the introns and joined together the resulting exons. They used only those sequences that had the TIS annotated, with at least 10 upstream and 150 downstream nucleotides. They also removed redundant sequences. This dataset has been used by several researchers for TIS prediction. Their dataset contains 3312 ATG sites that are TIS and 10063 sites that are not TIS, giving an imbalance of only around 1:3. Pedersen and Nielsen used a 3-layer neural network and a window size of 203 base pairs to predict the TIS. They obtained a sensitivity score of 78%, specificity of 87% and an accuracy of 85%. Their program is called NetStart and is available for public use.
Zien et al. (Zien et al., 2000) use a modified Support Vector Machine kernel to predict the TIS. They engineer the SVM kernel to incorporate prior biological knowledge in the learning scheme. Essentially they modified the kernel so that nucleotides that are close together have a greater impact on the outcome rather than those that are further apart. They achieved a sensitivity of 70%, specificity of 94% and an accuracy of 88% using the P&N dataset.
Zeng et al. (Zeng et al., 2002) and Li et al. (Li et al., 2004) focus on feature selection rather than a specific ML algorithm. They construct a huge variety of features from the P&N dataset and then use standard feature selection algorithms to decide which features to keep.

Zeng et al. used 9 different features for use in training. They claim that their technique is independent of the base classifier used and outperforms any other classifier.
Salamov et al. (Salamov et al., 1998) used the linear discriminant function to train their classifier using a set of 6 features constructed from the P&N dataset. They achieved an accuracy of 89%. Hatzigeorgiou (Hatzigeorgiou, 2002) used two feed-forward neural networks and a ribosome scanning rule to obtain a classifier with an accuracy of 94%.
It should be noted, however, that none of these methods deal directly with the entire chromosome. Most of them use the P&N dataset that has an imbalance of only 1:3. As mentioned earlier the amount of imbalance in the human genome is about 1:4522. The author suspects that these methods will fail when applied to the entire genome.
The problem of imbalanced datasets has been approached from two main directions. The first approach is to preprocess the data by under sampling the majority instances or oversampling the minority instances. Kubat and Matwin (Kubat & Matwin, 1997) proposed a one-sided selection process which under sampled the majority class in order to remove noisy, borderline, and redundant training instances. But if we use SVM as our classifier, removing redundant (far away) instances has no effect and removing borderline instances may adversely affect the accuracy of the learned hyper plane.
Japkowicz (Japkowicz, 2000) evaluated the oversampling and under sampling techniques for skewed datasets and concluded that both methods were effective. Ling and Li (Ling & Li, 1998) combined oversampling with under sampling, but this combination did not provide significant improvement in the "lift index" metric that they used. Chawla et al. (Chawla et al., 2002) devised a method called Synthetic Minority Oversampling Technique (SMOTE). This technique involved creating new instances through "phantom-transduction." For each positive instance, its nearest positive neighbours were identified and new positive instances were created and placed randomly in between the instance and its neighbours.
The other approach to dealing with imbalanced datasets using SVM biases the algorithm so that the learned hyper plane is further away from the positive class. This is done in order to compensate for the skew associated with imbalanced datasets which pushes the hyper plane closer to the positive class. This biasing can be accomplished in various ways. In (Wu & Chang, 2003) an algorithm is proposed that changes the kernel function to develop this bias, while in (Cristianini, 2002) the kernel matrix is adjusted to fit the training data. Veropoulos et al. (Veropoulos et al., 1999) suggested using different penalty constants for different classes of data, making errors on positive instances costlier than errors on negative instances.

5. Our Method

Since an imbalance ratio of over 1:1000 is well beyond the performance capabilities of any ML algorithm, we decided to generate the TIS data from the human genome with an imbalance of 1:100 for our current scheme. Even this ratio causes most ML algorithms to perform very poorly. This ratio is still much higher than the P&N dataset which has an imbalance ratio of only 1:3.
Our first strategy was to construct a dataset containing sequences from the human genomic data and then use it to generate several candidate features for our algorithm. We then used feature selection algorithms to select the best attributes from among them. This technique was originally proposed by Zeng et al. (Zeng et al., 2002). To begin with, we randomly chose

known ATG TIS sites from the NCBI database for our positive examples. Then we randomly picked ATG sites from the genome that are not known to be TIS sites, for our negative examples. We maintained a ratio of 1:100 for positive to negative examples. A window of 200 nucleotides was chosen for every example, running from 100 bps upstream of the ATG to 100 bps downstream of the ATG. This set constituted our raw dataset.

From this raw dataset, we generated several features. Every position in the raw data was used as a candidate feature. In addition, we generated the frequency of occurrence of all possible monomers, dimers, trimers, all the way up to hexamers that lie upstream of the ATG and also for those that lie downstream of the ATG. This gave us a total of 11120 features. Then we ran several different feature selection algorithms on this large set of attributes to determine the top attributes. We ran the Correlation Feature Selection (CFS) algorithm, which prefers those set of attributes that have a high correlation with the class label, but low correlation among themselves, and also Information Gain, Gain Ratio, and chi-squared test. By observing their results, we were able to choose the top 15 of the 11120 attributes, which were found to be the following (in order of importance): dn-CG, dn-TA, dn-AT, up-AT, up-CG, dn-GC, dn-G, up-TA, dn-CGG, up-CGG, dn-T, dn-ATT, pos -3, pos -1, pos +4, where dn-CG means the frequency of occurrence of CG downstream of the ATG, and up-CG means the frequency of CG upstream of the ATG, pos -3 means the nucleotide at position -3. Although we found pos -3, pos -1 and pos +4 to be the most important positions, the relevance score for these was much lower than the relevance score for the frequency counts, but we included them in our experiments nevertheless. It should also be noted that these positions correspond to the Kozak consensus sequence (Kozak, 1996). Our final dataset consisted of these 15 selected features. We used a similarly generated separate test set for evaluation.

We needed to modify the basic SVM algorithm to overcome some of the problems mentioned in Section 2. One of those problems is that with imbalanced datasets, the learned boundary is too close to the positive instances. We need to bias SVM in a way that will push the boundary away from the positive instances. Veropoulos et al. (Veropoulos et al., 1999) suggest using different error costs for the positive (C^+) and negative (C^-) classes. Specifically, they suggest changing the primal Lagrangian (eq. 2) to:

$$L_p = \frac{\|w\|^2}{2} + C^+ \sum_{\{i|yi=+1\}}^{n_+} \xi_i + C^- \sum_{\{j|yj=-1\}}^{n_-} \xi_j - \sum_{i=1}^{n} \alpha_i \left[y_i (w \, . \, x_i + b) - 1 + \xi_i \right] - \sum_{i=1}^{n} r_i \xi_i \tag{4}$$

The constraints on a_i then become:

$$0 \le \alpha_i \le C^+ \text{ if } y_i = +1 \quad \text{and} \quad 0 \le \alpha_i \le C^- \text{ if } y_i = -1 \tag{5}$$

Furthermore, we note that $\xi_i > 0$ only when $a_i = C$ (Liu et al., 2004). Therefore non-zero errors on positive support vectors will have larger a_i while non-zero errors on negative support vectors will have smaller a_i. The net effect is that the boundary is pushed more towards the negative instances. However, a consequence of this is that SVM becomes more sensitive to the positive instances and obtains stronger cues from the positive instances about the orientation of the plane than from the negative instances. If the positive instances are sparse, as in imbalanced datasets, then the boundary may not have the proper shape in the input space as illustrated in Figure 2.

The solution we adopted to remedy the problem of sparse positive instances was to generate several synthetic minority instances, in line with Chawla et al's technique (Chawla et al., 2002). We repeatedly randomly selecting two neighbouring positive instances using the Euclidean distance measure and then generated a new instance that lies somewhere randomly in between these instances. The underlying assumption was that the space between two positive neighbouring instances was assumed to be positive. We found this assumption to hold for our dataset. We found that over sampling the minority class in this way was much more effective than the traditional over sampling technique of generating multiple identical copies of existing minority instances. Simply resampling the minority instances merely overlaps the instances on top of each other and does not help in "smoothing out" the shape of the boundary. We synthetically generated new instances between two existing positive instances which helped in making their distribution more well-defined. After this over sampling, the input space may look like Figure 3.

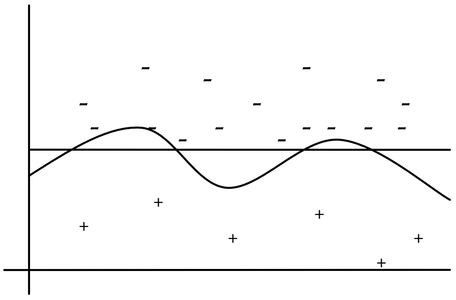


Fig. 2. The learned boundary (*curved line*) in the input space closely follows the distribution of the positive instances. The ideal boundary is denoted by the horizontal line

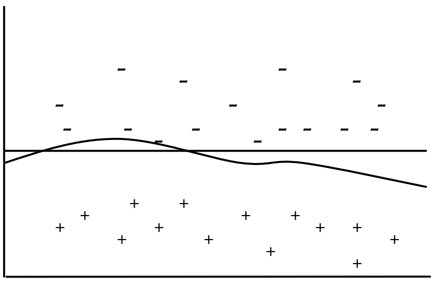


Fig. 3. After oversampling, the positive instances are now more densely distributed and the learned boundary (*curved line*) is more well defined

Synthetic oversampling also alleviates the problem of soft margins, since now the cost of misclassifying minority instances is significant. This, together with higher costs of misclassifying minority instances levels the playing field for both classes.
To summarise, our method consists of:

1. Generating and selecting the most relevant features for the problem.
2. Using different error costs for different classes to push the boundary away from the positive instances and overcome soft margin weakness.
3. Generating synthetic minority instances to make the positive instances more densely distributed in order to make the boundary more well defined.

6. Results

We compared our algorithm with several other standard ML algorithms for predicting TISs in the human genome. We also compared our technique with the common approach of over sampling the minority class or under sampling the majority class in order to reduce the

Algorithm	F-Measure
Voted Perceptron	0
ZeroR	0
SVM	0
SVM with Under Sampling	0.041
SVM with Over Sampling	0.082
Neural Network	0.133
AdaBoost with C4.5	0.148
3 Nearest Neighbours	0.182
Decision Tree	0.2
Naive Bayes	0.205
Bagging with C4.5	0.25
Our Algorithm	0.44

Table 1. Performance of various ML algorithms vs. our algorithm on the TIS dataset

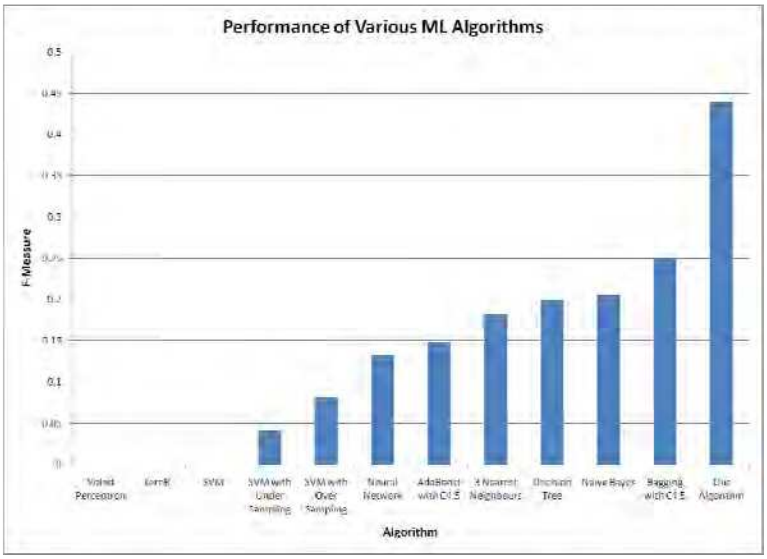


Fig. 4. Plotted F-Measure of various ML algorithms vs. our algorithm on the TIS dataset

imbalance ratio in the dataset prior to training. For evaluation, we used the F-measure as our metric, which is the harmonic mean of the recall and precision. The results are shown in Table 1 and plotted in Figure 4. They illustrate how poorly standard ML approaches perform for predicting TISs at the genomic level due to the high imbalance ratio. Our approach improves the performance significantly. By varying the parameters of our algorithm we are able to obtain different recall and precision values. Some examples of recall/precision obtained respectively are: 15%/100%, 29%/95%, 85%/4%. Thus, depending on the application the algorithm parameters can be varied to obtain the desired level of recall vs. precision.

While the results are encouraging, we must bear in mind that the TIS dataset we generated was a watered down version with 1:100 imbalance ratio, compared to the 1:4522 imbalance ratio found in the human genome. The search for algorithms that can deal with such large imbalances is far from over. However, we suggest using heuristics based on domain knowledge to discard those ATG codons which are unlikely to be TISs, instead of directly feeding them into the ML classifier. These heuristics may include ATG codons that are too far from, or too close to stop codons or splice sites. This will reduce the imbalance ratio to some extent and may improve performance.

7. Applying SVM to Network Security

In network intrusion detection, the goal is to identify compromised nodes in the network. One approach towards accomplishing this is to monitor the behaviour of nodes in order to detect anomalous or malicious behaviour. Reputation Systems, such as seller ratings on eBay, rely on the postulate that past behaviour can be used to predict future behaviour. If a node has behaved maliciously in the past, it will likely behave maliciously in future. The objective is to detect nodes that behave maliciously and avoid interacting with them a priori. In general, we can summarize existing RSs found in the literature (Jiang & Baras, 2006; Srivatsa et al., 2005; Kamvar et al., 2003; Josang & Ismail, 2002) within a general framework as shown in Figure 5. According to this framework, a node that needs to decide whether to transact with another node or not must first gather historical data about that node (e.g., the proportion of good vs. bad transactions in the last *x* minutes). Then it applies a customized mathematical equation or statistical model to the data to produce an output score. For example, the RS in (Kamvar et al., 2003) is based on using Eigen values from Linear Algebra, the one in (Srivatsa et al., 2005) is based on using derivatives and integrals, whereas the one in (Josang & Ismail, 2002) is based on Bayesian systems utilizing the Beta distribution. Depending on the output of the equation or model, the system then decides how to respond. In most cases, the equation or model is customized to detect specific types of malicious behaviour only. For instance, the algorithm in (Srivatsa et al., 2005) is designed to detect malicious behaviour that alternates with good behaviour and varies over time.

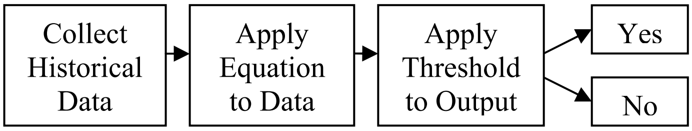


Fig. 5. General framework of a Reputation System that decides whether to transact with a given node or not.

In contrast to developing a separate module for each attack pattern, we can employ Machine Learning, specifically Support Vector Machines (SVM), to build a flexible and dynamic RS that can be trained to thwart a multitude of attack patterns easily and efficiently. It can also be retrained to detect new, previously unknown attack patterns.

8. Basic Machine Learning Approach

Using Figure 5, we can redefine the problem of designing Reputation Systems (RS) into one of finding the optimal set of input features and equations (steps 1 and 2 in Fig. 5) that allow us to distinguish between malicious and benign nodes with high accuracy. Machine Learning (ML) is of particular significance in this context since many ML algorithms are able to determine and approximate the optimal equation needed to classify a given set of data. We envision the problem of RS as a time series prediction problem, which states: Given the values of the dependent variable at times (*t, t-1, t-2, ..., t-n*), predict the value of the variable at time (*t + 1*) (Baras & Jiang, 2005; Jiang & Baras, 2004). The dependent variable in this case is the proportion of good transactions conducted by a node in a given time slot. Predicting this variable at time (*t + 1*) gives us the probability that the node will behave well if we choose to transact with it at time (*t + 1*). Therefore, we opted to use Support Vector Machines (SVM) as our ML algorithm because it has been shown to successfully approximate mathematical functions (Abe, 2005) and make time series predictions (Camastra & Filippone, 2007).

In our scheme, we build SVM models against different types of malicious behaviours offline, and then upload those models to the nodes in the network. The nodes can use those models to classify new nodes and predict if a new node is malicious or not. Constructing models is computationally expensive so it is done offline, possibly by a third party. However, the classification step is not very expensive and can be done on the node in real time. When a new type of attack is discovered, a new model can be constructed against it. This is similar to how anti-virus systems work where the anti-virus is developed offline and then uploaded to clients. Similarly, in our scheme the vendor of the RS might update its subscribers with SVM models against new attacks.

An implied assumption is that after a transaction has taken place, a node can determine if the transaction was good or bad with a certain high probability. This is true in many cases, such as in commercial transactions on eBay, as well as in file downloads (where a corrupted or virus infected file would be considered bad), or in providing network services (Baras & Jiang, 2005; Jiang & Baras, 2004). Another assumption is that the feedbacks can be reliably transmitted without being tampered with. This can be accomplished by a node digitally signing every feedback it sends. These assumptions are made by many researchers in the field (Jiang & Baras, 2006; Srivatsa et al., 2005; Kamvar et al., 2003) and we also make the same assumptions in our study. However, a few transactions might be incorrectly labelled good or bad. SVM can handle fair amounts of such “noise” in the dataset (Abe, 2005).

9. Building the Core SVM based Reputation System

If all the participants in a network gave honest and correct feedbacks about the transactions they conducted, then it would be trivial to spot malicious nodes since all the good nodes would have 100% positive feedbacks, whereas the malicious nodes would not. But in reality, this is not the case and we have to deal with three principle challenges:

i. Dishonest feedback given by malicious nodes against other nodes they have transacted with.
ii. Incorrect feedback from legitimate nodes by mistake (noise).
iii. Fake feedback given by malicious nodes about transactions that never really occurred.

Our goal is to use SVM to tackle problems *i* and *ii*. However, SVM cannot detect if a feedback was fake, but digitally signed certificates can be used to solve problem *iii* (Akbani et al., 2008). We assume that the proportion of dishonest to honest feedbacks given by malicious nodes is much higher than the proportion of incorrect to correct feedbacks given by legitimate nodes. This is how we can distinguish between inadvertent noise and deliberately false feedbacks. If malicious nodes reduce the proportion of dishonest feedbacks to match those of incorrect feedbacks, we have still succeeded in our goal of reducing malicious behaviour.

We have to take into account several factors when building the SVM based RS that will deal with problems *i* and *ii*. The most important factor is the set of features to use. We divide time into regular intervals called time slots. The network administrator can choose and fix a time slot that is a few minutes to a few hours long, depending on how frequently nodes in the network transact on average. The features in our experiments consist of the proportion of positive vs. negative feedbacks assigned to a node during a given time slot by the nodes it has transacted with. To collect features for a test node, we need to query all the nodes in the network and ask them to provide us any feedbacks they have about the node for a given slot. The fraction of positive feedbacks versus total feedbacks for that slot forms a single feature. Each time slot then corresponds to one feature. This is in accordance with (Srivatsa et al., 2005), and is also based on features used in time series prediction problems (Camastra & Filippone, 2007). The number of features is also important. Using too few features might not provide sufficient information to the classifier, whereas using too many might result in the "Curse of Dimensionality" (Abe, 2005) and spurious overheads. We can vary the number of features by varying the number of time slots used. We use 15 time slots for our core SVM.

Next, we need to consider the proportion of malicious nodes vs. good nodes for the training set, called the imbalance ratio. In an actual setting, we would not know the proportion of malicious nodes in the network, so the testing should be done with varying imbalance ratios. However, the training set can only have one imbalance ratio since we need to build just one SVM model. We use a malicious node proportion of about 60% since that ratio yields good results.

For SVM, the kernel used is another key factor. We decided to use the linear kernel since it performs well and it is computationally less expensive to build and test than other kernels. The size of the training dataset used to train the classifier was 1,000 instances.

10. Evaluation of SVM based Reputation System

In order to evaluate the SVM based RS, we generated several datasets using simulations of a network consisting of 1,000 nodes. Time was divided into slots and in each time slot, several transactions were conducted between two randomly chosen pairs of nodes. Each node would then label the transaction as good or bad and store that label. The label may or may not reflect the true observation of a node, i.e. a node may lie about a transaction and give dishonest feedback (problem *i*).

Good Behaviour: Good behaviour is characterized as a node conducting a normal transaction and giving honest feedback about it.

Bad Behaviour: Bad behaviour is characterized as a node conducting a malicious transaction and/or giving dishonest feedback.

In addition, we introduced a random error of 5% to account for the fact that a node may incorrectly detect a transaction and mistakenly label it good or bad. This corresponds to problem *ii* described above.

The simulation was allowed to run for several time slots and then data about each node was gathered. To gather data about a node *x*, all the other nodes in the network were queried and asked to give information about *x* going back a certain number of time slots. The total number of good and bad transactions conducted by *x* in a given time slot were accumulated and the proportion of positive feedback was computed. This computation was repeated for each time slot of interest. In this way a concise, aggregate historical record of *x* was obtained. The correct label of malicious or benign was assigned to *x* by us, based on its role in the simulation, for testing purposes only. The adversarial model was as follows.

10.1 Adversarial Model

All the malicious nodes in the network behaved in one of four different ways. A quarter of the malicious nodes behaved maliciously all the time. Another quarter oscillated their behaviour, alternating between good and bad with a randomly chosen frequency and duty cycle. The third quarter also oscillated their behaviour, but they colluded with other malicious nodes and left positive feedbacks about each other whenever they interacted. The last quarter had oscillating behaviour with lots of collusion where malicious nodes conducted multiple transactions with each other. In a real world setting we would not know which, if any, attack was being launched by any given node, so the performance of the RS in this attack scenario would tell us what would happen if similar attacks were conducted simultaneously.

10.2 Experiments and Results

We evaluated our core SVM based RS against two other algorithms, TrustGuard Naïve and TrustGuard TVM (Trust Value based credibility Measure) (Srivatsa et al., 2005). We set the same parameters for TrustGuard that their authors used in their paper. TrustGuard's authors have shown that it performs very well compared to eBay's reputation system, which is commonly used as a benchmark in the literature for RSs. Therefore, we decided to directly compare our performance with TrustGuard, instead of eBay.

In the initial set of experiments, we collected data going back 15 time slots for each simulation run. For oscillating behaviour, the period of oscillations was kept less than 15 to ensure it was distinguishable from legitimate behaviour. For SVM, a separate set of training data was also generated and SVM was trained on it. The training data had a fixed proportion of malicious nodes (about 60%).

For each node, its transaction history for the last 15 slots was fed into each RS. Then using the output of the RS, a determination was made about whether the node was malicious or benign. For SVM this was done by looking at the distance between the test node and the decision boundary. If this distance was greater than a threshold, the node was considered benign. Larger thresholds result in fewer false positives, but also fewer true positives. This might be desirable in critical applications where we want to be sure that a node that is given access to a resource is indeed good, even if that means denying access to some legitimate nodes. TrustGuard also outputs a score that can also be compared against a threshold and access can be granted if the score is greater than the fixed threshold.

Classification Error: In the first set of experiments, the thresholds were fixed at their midpoint values so that the results were not artificially biased either towards increasing true positives (lower thresholds) or decreasing false positives (higher thresholds) but were halfway. Since the range of thresholds for SVM is $(-\infty, \infty)$, its threshold was set to 0. The range for TrustGuard is $[0, 1]$, so its threshold was set to 0.5. Then the percentage of malicious nodes in the network was varied. The proportion of nodes that were misclassified, or the classification error, was measured. The results are illustrated in Figure 6. The results show that SVM significantly outperforms TrustGuard's Naïve and TVM algorithms, even if the proportion of malicious nodes is very large (i.e. 80%). It is also interesting to note that there is not much difference between TrustGuard's TVM and Naïve algorithms, even though TVM is much more complex.

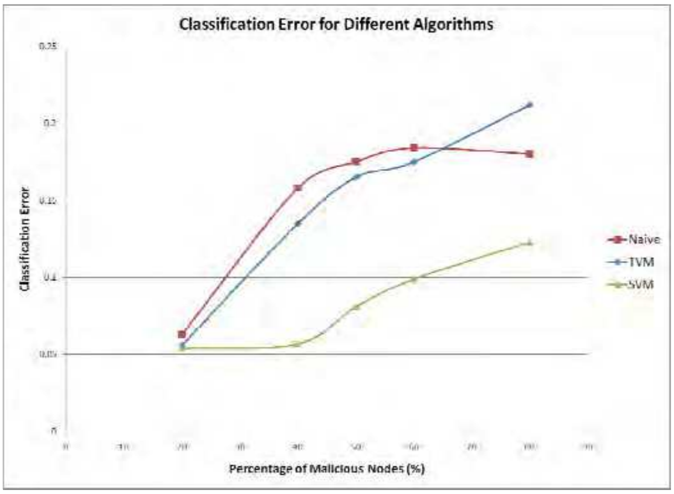


Fig. 6. Classification Error vs. Proportion of malicious nodes

ROC Curves: To obtain more in depth evaluations, we ran hundreds of more simulations in order to generate ROC curves for all three RSs. ROC curves are commonly used in Machine Learning to evaluate classifiers, irrespective of the thresholds used. The curve is obtained by varying the threshold, so that we can compare how the true positive rate varies with the false positive rate. The area under the ROC curve shows how good a classifier is. Classifiers with larger areas under the curve are better. The ideal ROC curve is an upside down L-shaped curve, containing the point (0, 1) that corresponds to 100% true positive rate and 0% false positive rate.
Each point on the curve was obtained by running 30 simulations with different random number seeds, and then taking their mean. Confidence Intervals were taken around each point to ensure that the curve of SVM did not overlap with that of TrustGuard (the confidence intervals are too small to be visible on the graphs). The results are shown in Figure 7, along with the diagonal random "guessing" line which corresponds to randomly guessing the label of the given nodes. The results show that SVM again outperforms TrustGuard, regardless of the thresholds used. The area under the curve is greater for SVM than TrustGuard.

11. Dynamic Thresholds with SVM

The major problem with network intrusion datasets is that we do not know the imbalance ratio at the time of training. We do not know how many malicious nodes there will be in a real network and we cannot expect their proportion to remain constant. However, the classification error could be greatly improved if we knew the imbalance ratio in the network at any given point in time. We could, for instance, vary the bias that SVM uses in order to obtain better classification accuracy.

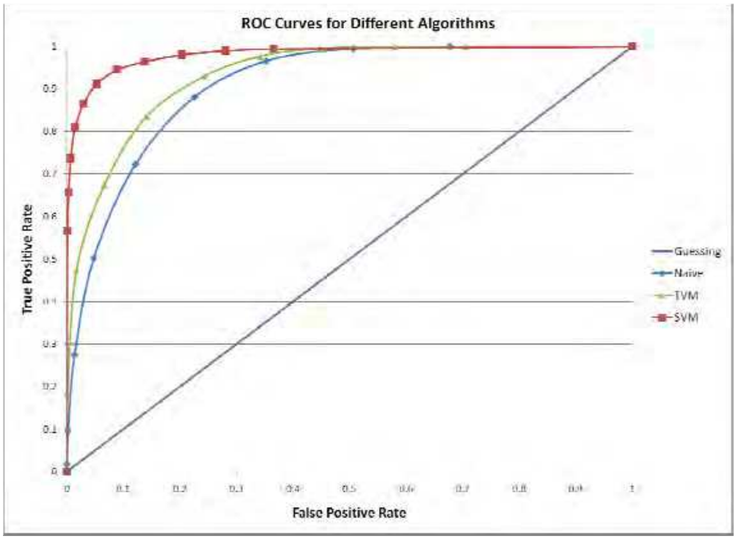


Fig. 7. ROC Curves for Different Algorithms

The general equation of a linear SVM is (Abe, 2005):

$$w.x + b \geq 0 \quad \text{for positive class}$$
$$w.x + b < 0 \quad \text{for negative class}$$

Where x is the instance vector, w is the normal to the SVM hyper plane, and b is the bias threshold. The reason why SVM's performance degraded at high malicious node proportions was because it was misclassifying the positive (non-malicious) instances. Therefore, we can increase the threshold, b, slightly so that those instances close to the SVM boundary are classified as positive. In other words, by increasing b we can effectively trade false negatives with false positives.

The bias pays off at higher proportions of malicious nodes when there are more false negatives than false positives. However, it costs us when the proportion of malicious nodes is small since there are more false positives than false negatives. This increases the error for small proportions. This observation led us to the idea that if we could know the proportion of malicious nodes in the network, we could adjust the bias threshold accordingly to optimize accuracy. The threshold would be decreased for fewer malicious nodes, and increased for more malicious nodes. We propose a scheme called "Dynamic Thresholds" that utilizes this idea.

To begin with, we determine what the ideal thresholds are for given proportions of malicious nodes using brute force trial and error. The ideal threshold is defined as that threshold which maximizes accuracy. We expect the threshold curve to be specific for a given SVM model, and each model would have its own associated curve. The ideal thresholds curve for our dataset is given in Figure 8.

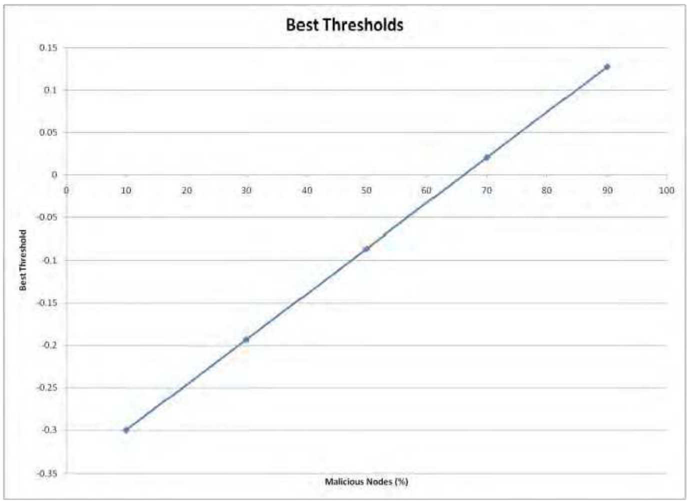


Fig. 8. SVM’s best threshold vs. Malicious Nodes percentage

In our simulations, all the $w.x$ values were normalized between the range (-1, 1). The threshold value, b, is with reference to this range. We generated more datasets and introduced greater variations in the duty cycles and frequencies of the oscillating behaviour of malicious nodes. As expected, this degraded the performance of the default SVM model. Then we used the best thresholds to determine the difference in error. Figure 9 shows the reduction in error achieved using the optimum thresholds versus using the default threshold of zero.

The results clearly show that dynamic thresholds can be very useful for significantly reducing the error. However, the challenge is that in a real world setting, we do not know the proportion of malicious nodes in the network and therefore, we cannot decide what threshold to use. To overcome this, we propose estimating the proportion of malicious nodes through sampling.

The idea is that a new node that joins the network would initially use the default threshold of zero. It would conduct transactions as usual and estimate the proportion of malicious nodes from all the nodes it has interacted with or received feedback about. A node is considered malicious if either the Reputation System classifies it as malicious, or if a transaction is conducted with the node and the transaction is deemed to be malicious. Once a large enough sample is collected by the node, it can use that sample to estimate the proportion of malicious nodes in the network and then dynamically adjust its threshold to improve the RS’s accuracy.

We conducted experiments to determine what a good sample size would be before adjusting the threshold. We determined that once about 20 to 25 samples have been collected, a fairly good estimate of the actual proportion can be obtained. We therefore recommend that nodes should obtain a sample of at least 20 to 25 before adjusting their thresholds.

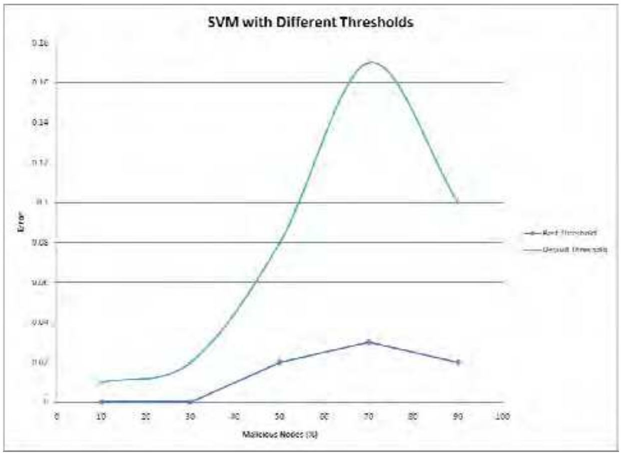


Fig. 9. SVM error under the default threshold and under the best thresholds

Figure 10 shows the reduction in error when Dynamic Thresholds are put into practice. Sample sizes of 20 and 25 are used. These samples are randomly collected and classified, based on a node's interactions with other nodes, and used to estimate the proportion of malicious nodes in the network. The threshold is adjusted based on the estimated proportions. The results show a significant reduction in error even at high proportions. Once the threshold has been adjusted, the sample is discarded and a fresh sample is started. In this way, the node can continuously monitor the proportion of malicious nodes and adjust its threshold as the proportion changes over time.

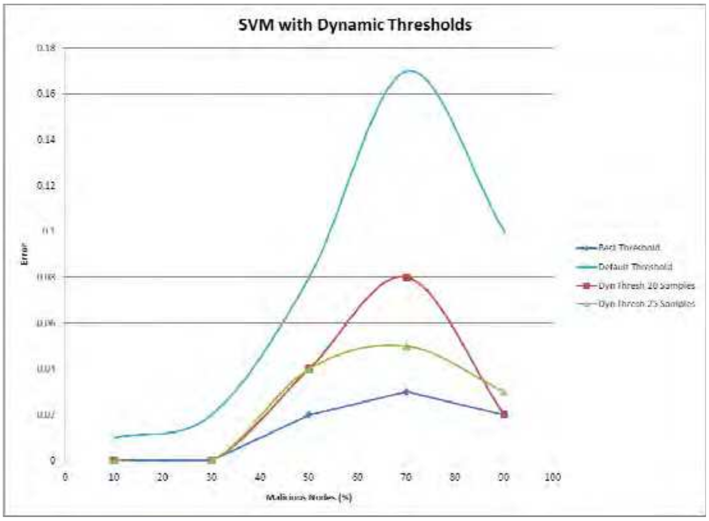


Fig. 10. Dynamic Thresholds Error with 20 and 25 samples compared to the default error and the minimum possible error

12. Conclusions and Future Work

This chapter discusses the issues faced when trying to train SVM on imbalanced datasets. The main reason why SVM performs poorly for such datasets is because of the weakness of soft margins. Soft margins were introduced in order to make SVM resilient against non-separable datasets. The idea was to tolerate some classification error as a trade off for maximizing the margin between the support vectors. But this has an adverse effect when it comes to imbalanced datasets. SVM ends up with a hyper plane that is far from the entire cluster of instances. Any instance that is on the same side of the plane as the cluster is classified as the majority class. Having the hyper plane further from the instances maximizes the margin, at the cost of misclassifying the minority class. But since there are only a few instances of the minority class, the error is rather small and the benefit of larger margins overcomes this. Therefore, everything is classified as the majority class.

Some solutions to this problem are presented in the chapter and evaluated against human genome and network intrusion datasets. The imbalance ratio in the genome dataset can be as high as 1:4500. This is well beyond the capability of traditional ML algorithms. However, we can use under sampling of the majority class and heuristics to reduce the imbalance ratio. Then we can use the techniques presented in this chapter to improve the performance of SVM on this dataset. These techniques include generating and selecting good features, using different error costs for majority and minority instances, and generating synthetic minority instances to even out the imbalance.

Then we discussed datasets where the imbalance ratio is not known at the time of training. Network intrusion is one such application domain where the number of malicious nodes in the network is unknown at the time of training. We introduced dynamic thresholds to try to estimate this proportion and then adjust the SVM model's parameters to significantly improve its performance. We also showed that building Reputation Systems and automatically determining their rule sets using Machine Learning is not only feasible, but yields better results than some of the manually generated rule sets found in the literature.

Although the techniques presented in this chapter have been shown to significantly improve SVM's performance on imbalanced datasets, there are still limitations on what degrees of imbalance SVM can handle. We have tested SVM on imbalance ratios of 1:100, however, bioinformatics datasets have imbalances of 1 to several thousands. In future, researchers need to invent better algorithms that are capable of handling such huge imbalances.

13. Acknowledgments

This work was supported by US Dept. of Defence Infrastructure Support Program for HBCU/MI, Grant: 54477-CI-ISP (Unclassified), and by the National Science Foundation under grant CRI – 0551501. We would like to thank Dr. G.V.S. Raju and Dr. Stephen Kwek for their assistance and input towards this research.

14. References

Abe, S. (2005). Support Vector Machines for Pattern Classification (Advances in Pattern Recognition), Springer, ISBN 1852339292, Chp 6, 11, London, UK

Aha, D. (1992). Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. International Journal of Man-Machine Studies, Vol. 36, 267-287

Akbani, R.; Kwek, S. & Japkowicz, N. (2004). Applying Support Vector Machines to Imbalanced Datasets, Proceedings of the 15th European Conference on Machine Learning (ECML), pp. 39-50, Pisa, Italy, Sept. 2004, Springer-Verlag, Germany

Akbani, R.; Korkmaz, T. & Raju, G.V.S. (2008). Defending Against Malicious Nodes Using an SVM Based Reputation System, Proceedings of MILCOM 2008, Sponsored by Raytheon, IEEE, AFCEA, San Diego, California, USA, 2008

Baras, J. S. & Jiang, T. (2005). Managing trust in self-organized mobile ad hoc networks, Workshop NDSS, Extended abstract, 2005

Camastra, F. & M. Filippone (2007). SVM-based time series prediction with nonlinear dynamics methods. Knowledge-Based Intelligent Information and Eng. Systems, LNCS, Springer, Vol. 4694, 2007, pp 300-307

Chawla, N.; Bowyer, K.; Hall, L. & Kegelmeyer, W. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, Vol. 16, 2002, pp. 321-357

Cortes, C. (1995). Prediction of Generalisation Ability in Learning Machines. PhD thesis, Department of Computer Science, University of Rochester

Cristianini, N. & Shawe-Taylor, J. (2000). An Introduction to Support Vector Machines and other kernel-based learning methods, Cambridge University Press, ISBN 0521780195, Cambridge, UK

Cristianini, N.; Kandola, J.; Elisseeff, A. & Shawe-Taylor, J (2002). On Kernel Target Alignment. Journal of Machine Learning Research, Vol. 1, 2002

Hatzigeorgiou, A. G. (2002). Translation initiation start prediction in human cDNAs with high accuracy. Bioinformatics 18, pp. 343-350, 2002

Japkowicz, N. (2000). The Class Imbalance Problem: Significance and Strategies, Proceedings of the 2000 International Conference on Artificial Intelligence: Special Track on Inductive Learning, Las Vegas, Nevada, 2000

Jiang, T. & Baras, J. S. (2004). Ant-based adaptive trust evidence distribution in MANET, Proceedings of the 24th International Conference on Distributed Computing Systems, Tokyo, Japan, March, 2004

Jiang, T. & Baras, J. S. (2006). Trust evaluation in anarchy: A case study on autonomous networks, Proceedings of the 25th Conference on Computer Communications, 2006

Joachims, T. (1998). Text Categorization with SVM: Learning with Many Relevant Features, Proceedings of the 10th European Conference on Machine Learning (ECML), 1998

Josang, A. & Ismail, R. (2002). The beta reputation system, Proceedings of the 15th BLED Electronic Commerce Conference, Slovenia, June, 2002

Kamvar, S. D.; Schlosser, M. T. & Garcia-Molina, H. (2003). The EigenTrust algorithm for reputation management in P2P networks, Proceedings of the International World Wide Web Conference, WWW, 2003

Kozak, M. (1996). Interpreting cDNA sequences: Some insights from studies on translation. Mammalian Genome 7, 1996, pp. 563-574

Kubat, M. & Matwin, S. (1997). Addressing the Curse of Imbalanced Training Sets: One-Sided Selection, Proceedings of the 14th International Conference on Machine Learning, 1997

Ling, C. & Li, C. (1998). Data Mining for Direct Marketing Problems and Solutions, Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 1998

Liu, H.; Han H.; Li J. & Wong, L. (2004). Using amino acid patterns to accurately predict translation initiation sites. In Silico Biology 4, Bioinformation Systems, 2004

Pedersen, A. & Nielsen, H. (1997). Neural network prediction of translation initiation sites in eukaryotes: perspectives for EST and genome analysis, Proceeding of the International Conference on Intelligent Systems for Molecular Biology, pp. 226-233, 1997

Provost, F. & Fawcett, T. (2001). Robust Classification for Imprecise Environments. Machine Learning, Vol. 42, No. 3, pp. 203-231

Salamov, A.; Nishikawa, T. & Swindells, M. A. (1998). Assessing protein coding region integrity in cDNA sequencing projects. Bioinformatics 14, pp. 384-390, 1998

Srivatsa, M.; Xiong, L. & Liu, L. (2005). TrustGuard: Countering vulnerabilities in reputation management for decentralized overlay networks, Proceedings of the International World Wide Web Conference, WWW, 2005

Stormo, G.; Schneider, T.; Gold, L. & Ehrenfeucht, A. (1982). Use of the 'Perceptron' Algorithm to Distinguish Translational Initiation Sites in E.coli. Nucleic Acids Res., Vol. 10, pp. 2997–3011

Tong, S. & Chang, E. (2001). Support Vector Machine Active Learning for Image Retrieval, Proceedings of ACM International Conference on Multimedia, pp. 107-118

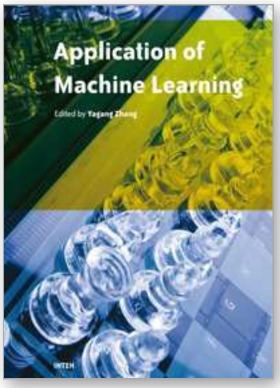
Vapnik, V. (1995). The Nature of Statistical Learning Theory. Springer, ISBN 0387987800, New York, NY, USA

Veropoulos, K.; Campbell, C. & Cristianini, N. (1999). Controlling the sensitivity of support vector machines, Proceedings of the International Joint Conference on AI, pp. 55–60, 1999

Wu, G. & Chang, E. (2003). Class-Boundary Alignment for Imbalanced Dataset Learning, Proceedings of ICML 2003 Workshop on Learning from Imbalanced Data Sets II, Washington DC, USA

Zeng, F.; Yap, H. C. & Wong, L. (2002). Using feature generation and feature selection for accurate prediction of translation initiation sites, Proceedings of 13th Workshop on Genome Informatics, Universal Academy Press, pp. 192-200, 2002

Zien, A.; Ratsch, G.; Mika, S.; Scholkopf, B.; Lemmen, C.; Smola, A.; Lengauer, T. & Muller, K. R. (2000). Engineering support vector machine kernels that recognize translation initiation sites. Bioinformatics, Vol. 16, pp. 799-807, 2000



Application of Machine Learning
Edited by Yagang Zhang

ISBN 978-953-307-035-3
Hard cover, 280 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

The goal of this book is to present the latest applications of machine learning, which mainly include: speech recognition, traffic and fault classification, surface quality prediction in laser machining, network security and bioinformatics, enterprise credit risk evaluation, and so on. This book will be of interest to industrial engineers and scientists as well as academics who wish to pursue machine learning. The book is intended for both graduate and postgraduate students in fields such as computer science, cybernetics, system sciences, engineering, statistics, and social sciences, and as a reference for software professionals and practitioners. The wide scope of the book provides them with a good introduction to many application researches of machine learning, and it is also the source of useful bibliographical information.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Rehan Akbani and Turgay Korkmaz (2010). Applications of Support Vector Machines in Bioinformatics and Network Security, Application of Machine Learning, Yagang Zhang (Ed.), ISBN: 978-953-307-035-3, InTech, Available from: http://www.intechopen.com/books/application-of-machine-learning/applications-of-support-vector-machines-in-bioinformatics-and-network-security