LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-TR-225264

Survey of Anomaly Detection Methods

Brenda Ng

October 13, 2006

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Survey of Anomaly Detection Methods

Brenda Ng
Lawrence Livermore National Laboratory
Livermore, CA
bmng@llnl.gov

Abstract

This survey defines the problem of anomaly detection and provides an overview of existing methods. The methods are categorized into two general classes: *generative* and *discriminative*. A generative approach involves building a model that represents the joint distribution of the input features and the output labels of system behavior (e.g., normal or anomalous) then applies the model to formulate a decision rule for detecting anomalies. On the other hand, a discriminative approach aims directly to find the decision rule, with the smallest error rate, that distinguishes between normal and anomalous behavior. For each approach, we will give an overview of popular techniques and provide references to state-of-the-art applications.

1 Introduction

The goal of anomaly detection is to identify the onset of faulty or novel system behavior, to characterize the nature of such behavior (i.e., benign or malicious) and to propose possible causes or correlated factors that may be of use to the analyst who is diagnosing the system. In many practical applications, it is especially important to distinguish between benign faults (due to unintentional causes, such as natural wear-and-tear of physical components) and malicious faults (due to intentional cases, such as illegitimate intrusions into a secure network system), since the nature of the fault directly affects the type of recovery actions to be initiated by the analyst. In general, anomaly detection is a statistical learning problem, in which the task is to train a classifier with knowledge of *normal* behavior to distinguish between abnormal or *anomalous* behavior.

Anomaly detection is a broad research topic that has inspired a long history of innovation from different research communities (e.g., signal processing, machine learning, statistics). Most algorithms that stemmed from this work have either been custom-tailored to specific domains or have restrictive assumptions. In general, it has been realized in practice that anomaly detection is an extremely challenging task, where different paradigms of detection schemes have been shown to perform well on different data. Thus, it is difficult and almost impossible to choose one single method to address the myriad of challenges faced by general real-world applications.

The difficulty of these challenges is strongly correlated with the statistical properties of the data, as well as the amount of information that is available about the domain. In particular, the applicability of a particular approach will depend on the following:

- The availability of domain knowledge about the system's behavior under normal and anomalous operating modes, i.e., is there enough information to build models of the system under different modes?
- The availability of data and whether they are labelled according to the corresponding modes of system behavior, i.e., is there enough data to learn the adequate models in the dearth of domain knowledge?

- The applicability of domain knowledge over time, i.e., how reliable is this knowledge and how stringent is the need to update our models over time?

Depending on the various degrees of available data and domain-specific knowledge, different methods have been applied to tackle the problem of anomaly detection. These methods are categorized into two main classes: *discriminative* approaches and *generative* approaches.

In discriminative methods, the focus is to optimize a decision rule that classifies data into categories that correspond to normal or abnormal modes of system behavior. No effort is made in trying to model the causal relationships between the data and the underlying system process. On the other hand, the focus of generative methods *is* to learn a model that describes the system process. With a generative model, one can interpret the system and understand the causality between the hidden system state and its observed behavior. This is in contrast to discriminative methods, which treat the underlying system as a black box. However, since parameters for generative models are often chosen to maximize the likelihood of the data, these models will generally be less optimized for the classification task (of anomaly detection) at hand. Nonetheless, depending on the specific application, one approach may be better suited to a particular domain, as we will see in later sections.

In Section 3, we lay the mathematical foundations for the theory behind discriminative and generative classifiers, and describe their qualitative differences. In Section 4, we explain a popular subset of methods that fall under the category of discriminative approaches and provide references for interesting applications that utilize such methods. In Section 5, we do the same for generative approaches. Finally, we summarize the tradeoffs between the two approaches and provide references to hybridization attempts in Section 6. For the rest of this survey, the terms *anomaly detection* and *novelty detection* will be used interchangeably.

2 Notation

In this section, we introduce the notation that will be used in our discourse. We use uppercase letters to denote random variables and lowercase letters to denote their instantiations. For example, given a binary variable $X \in \{0, 1\}$, X can either take on the value $x = 0$ or $x = 1$.

We use boldface when referring to a collection or set of similar items. For example, given d variables $\{X_1, ..., X_d\}$, the collection of these variables is referred to as $\mathbf{X} = \{X_1, ..., X_d\}$. We also use boldface for vectors, as vectors are usually the collection of more than one element.

In general, superscripts are often used to index a specific data point from a collection of data points. For example, a set of training data may consist of N data vectors, $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}\}$. The n^{th} data vector is denoted by $\mathbf{x}^{(n)}$ and its i^{th} element is denoted by $\mathbf{x}_i^{(n)}$. Note that $\mathbf{x}_i^{(n)}$ is not represented by a boldface letter because it is a single element instead of a vector.

In addition, we use $p(\cdot)$ to denote probability densities and $P(\cdot)$ to denote probability mass functions.

3 Discriminative vs. Generative

Anomaly detection is closely related to classification [Steinwart *et al.*, 2005]. In fact, one can define the problem of anomaly detection as the act of classifying data into the various categories that correspond to normal and abnormal modes of system behavior. As a result, we will examine the differences between discriminative and generative approaches in terms of their classification capabilities. As such, we will lay the mathematical theory of these two approaches in the setting of supervised classification.

In supervised classification, the input features is represented by the random vector $\mathbf{X}$ and its output label is represented by the random variable C. While $\mathbf{X}$ can be real- or discrete-valued, C is assumed to be discrete and takes on finite values that correspond to the different classes. $\mathbf{X}$ and C are derived from an unknown probability distribution $p(\mathbf{X}, C)$. Generative classification takes the approach of approximating $p(\mathbf{X}, C)$ using a parametric family of models, then applying Bayes' rule to compute the class-conditional distributions $P(C|\mathbf{X})$. Each new data vector $\mathbf{x}$ is then assigned to the most probable label c in respect to $P(C|\mathbf{X})$. The complementary approach of discriminative classification is to directly find a classification rule with the smallest error rate. In other words, this approach

learns $P(C|\mathbf{X})$ from the data without first estimating the joint distribution $p(\mathbf{X}, C)$. The obvious difference with the discriminative approach is that it makes no assumption about the input distribution $p(\mathbf{X})$, while the generative approach makes indirect assumption about $\mathbf{X}$ in its computation of the joint distribution $p(\mathbf{X}, C)$ before computing the conditional distribution $P(C|\mathbf{X})$. Put another way, the key difference is as follows: discriminative approaches apply $P(C = k|\mathbf{X} = \mathbf{x})$ to directly *discriminate* the value k for any instance $\mathbf{x}$, while *generative* approaches estimate $P(C = k|\mathbf{X} = \mathbf{x})$ from $P(C = k)$ and $p(\mathbf{X} = \mathbf{x}|C = k)$, the latter of which can be used to *generate* random instances $\mathbf{x}$ conditioned on a target label k.

We now examine more closely the mathematical relationship between discriminative and generative classifiers and we follow the discourse from [Bouchard and Triggs, 2004]. Assume that the training data, $\{\mathbf{x}^{(n)}, c^{(n)}\}_{n=1}^{N}$ where $\mathbf{x}^{(n)} \in \mathbb{R}^d$ and $c^{(n)} \in \{1, ..., K\}$, are independent and identically distributed according to some unknown distribution $p(\mathbf{X}, C)$. The goal is to compute $P(C|\mathbf{X})$, which would be used to devise a classification rule that categorizes new data with the least amount of error. To do so, one must compute the class-conditional probability $P(C = k|\mathbf{X})$ for each class k. For each class k, $p(\mathbf{X}|C = k)$ is modelled by some distribution f_k with parameters θ_k, and $P(C = k)$ is parametrized by the prior probability p_k. Altogether, the parameters for the joint distribution are $\Theta = \{p_1, ..., p_K, \theta_1, ..., \theta_K\}$. Assuming Θ is known, the classification task boils down to assigning the new data vector $\mathbf{x}$ to the class k that maximizes

$$P_\Theta(C = k|\mathbf{X} = \mathbf{x}) = \frac{p_k f_k(\mathbf{x}; \theta_k)}{\sum_{c=1}^{K} p_c f_c(\mathbf{x}; \theta_c)} \tag{1}$$

Both generative and discriminative methodologies take this same high-level approach. Their departure from one another lies in the estimation of Θ.

Given data $\{\mathbf{x}^{(n)}, c^{(n)}\}_{n=1}^{N}$, the parameters of a generative classifier are chosen to maximize the likelihood of the data, as follows:

$$\hat{\Theta}_{Gen} = \arg\max_{\Theta} \mathcal{L}_{Gen}(\Theta) \quad \text{where} \quad \mathcal{L}_{Gen}(\Theta) = \sum_{n=1}^{N} \log p_{c^{(n)}} f_{c^{(n)}}(\mathbf{x}^{(n)}; \theta) \tag{2}$$

In contrast, the parameters to a discriminative classifier are chosen to minimize the classification loss, which is approximated by $-\mathcal{L}_{Disc}$, as follows:

$$\hat{\Theta}_{Disc} = \arg\max_{\Theta} \mathcal{L}_{Disc}(\Theta) \quad \text{where} \quad \mathcal{L}_{Disc}(\Theta) = \sum_{n=1}^{N} \log \frac{p_{c^{(n)}} f_{c^{(n)}}(\mathbf{x}^{(n)}; \theta)}{\sum_{k=1}^{K} p_k f_k(\mathbf{x}^{(n)}; \theta)} \tag{3}$$

Once $\mathcal{L}_{Disc}$ is expanded, one can easily see its relationship to $\mathcal{L}_{Gen}$:

$$\mathcal{L}_{Disc}(\Theta) = \underbrace{\sum_{n=1}^{N} \log p_{c^{(n)}} f_{c^{(n)}}(\mathbf{x}^{(n)}; \theta)}_{\mathcal{L}_{Gen}(\Theta)} - \underbrace{\sum_{n=1}^{N} \log \sum_{k=1}^{K} p_k f_k(\mathbf{x}^{(n)}; \theta)}_{\mathcal{L}_{\mathbf{X}}(\Theta)} \tag{4}$$

Thus, the difference between $\mathcal{L}_{Disc}$ and $\mathcal{L}_{Gen}$ is $\mathcal{L}_{\mathbf{X}}$, which represents the log-likelihood of the probability model over the input space $\mathbf{X}$. This explains the fact that generative models tend to be biased towards those that maximize the likelihood of training data while the discriminative models are free from bias errors due to any misrepresentation of the input distribution $p(\mathbf{X})$.

To further illustrate the different flavors of the two approaches, we present a well-studied discriminative-generative pair of classifiers: naive Bayes and logistic regression. (For details, see [Mitchell, 2005].) Here, we assume that the parameters to the classifiers are already estimated, based on the procedure described above, and our goal is to show the different classification rules associated with each classifier. The parameters to the Naive Bayes classifier are the estimates to the distributions $P(C)$ and $p(\mathbf{X}|C)$, while the parameters to the logistic regression classifier are the weights $\{w_m\}_{m=0}^{d}$.

For simplicity, we assume that there are only two classes, i.e., $C \in \{0, 1\}$. Given a new input vector $\mathbf{x}^{new} = \{x_1, ..., x_d\}$, the naive Bayes classifier will assign $\mathbf{x}^{new}$ to the label c^{new} that satisfies

$$c^{new} \leftarrow \arg\max_{k} P(C = k) \prod_{i=1}^{d} p(X_i = x_i|C = k) \tag{5}$$

while logistic regression will assign $\mathbf{x}^{new}$ to $c^{new} = 0$ if

$$\overbrace{\frac{1}{1+\exp(w_0+\sum_{i=1}^{d} w_i x_i)}}^{P(C=1|\mathbf{X}=\mathbf{x}^{new})} \quad < \quad \overbrace{\frac{\exp\left(w_0+\sum_{i=1}^{d} w_i x_i\right)}{1+\exp\left(w_0+\sum_{i=1}^{d} w_i x_i\right)}}^{P(C=0|\mathbf{X}=\mathbf{x}^{new})} \tag{6}$$

which boils down to

$$1 \quad < \quad \exp\left(w_0+\sum_{i=1}^{d} w_i x_i\right), \tag{7}$$

and to $c^{new} = 1$ otherwise. Comparing the two, one can see that the classification rules are drastically different and thus this illustrates the difference in approach between generative and discriminative classifiers.

In general, the generative approach will learn the best model for the joint distribution $p(\mathbf{X}, C)$ but its conditional distribution $p(C|\mathbf{X})$ will result in a biased classifier unless an accurate model of $p(\mathbf{X})$ is used. Since the true distribution $p(\mathbf{X})$ is rarely known, the generative model will result in some degree of bias and thus a discriminative classifier is generally believed to be superior to its generative counterpart. However, this long-held belief is only partly true, as [Ng and Jordan, 2001] show in their empirical comparison of naive Bayes to logistic regression. Their study confirms that the naive Bayes model indeed has a higher asymptotic error than logistic regression, but it also reveals an interesting discovery: the Bayes model converges to its steady-state parameters at a much faster rate. If d is the dimension of the input vector, the naive Bayes model converges with O $(\log d)$ number of training examples, while logistic regression requires O (d) number of training examples for convergence. Thus, this suggests an optimal classification policy whereby one should first apply the generative model, then switch to the discriminative model when there are sufficient data for the discriminative learning to converge to a model of lower asymptotic error.

Theoretical results and empirical results are given in [Ng and Jordan, 2001] to support this hypothesis. The experiments were performed on 15 datasets from the UCI Machine Learning repository [Newman *et al.*, 1998]. Figure 1 shows the empirical results, where the asymptotic classification error is plotted against the number of training examples. Eight datasets are with continuous inputs and seven are with discrete inputs, as labelled in the figure. It was found that in most cases, the naive Bayes model did converge faster but to a model of higher asymptotic error, compared to the logistic regression model. The few exceptions to this observation were small datasets that did not have enough training examples for logistic regression to converge to its optimal model of lower asymptotic error.

4 Discriminative approaches

In many aspects, discriminative approaches can be interpreted as function fitting. Given $\mathbf{X}$, discriminative approaches aim to learn a direct mapping from the input $\mathbf{X}$ to the output label C, either through the direct estimation of the class-conditional probability distribution $P(C|\mathbf{X})$ or through other methods that achieve minimal classification error. The advantage of discriminative classifiers is that they concentrate on finding the decision boundary that separates the various classes of normal and anomalous system behaviors. As a result, compared to generative classifiers, discriminative classifiers are usually more robust against outliers in the data. But as a result of this focus (solely on the decision boundary), the rest of the space is generally ignored. Thus, discriminative approaches offer much less insights about the structure of the underlying system, which makes it difficult for discriminative approaches to deal with missing data.

Popular discriminative approaches include logistic regression, linear/quadratic/regularized discriminant analysis, random forests, distance-based discrimination, support vector machines and traditional neural networks. In particular, this section focuses on the last three methods. For each method, we will provide a brief explanation and present a subset of the current literature that is especially relevant to anomaly detection.

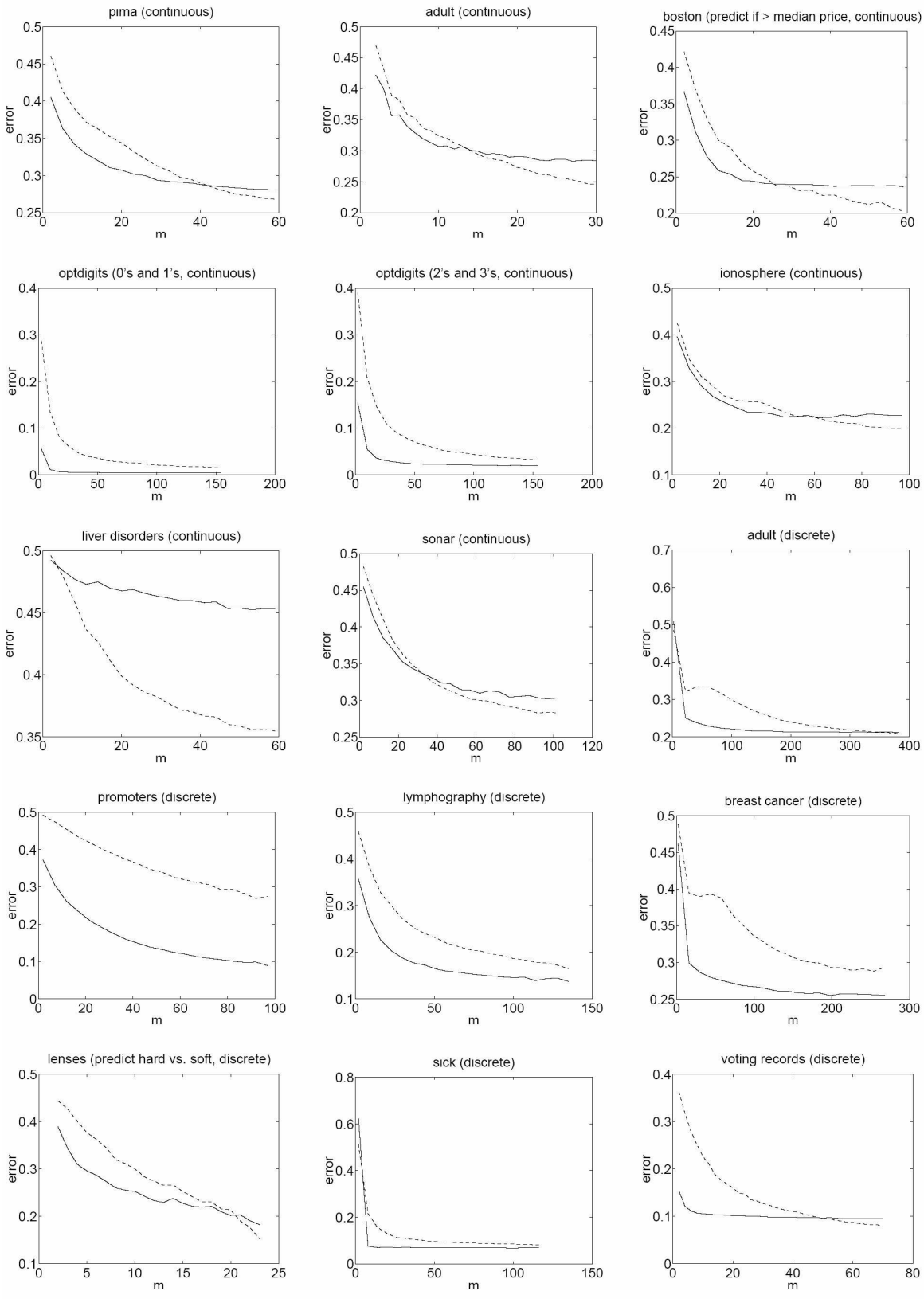

Figure 1: A comparison of the generalization error from the naive Bayes model and the logistic regression model, as a function of m, the number of training examples. The results for logistic regression are shown as dashed lines while the results for naive Bayes are shown as solid lines. Reproduced from [Ng and Jordan, 2001].

4.1 Distance-based discrimination

In this subsection, we will introduce a variety of classifiers and outlier detectors that uses the notion of spatial distance to discriminate between normal and anomalous feature vectors. (Normal feature vectors are feature vectors that correspond to normal classes; anomalous feature vectors are defined similarly.) The distinction between classifiers and outlier detectors is subtle, but primarily, classifiers are trained in a supervised learning setting, where the training data consist of labelled instances, i.e., $\{\mathbf{x}^{(n)}, c^{(n)}\}_{n=1}^{N}$, while outlier detectors may employ clustering or dimension reduction techniques that are trained in an unsupervised learning setting, where the training data are unlabelled, i.e., $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$.

In addition, many outlier detection schemes are based on the following two assumptions about the training data: The first is that the training data contains a large portion of normal feature vectors. The second assumption is that the anomalous feature vectors can be qualitatively distinguished from the normal feature vectors. With these two assumptions of rarity and deviation from normal characteristics, the anomalous feature vectors can be treated as outliers, and thus outlier detection algorithms can be used to detect anomalies.

4.1.1 Nearest neighbor

The nearest neighbor classifier is one of the most commonly used methods for anomaly detection. The intuition behind this algorithm is simple: feature vectors that are close together, in respect to some distance metric, belong to the same class. The nearest neighbor classifier assumes labelled training data $\{\mathbf{x}^{(n)}, c^{(n)}\}_{n=1}^{N}$ and assigns the new instance $\mathbf{x}^{new}$ to the same class as its closest neighbor. A popular generalization of this method is the k nearest neigbor (kNN) classifier, where the k nearest neighbor to $\mathbf{x}^{new}$ are used to determine its class c^{new}. One way of determining c^{new} is by majority vote, where c^{new} is assigned to the most common class among the k neighbors. Another way is to weigh each neighbor's vote as a function of its distance to $\mathbf{x}^{new}$, so that closer neighbors may have higher influence on the vote than farther neighbors. In this weighted majority voting scheme, the weights corresponding to each distinct class are summed together and c^{new} is assigned to the class with the maximum weight.

The kNN approach has been applied with success to anomaly detection in [Liao and Vemuri, 2002]. In this work, the kNN classifier was used to detect network intrusions from traces of program behavior. This work leverages existing work in text categorization, and translates a program behavior into a text format, where kNN can be used to classify between normal or intrusive behavior. Specifically, the approach treats each system call as a "word" and a collection of system calls throughout program execution as a "document". The study first trained the kNN classifier using simulated data that were free of attacks, in order to characterize normal behavior. A new instance $\mathbf{x}^{new}$ is characterized as an anomaly (associated with an intrusive attack) if the average distance of its k nearest neighbors falls above a given threshold. Experiments were performed on the 1998 DARPA Intrusion Detection System Evaluation data [Lincoln Laboratory, 1998], which include a large sample of computer attacks embedded in normal background traffic. For a given k, the performance of kNN is measured using the Receiver Operating Characteristic (ROC) curve, which plots the intrusion detection accuracy as a function of the false positive probability. Figure 2 shows the performance of KNN for different values of k.

For small k, the runtime for kNN is $\mathrm{O}(N)$ where N is the number of computer processes in the training data. As a result, kNN may not be efficient when N is large. To improve upon kNN , it may be advantageous to combine kNN with *signature verification*, which establishes a set of rules or properties that correspond to a particular class. The improved version of kNN learns new classes that correspond to a subset of known malicious program behavior. Table 1 shows its effectiveness for detecting novel malicious behavior.

A nice theoretical property of the nearest-neighbor method is that, as the number of training examples tends to infinity, the error rate is never worse than twice the Bayes rate [Cover and Hart, 1967]. Despite this useful property, the use of nearest neighbor is not always meaningful, as shown theoretically and empirically in [Beyer *et al.*, 1999]. In general, one must make sure that the data is spatially distributed in such a way that there is a clear distinction between the nearest and the farthest neighbors for any typical input feature vector $\mathbf{x}^{new}$. (In some literature, $\mathbf{x}^{new}$ may also be

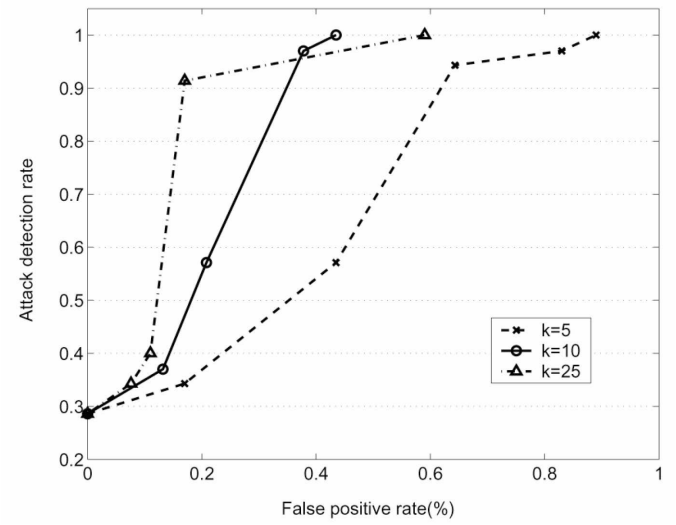

Figure 2: Performance of kNN as ROC curves, which display the false positive rate vs. the attack detection rate for different values of k. Reproduced from [Liao and Vemuri, 2002].

Table 1: Attack detection rate for the DARPA data when kNN is combined with signature verification. Reproduced from [Liao and Vemuri, 2002].

Attack type	**Instances**	**Detected**	**Detection rate**
Known attacks	16	16	100%
Novel attacks	8	6	75%
Total	24	22	91.7%

referred to as the *query* vector.) As dimensionality increases, $\mathbf{x}^{new}$'s distance to its nearest neighbor typically approaches the distance to its farthest neighbor, in as few as 10-15 dimensions.

This phenomenon is confirmed by empirical results that are shown in Figure 3, where the average ratio of the farthest neighbor's distance ($DMAX$) to the closest neighbor's distance ($DMIN$) is plotted as a function of the data's dimension (i.e., each training vector is m-dimensional). The average is taken over 1000 query instances on synthetic data sets of one million tuples. The data sets are generated by different probability distributions. The line corresponding to "uniform" shows the performance on a uniformly distributed data set. Similarly, the line corresponding to "recursive" shows the performance for a data set where every pair of dimensions is correlated and every new dimension has a larger variance. Lastly, the line corresponding to "two degrees of freedom" shows the performance for a data set generated from the weighted sum of two uniformly distributed random variables. For $m = 1$, the ratio $\frac{DMAX_m}{DMIN_m} \approx 10^7$, which provides quite a contrast between the closest neighbor and the farthest neighbor. But as m is increased, the contrast becomes insignificant, as seen by the reduction in $\frac{DMAX_m}{DMIN_m}$'s orders of magnitude.

To increase the effectiveness of the nearest-neighbor method for high dimensions, an interactive system was proposed in [Aggarwal, 2002]. This work describes a human-computer interactive system for high-dimensional nearest neighbor search, whereby the high-dimensional training data are projected onto carefully chosen lower-dimensional representations, with the hopes that these lower-dimensional representations better capture meaningful relationships between the training data and the query vector $\mathbf{x}^{new}$. Projections are chosen based on how well the projection distinguishes the (lower dimensional) clusters containing $\mathbf{x}^{new}$ from the rest of the data. After the computer finds these projections, these projections are presented visually to the user, who can then express his or her preferences about these projections so that the meaningfulness of $\mathbf{x}^{new}$'s nearest neighbors can be tailored from the user's perspective. The motivation for this approach is that repeated feedback

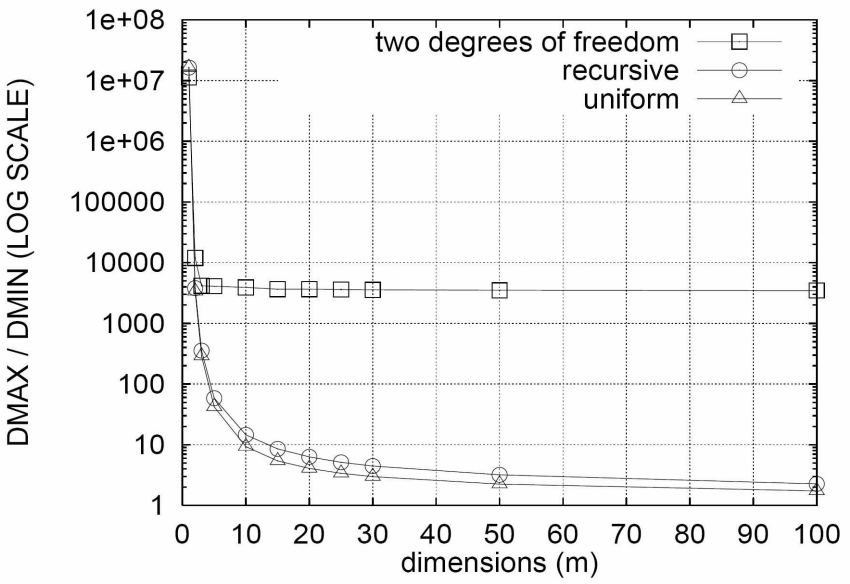

Figure 3: Performance of nearest neighbor for different spatially distributed sets of data. For each distribution, the ratio of the farthest neighbor's distance to the closest neighbor's distance is plotted as a function of the data's dimension. Reproduced from [Beyer *et al.*, 1999].

from the user over several iterations should allow the system to find a set of statistically significant and meaningful neighbors.

This interactive system was tested on a number of real data sets from the UCI machine learning repository [Newman *et al.*, 1998]. In particular, a comparison was made between the proposed interactive nearest-neighbor algorithm and the standard (full-dimensional) nearest-neighbor algorithm, on the ionosphere and segmentation data sets from the UCI repository. The experiment measured the nearest neighbor classification accuracy for 10 query vectors. The experimental results are shown in Table 2, where the performance of the interactive system is shown to be clearly superior.

Table 2: Classification accuracy for the standard nearest neighbor algorithm and the proposed interactive version of the nearest neighbor algorithm. Reproduced from [Aggarwal, 2002].

Data set (dimensionality)	**Accuracy** (Standard NN)	**Accuracy** (Interactive NN)
Ionosphere (34)	71%	86%
Segmentation (19)	61%	83%

4.1.2 Distance-based outlier detectors

Alternatively, one can avoid the need for labelled training data by using outlier detection methods. In these methods, the anomalies are treated as outliers to the training data and are identified purely by their relative spatial location to the other vectors in the training data.

In [Ramaswamy *et al.*, 2000], the distance of a feature vector to its k^{th} nearest neighbor is used to define the notion of distance-based outliers. In this framework, each vector in the data set is ranked on the basis of its distance to its k^{th} nearest neighbor and the m highest ranked vectors are identified as outliers. This heuristic makes intuitive sense because the vectors that are ranked the highest will not be clustered as densely as those in the lower ranks, and thus the highest ranked vectors are outliers relative to the rest of data.

Let $D^k(\mathbf{x})$ denote the distance from $\mathbf{x}$ to its k^{th} nearest neighbor. While standard algorithms (such as nested-loop and index-based algorithms) can be used to compute $D^k\left(\mathbf{x}^{(n)}\right)$ for each $\mathbf{x}^{(n)}$ in the data

set $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$, these algorithms are computationally expensive, requiring as much as O $\left(N^2\right)$ computations. To address this inefficiency, a partition-based algorithm is also presented in [Ramaswamy *et al.*, 2000]. This partition-based algorithm employs a divide-and-conquer approach, whereby it partitions the data set into disjoint subsets then prunes entire partitions that are determined to be free of outliers. Thus, much fewer computations for $D^k(\mathbf{x})$ are needed, resulting in substantial speedup in runtime. The standard algorithms and the proposed partition-based algorithm were tested on a synthetic data set that contained 100 hyper-spherical clusters of uniformly distributed data, along with 1000 uniformly scattered outliers. In Figure 4, the runtime for each algorithm is plotted against the number of data vectors N, the neighbor index k, and the dimension of the data vector.

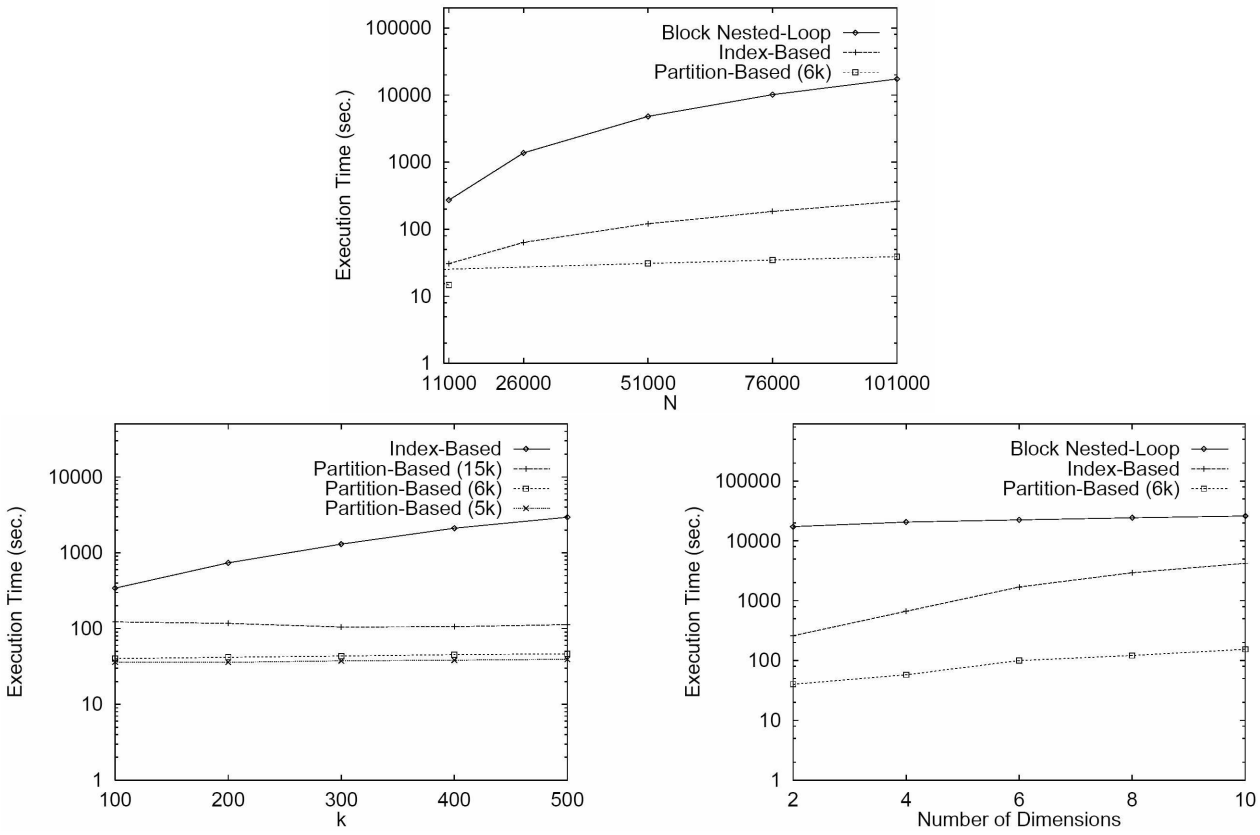

Figure 4: Comparison of the nested-loop, the index-based, and the proposed partition-based algorithms. The runtime for each algorithm is plotted against the number of data vectors N, the neighbor index k, and the dimension of the data vector. In the lower left plot, the nested-loop method was too slow to be competitive with the other algorithms and was therefore omitted. Instead, the partition-based algorithm was ran with different number of partitions and the results were presented instead. Reproduced from [Ramaswamy *et al.*, 2000].

From Figure 4, one can see that the partition-based algorithm is much faster than the standard algorithms, and scales well with respect to both the size and dimension of the data set. In addition to this set of experiments on synthetic data, the partition-based algorithm was also tested on a real-life NBA (National Basketball Association) database, where particular players were flagged as outliers, due to their dominance by a wide margin in a particular gaming aspect.

In the same spirit of divide-and-conquer, [Knorr *et al.*, 2000] presents a similar approach whereby a data vector is identified as an outlier if at least a fraction f of the data set is located greater than a distance D away. Such an outlier is denoted as a $\mathrm{DB}(f, D)$ outlier. Again, the two simple algorithms of index-based and nested-loop approaches were presented for finding the $\mathrm{DB}(f, D)$ outliers. To find all $\mathrm{DB}(f, D)$ outliers in a data set, both algorithms have a worst-case complexity of O (dN) where d is the dimensionality and N is the size of the data set. An optimized cell-based algorithm, that scales linearly with N but exponentially with d, is presented. The idea is similar to that employed in

[Ramaswamy *et al.*, 2000], where data vectors are partitioned into cells and outliers are determined on a cell-by-cell basis, rather than on a vector-by-vector basis. This approach allows rapid pruning of a large number of data vectors that cannot be outliers, which results in significant reduction in runtime. Experimental results indicate that this cell-based approach outperforms the index-based and nested-loop approaches for $d \leq 4$. This method was also applied to three real-life applications, which include analysis of NHL (National Hockey League) statistics, spatio-temporal trajectories from surveillance videos, and workers' compensation employer performance data. Figure 5 shows a subset of the results from the case study on detecting outliers from surveillance videos. The outliers were determined based on their differences in speed and/or trajectories between a pedestrian's entry and exit points. The results show that the idea of DB outliers can be applied with success to detect anomalies in spatio-temporal data.

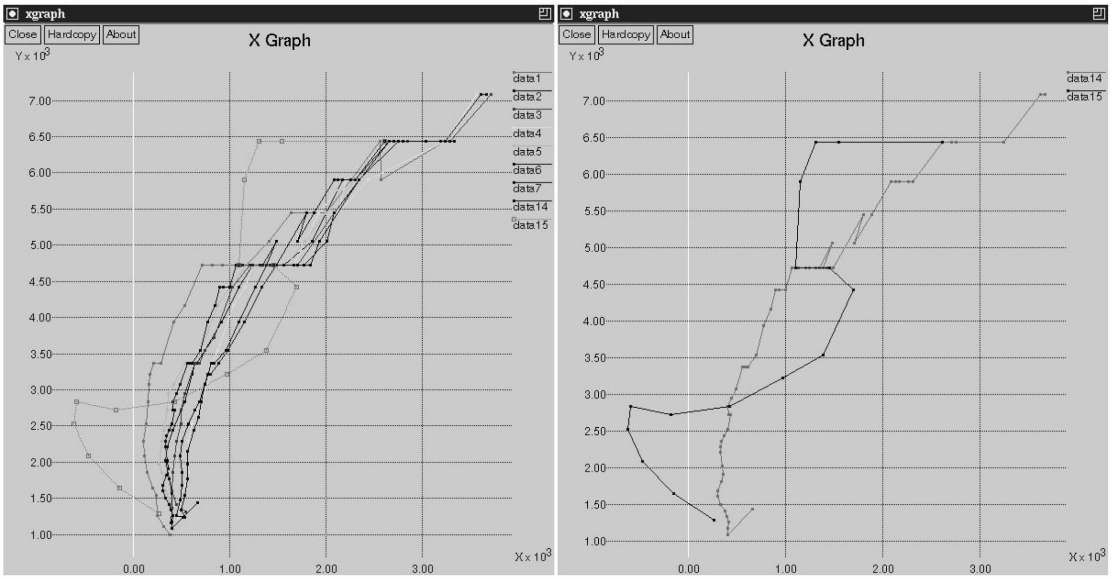

Figure 5: The results of anomaly detection in spatio-temporal data from surveillance videos. The left plot shows the entire data set. The right plot shows the anomalous trajectories detected using the method of DB outliers. Reproduced from [Knorr *et al.*, 2000].

So far, we have only examined algorithms that treat the state of being an outlier as a binary property, in that, a feature vector $\mathbf{x}$ is an outlier with 0% probability or 100% probability. But in some scenarios, it may be more meaningful to attribute $\mathbf{x}$ with the *degree* of being an outlier instead. Such an approach was taken in [Breunig *et al.*, 2000], in which a new outlier detection approach, based on the notion of the *local outlier factor* (LOF), was proposed. The LOF measures the degree to which each data vector $\mathbf{x}$ is an outlier, dependent on how isolated $\mathbf{x}$ is with respect to its surrounding neighborhood. In contrast to the k-nearest-neighbor algorithm, the LOF approach utilizes the density of other points around $\mathbf{x}$ rather than just the distances from $\mathbf{x}$ to its k closest neighbors.

The LOF method is implemented as a two-step algorithm. For each data vector $\mathbf{x}$, the first step finds all neighboring vectors that are within distance $D^k(\mathbf{x})$ from $\mathbf{x}$, and stores their actual distance from $\mathbf{x}$ in a database. The second step computes the LOFs from this database. The complexity of the first step is implementation-dependent and was reported to be $\mathrm{O}\,(N \log N)$ in [Breunig *et al.*, 2000]'s implementation and the complexity of the second step is $\mathrm{O}\,(N)$, where N is the number of vectors in the data set.

The LOF algorithm was tested on a synthetic 2-dimensional data set and two real-life sports-related data sets, one on hockey and another on soccer. The results for the synthetic data set are shown in Figure 6, where a clear graphical view for all the computed LOF values is presented. Empirical results for the real-life data sets also show that the LOF method can find meaningful outliers that are otherwise undetected by existing approaches. This observation is confirmed by a comparison study by [Lazerevic *et al.*, 2003], where popular outlier detection methods, including the LOF method,

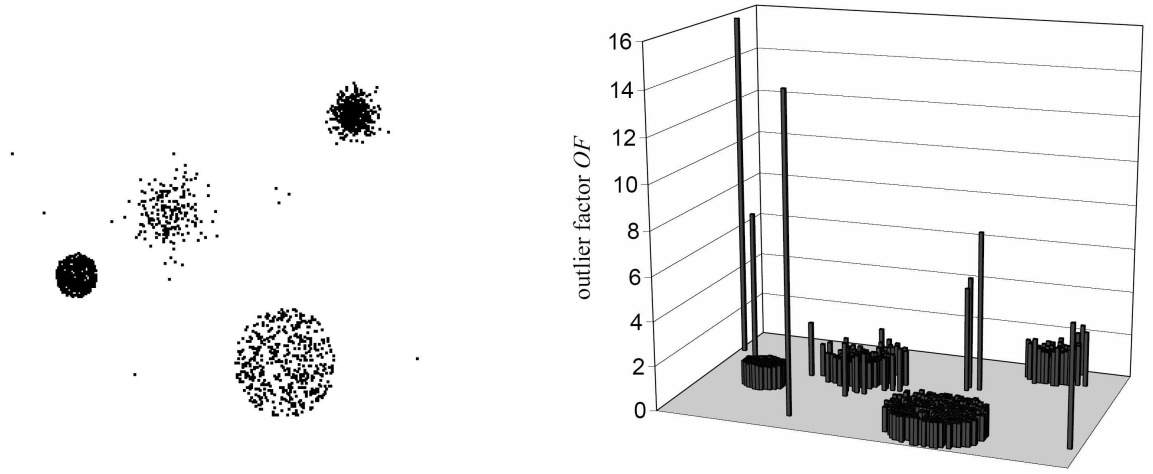

Figure 6: The local outlier factors for the data in the synthetic data set. Reproduced from [Breunig *et al.*, 2000].

were evaluated on the 1998 DARPA Intrusion Detection System Evaluation data set [Lincoln Laboratory, 1998]. In this study, LOF was compared against:

- Nearest-neighbor: A feature vector is an outlier if the distance to its nearest neighbor exceeds a given threshold.
- Mahalanobis-distance-based: The mean and standard deviation for the training data is computed. A feature vector is an outlier if the Mahalanobis distance to the mean of the training data exceeds a given threshold.
- Unsupervised support vector machines: To be explained in Subsection 4.2

The results are presented as ROC curves in Figure 7, which shows that LOF outperforms all other methods in detecting network intrusions for the DARPA data set.

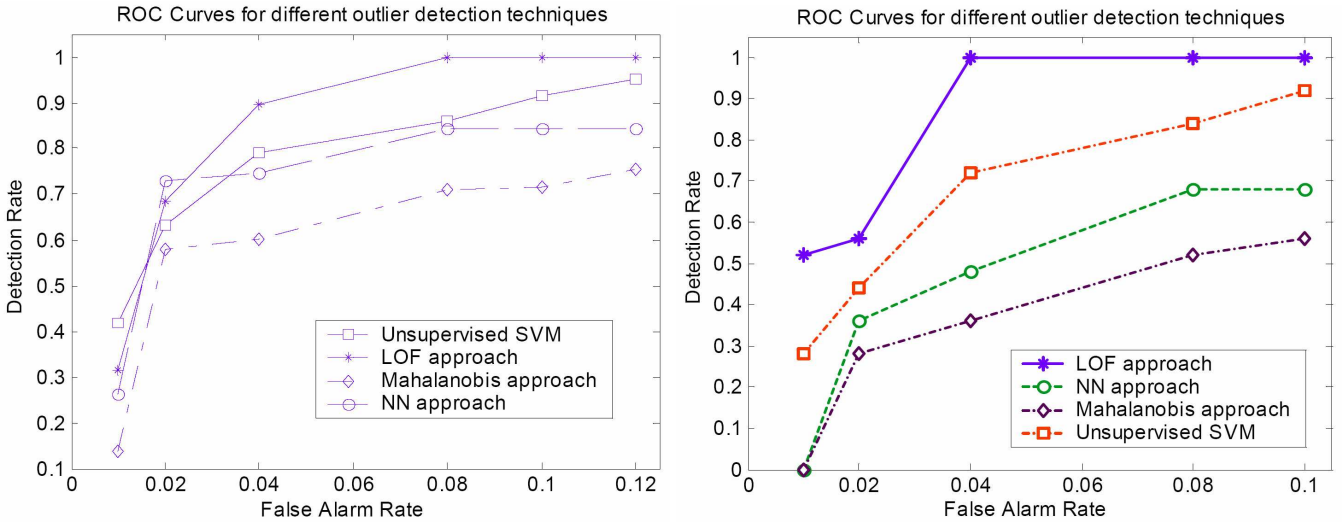

Figure 7: Comparisons of different anomaly detection algorithms on bursty attacks (right) and on single-connection attacks (left). Reproduced from [Lazerevic *et al.*, 2003].

4.2 Support vector machines

Aside from distance-based methods, support vector machines have also been widely used for anomaly detection, especially in the areas of intrusion detection and medical diagnosis. In this subsection, we will first explain the supervised version of support vector machines, then briefly discuss its unsupervised counterpart through an application.

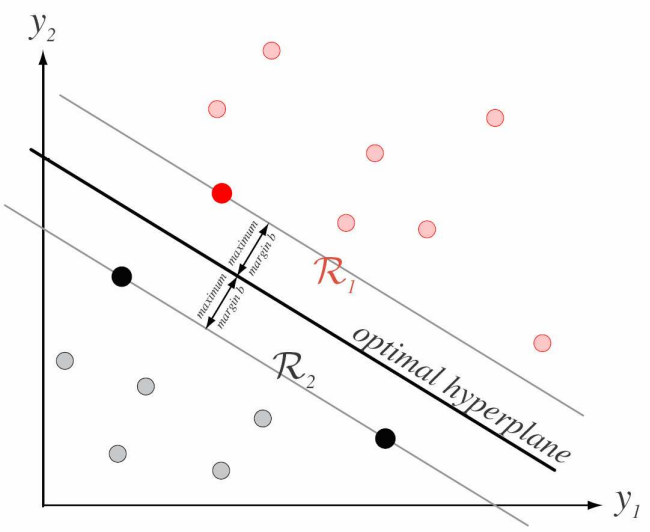

Figure 8: The goal of a SVM is to find the optimal hyperplane that has the maximal distance from the nearest training patterns. The support vectors (shown as solid dots) are those nearest patterns that are fixed distance from the optimal hyperplane. Reproduced from Richard O. Duda, Peter E. Hart and David G. Stork, *Pattern Recognition*. ©2001 by John Wiley & Sons, Inc.

The objective of a support vector machine (SVM) [Burges, 1998; Duda *et al.*, 2001] is to define a decision hyperplane that separates the different classes with the largest margin from the nearest training examples. The support vectors, as shown and defined in Figure 8, are the training examples that define the optimal hyperplane, which forms the perpendicular bisector of the support vectors. In essence, the support vectors aim to represent the most informative patterns that allow one to best distinguish between the different classes.

To define these support vectors, SVMs apply a transformation to the data so that the patterns represented by the data are linearly separable (i.e., can be separated by a hyperplane). This is possible because nonlinearly-separable patterns can always become linearly separable in a sufficiently high-dimensional representation. Thus, the data are mapped by an appropriate (non-linear) function to a higher dimension and optimization is performed to find the optimal separating hyperplane.

SVM variants include hard-margin SVMs for separable classes, soft-margin SVMs for non-separable classes and robust SVMs that generalize to noisy data. The ability to handle noisy data is important in any detection or classification setting, especially since noiseless or *clean* data may be difficult or expensive to obtain for real-world systems, where data may be be derived from noisy sensor readings or may be mislabelled due to human/machine error. In addition, for dynamic systems where normal behavior may change over time, it is especially important for an anomaly detection scheme to be able to handle noisy data, since the labels assigned to feature vectors during training may become unreliable.

In [Hu *et al.*, 2003], the standard and robust versions of SVM are compared to the k-nearest-neighbor classifier (kNN) [Liao and Vemuri, 2002] on clean and noisy data. The study used the 1998 DARPA Intrusion Detection System Evaluation data set [Lincoln Laboratory, 1998], where a clean data set and a noisy data set (see Table 3) were extracted for training and testing.

The detection results are shown in Figure 9, where the performance of the robust SVM, the standard SVM and the kNN classifier is expressed as ROC curves over the clean and noisy data. On the

Table 3: The clean and noisy data sets used in the [Hu *et al.*, 2003] study. Reproduced from [Hu *et al.*, 2003].

	Clean data	**Noisy data**
Training	300 normal processes	316 normal processes (16 mislabelled)
	28 intrusive processes	12 intrusive processes
Testing	5285 normal processes, 22 intrusive sessions	

clean data, the attack detection rate with zero false positive rate was 74.7% for robust SVM, 50% for standard SVM and 13.6% for kNN , while 100% attack detection rate was attained with a false positive rate of 3% for robust SVM, 14.2% for standard SVM and 8.6% for kNN . In particular, it appears that kNN performed the worst and the robust SVM performed the best. On the noisy data, the attack detection rate with zero false positive rate was 50% for robust SVM and 54% for standard SVM, while 100% attack detection rate was attained with a false positive rate of 8% for robust SVM and 100% for standard SVM (which is practically useless). The robust SVM shows very minor degradation in performance in the presence of noise, while SVM shows tremendous degradation. kNN's resilience to noise can be explained by the averaging that it performs on the test vector's k nearest neighbors, which allows it to smooth out the impact of the isolated noisy training examples. Nonetheless, if the training examples were incorrectly classified and the test vector happened to be one of these incorrectly classified training examples, then the kNN classifier would be unable to detect the intrusive attack. In general, this study shows that robust SVMs are quite well suited for anomaly detection in noisy data, because robust SVMs are not as prone to over-fitting the noise and they also lead to faster runtime, due to the reduced number of support vectors.

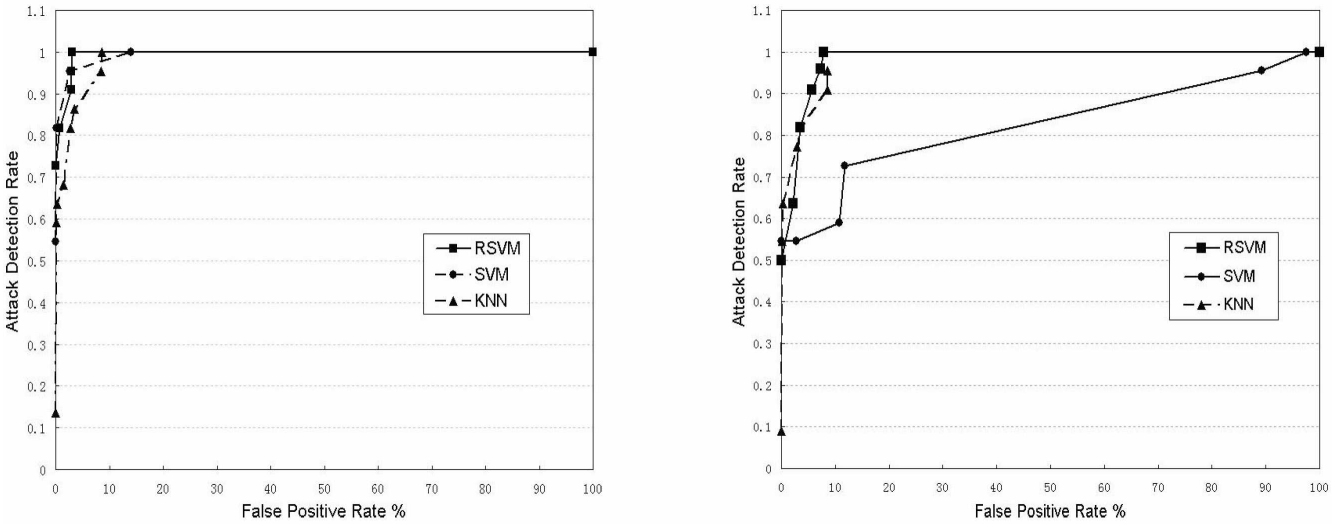

Figure 9: Performance of the robust SVM, the standard SVM and the kNN classifier is expressed as ROC curves over the clean data (left) and the noisy data (right). Reproduced from [Hu *et al.*, 2003].

The SVMs discussed so far are supervised methods, where one assumes the availability of labelled data $\{\mathbf{x}^{(n)}, c^{(n)}\}_{n=1}^{N}$ for training purposes. However, in the absence of labelled data or in the presence of highly unreliable labelled data, then *unsupervised* methods may be more desirable from a practical standpoint. An unsupervised version of SVMs has been proposed by [Schölkopf *et al.*, 2000]. The high-level idea of the unsupervised SVM is that it finds the region where the majority of the data lies and associates these data as one class. The complement of these data is then considered as belonging to a separate class. This algorithm was evaluated on the USPS (United States Postal Service) data set [Hull, 1994] of handwritten digits, which contains 9298 digital images of 256 pixels, in which the last 2007 images were used as the test set for this empirical study. The top 20 outliers are shown in Figure 10. Below each digital image, the italic number is the output of the

SVM and the boldface number is the class label assigned to the image. As one can see, these outliers correspond to atypical examples that are especially difficult to match to their representative digits.

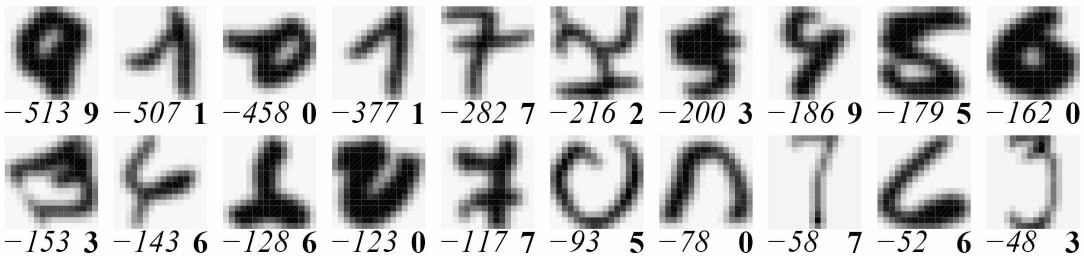

Figure 10: The outliers identified by the unsupervised SVM, ranked by the negative output of the algorithm. Reproduced from [Schölkopf *et al.*, 2000].

A comparative study on the effectiveness of the unsupervised SVM and other distance-based outlier detection algorithms in intrusion detection is presented in [Lazerevic *et al.*, 2003]. The results of this study are presented previously in Figure 7. Aside from intrusion detection systems, SVMs have also been applied with success to glaucoma diagnosis [Chan *et al.*, 2002].

4.3 Neural networks

A neural network [Haykin, 1998; Duda *et al.*, 2001] is a biologically-inspired method of computation based on an abstract representation of the brain. Analogous to the brain, which consists of a large number of highly interconnected network of neurons, a neural network consists of various units that are organized in layers to simulate the learning process of the brain.

Like the brain, a neural network learns by example, where each neural network is trained for a specific application through a learning process. This learning process can either be supervised or unsupervised. In supervised learning, a set of labelled data $\{\mathbf{x}^{(n)}, c^{(n)}\}_{n=1}^{N}$ is processed by the neural network. For each $\mathbf{x}^{(n)}$, the neural network compares its classification output c against the true label $c^{(n)}$, and uses this error to finetune its parameters accordingly. In contrast, unsupervised learning uses a set of unlabelled data $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$. The process by which a neural network self-organizes the data into different classes without the use of external labels is known as self-organization or adaptation. Generally, supervised learning is performed off-line and unsupervised learning is performed online. Neural networks are used extensively in anomaly detection, due to their success in pattern recognition and data classification. In this subsection, we focus on applications where the neural networks are learned in a supervised manner.

The basic unit of a neural network is referred to as a *neuron*, after the biological neuron that inspired its model. Each neuron has one basic function: to emit a response of the weighted sum of its inputs. The nature of the response depends on the *activation function* of the neuron. Each neuron can have a different activation function. But in practice, most neurons have the same activation function and it is often chosen to be the logistic function.

A neural network consists of an input layer, a variable number of hidden layers, and an output layer of neurons. Each layer can have a variable number of neurons, with the exception of the input layer, which is usually constrained to have as many neurons as the dimension of the input feature vector. The input layer takes as input the data, which is in turn processed by the hidden layer(s). The result of this hidden layer processing is then passed on to the output layer, which outputs the classification result. Figure 11 shows the basic structure of a three-layer neural network.

Theoretically, a three-layer neural network can implement any continuous function (either for density estimation or classification), given a sufficient number of hidden units and the proper model parameters. However, the question of choosing the optimal neural network structure for a particular problem still remains somewhat of an art. (See Figure 12 for a variety of decision boundary that one can implement with neural networks.) As a result, the design of a neural network for any non-trivial application may still require human experts who are adept in the art of neural networks.

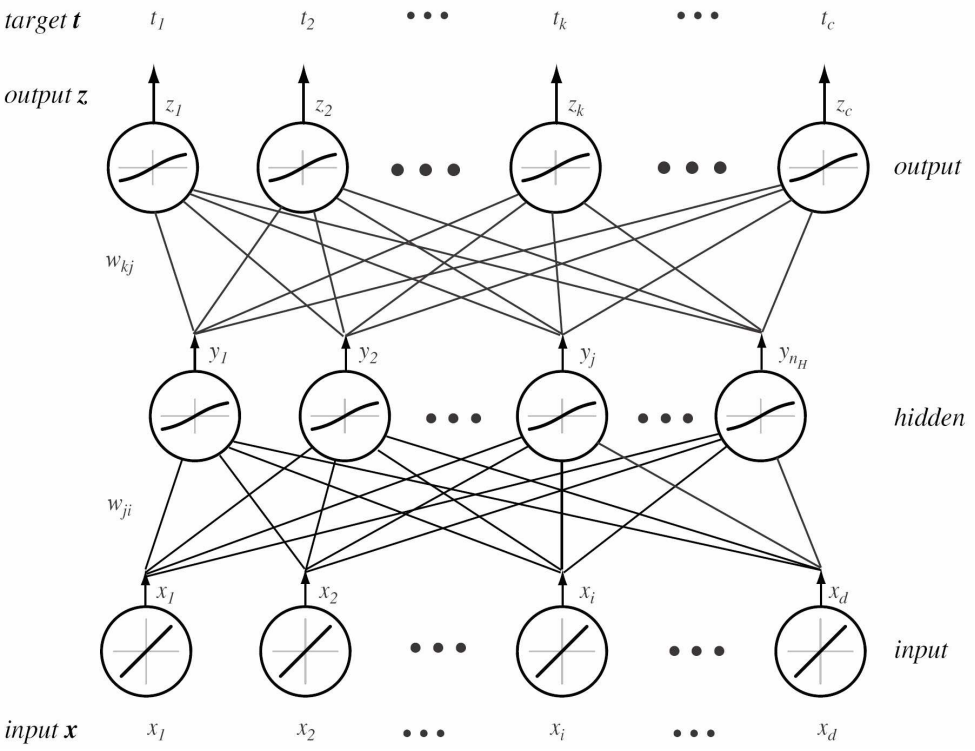

Figure 11: A fully-connected three-layer neural network. The input units represent the components of an input feature vector. The hidden units represent a black box that encapsulates the hidden relationships between the input feature vector and its corresponding output class. The output units represent the output of the discriminant function that determines the output class to which the input feature vector belongs. Reproduced from Richard O. Duda, Peter E. Hart and David G. Stork, *Pattern Recognition*. ©2001 by John Wiley & Sons, Inc.

Neural networks have been widely applied to anomaly detection. Popular applications include intrusion detection systems [Ryan *et al.*, 1998], handwriting recognition [LeCun *et al.*, 1990], image sequence analysis [Singh *et al.*, 2000; Markou and Singh, 2006] and medical diagnosis [Tarassenko, 1995; Chan *et al.*, 2002]. In the interest of space, we describe only a subset of this work.

In [Ryan *et al.*, 1998], a neural network was trained to detect network intrusions based on anomalous behaviors on the part of the individual users. This neural network intrusion detector, NNID for short, is trained to identify computer users based on the commands they issue during the day. At the end of each day, NNID is run to detect any anomalies in the uers' daily session. If anomalies are detected, then an investigation will be initiated to diagnose the cause for the anomalies. The NNID system is based on a three-layer neural network, in which the input layer consisted of 100 units, the hidden layer consisted of 30 units and the output layer consisted of 10 units, one for each of the ten users that partook in this experiment. The NNID system was built and tested on a machine at the University of Texas at Austin, where data were collected from this machine for 12 days, which resulted in 89 data vectors. NNID was trained on 8 randomly chosen days of data (65 data vectors) and tested on the remaining 4 days of data (24 data vectors). In an environment of 10 users, NNID exhibited a 96% detection accuracy rate with a false alarm rate of 7%. These results confirm NNID's promise as an offline monitoring system for intrusion detection.

In [LeCun *et al.*, 1990], a highly sophisticated neural network, consisting of six layers, were developed for classifying handwritten digits from the USPS (United States Postal Service) data set [Hull, 1994] of handwritten digits. This work leverages the idea that shape recognition can be improved by detecting and combining local features, and translates this idea into the architecture of the neural network by constraining the connections in the first few layers to be local, through the use of *feature maps*. Units on a feature map are constrained to perform the same operation on different parts of the

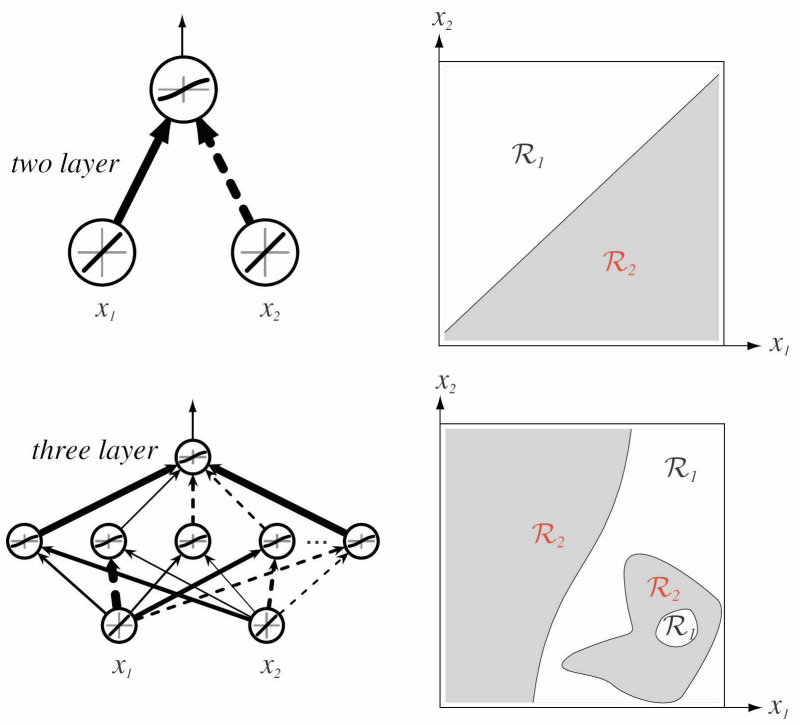

Figure 12: A two-layer neural network can only classify between two linearly separable classes. As the number of layers to a neural network is increased, an arbitrarily complex decision boundary can be formed, which can be used to classify between multiple nonlinearly separable classes. Reproduced from Richard O. Duda, Peter E. Hart and David G. Stork, *Pattern Recognition*. ©2001 by John Wiley & Sons, Inc.

image. Multiple feature maps extract different features from the same image, and are thus a necessary component of this neural network. The structure of the neural network is shown in Figure 13, where each hidden layer is labelled by an "H" label. Next to each "H" label, "$m@s \times s$" means that the hidden layer is composed of m groups of units, each group arranged in a s-by$s$ plane. Altogether,

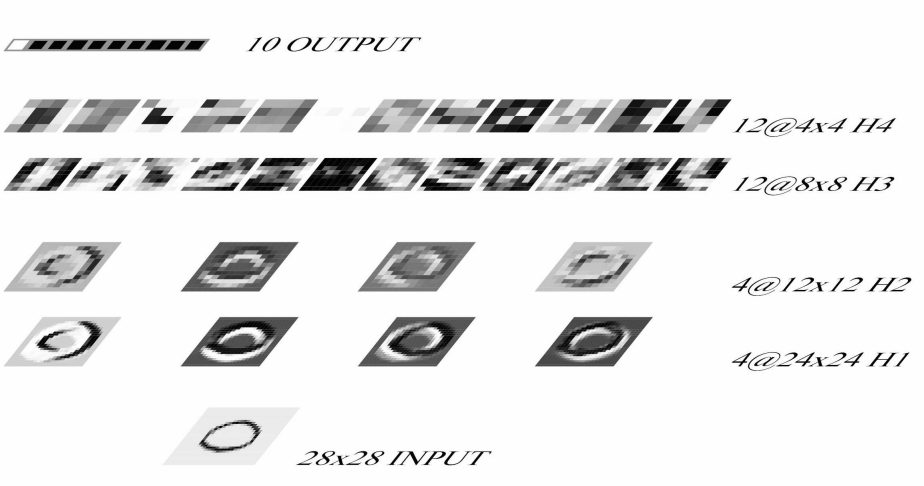

Figure 13: The architecture of the six-layer neural network used to classify handwritten digits from the USPS data set. Reproduced from [LeCun *et al.*, 1990].

the neural network contains 4635 units, 98442 connections and 2578 independent parameters. After 30 training passes on a training set of 7291 handwritten digits and 2549 printed digits, the neural network achieved an error rate of 1.1% and the MSE (mean of squared errors) of 0.017 on the training data. When tested on the test set of 2007 handwritten digits and 700 printed characters, the neural network achieved an error rate of 3.4% and the MSE of 0.024. The classification errors were solely due to the mislabelling of the handwritten characters.

In [Singh *et al.*, 2000], neural networks were used in conjunction with clustering to detect novel objects in video sequences. During a test run, the trained neural network processes the test vectors. Any test vectors, that result in a large discrepancy between the actual and the target outputs of the neural networks, are associated with one or more new classes. These test vectors, that correspond to one or more novel classes, are set aside in a bin. At the end of the test trial, the data in the bin is clustered and any cluster that is found to be statistically different from any known class distributions denotes a new class.

This algorithm was implemented using a three-layer neural network, that contains 42 units in the first layer, 175 units in the hidden layer and 4 units in the last layer. The image data consists of 3777 samples extracted from regions (such as trees, grass, sky, and river reflecting the sky or trees). Trials were conducted such that the training data consisted of all classes but one, and the testing data consisted of instances from the excluded class (not used in training), along with a noisy version of the training data. The results for detecting the class "Sky" are presented in Table 4. In this trial, the "Sky" data are completely excluded from training and are only used for testing. Table 4 shows that the test data is classified with an accuracy of 79.6%. From the 136 test examples from the "Sky" data, only 129 examples were correctly assigned to the bin for the clustering analysis. The composition of the clusters (also shown in Table 4) were then analyzed, and "Sky" was found to be statistically different enough to be assigned a new class.

Table 4: The results of using neural network in conjunction to clustering to detect novel objects in video sequences. The left table shows the neural network's confusion matrix and the bin composition at the end of the test run. The right table shows the cluster composition at the end of clustering. Reproduced from [Singh *et al.*, 2000].

	G	T	S	Rs	Rt
G	**1126**	213	0	4	116
T	122	**596**	0	2	43
S	0	1	**0**	6	0
Rs	1	0	0	**223**	0
Rt	24	25	0	0	**224**

Classification = 79.6%

BIN

Grass	Trees	Sky	Rs	Rt
465	271	129	1	48

CLUSTER 1

Grass	Trees	Sky	Rs	Rt
409	239	1	1	40

CLUSTER 2

Grass	Trees	Sky	Rs	Rt
51	28	0	0	4

CLUSTER 3

Grass	Trees	Sky	Rs	Rt
5	4	128	0	3

For a more extensive review of novelty detection literature based on neural networks, see [Markou and Singh, 2003].

5 Generative approaches

In contrast to discriminative approaches, generative methods estimate $p(\mathbf{X}|C)$ and $P(C)$ for each class C, then apply Bayes' rule to compute the class-conditional distribution $P(C|\mathbf{X})$:

$$P(C|\mathbf{X}) = \frac{p(\mathbf{X}|C)P(C)}{p(\mathbf{X})} \tag{8}$$

Under this paradigm, the classification task is effectively reduced to modelling the distributions $p(\mathbf{X}|C)$ and $p(C)$. In general, this approach requires the estimation of a larger number of parameters. Since these parameters are estimated via maximum likelihood, generative models are usually less optimized for classification compared to the discriminative models, which optimize the

classification error directly. Nonetheless, generative models are favored among the community of model-based diagnosis [de Kleer *et al.*, 1992; Williams and Nayak, 1996] because generative models offer insights about the structure of the system and are useful in providing causal explanations for observed phenomena.

Given a set of observed data, a generative model relates the observed data to hidden variables that might have caused the observed data. The observed data are represented by the feature vectors $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$ and the hidden variables constitute the unknown classes C. (In most cases, a generative model may contain additional hidden variables that are beyond the scope of the classes, but are useful in improving the prediction between the input vectors $\mathbf{X}$ and their output classes C.) Under this framework, an input feature vector $\mathbf{x}$ is interpreted as a noisy observation of some unknown process in the system. This unknown process is assumed to switch between different classes or *modes* of behavior. Depending on the class under which the process is currently operating, the system will generate observations that are specific to that class. Thus, the class of system behavior can be inferred through classifying the observations.

The goal of a generative classifier is to output the class c^* that would have generated, with the highest probability, the observation represented by the input feature vector $\mathbf{x}$. Thus, before classification or anomaly detection can occur, a model of the system must be developed, by incorporating prior information and using unsupervised learning on the unlabelled training data $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$. Once the model $\mathcal{M}$ is developed, inference is performed on $\mathcal{M}$ to compute $P(C = k|\mathbf{X} = \mathbf{x}; \mathcal{M})$, the probability of each class k conditional the input feature vector $\mathbf{x}$, with respect to the model $\mathcal{M}$. Classification boils down to simply choosing the class with the highest probability, i.e., $c^* = \arg\max_k P(C = k|\mathbf{X} = \mathbf{x}; \mathcal{M})$. In essence, a generative model must capture the system dynamics under each class or mode of behavior. On its own, a generative model will not predict the presence of new classes. Instead, one must apply probability thresholding, hypothesis testing or other more sophisticated detection schemes, to the generative models to detect possible emergence of new classes.

In this section, we start off by presenting two complementary methods of density estimation that are commonly used to create generative models. The first is a non-parametric method known as Parzen windows, while the other is a parametric method known as modelling by mixture of Gaussians. The second part of this section discusses more structured representations used for generative modelling. Before we discuss generative models of *temporal* processes, we will explain how state estimation is related to anomaly detection, as state estimation plays a crucial part in the anomaly detection of temporal processes. Lastly, we will examine popular models of temporal processes, such as hidden Markov models and dynamic Bayesian networks, and provide references to recent work that has applied these models for classification or anomaly detection.

5.1 Parzen windows

The Parzen windows algorithm [Duda *et al.*, 2001] is an unsupervised method of non-parametric density estimation and can be easily adapted for classification. This algorithm makes use of a *kernel function* to interpolate the probability of the input space that is not supported by the data. This kernel function can be quite general, as long as it satisfies the properties for a valid probability density function.

Given a kernel function, the Parzen windows method fits this kernel function around every element of the data set and uses a linear combination of these kernels to approximate the probability distribution of the data. For simplicity and convenience, the Gaussian distribution is often used as the kernel function, and the probability of a test vector $\mathbf{x}$ is approximated as a mixture of radially symmetric Gaussians with the same variance σ^2. For a data set consisting of N d-dimensional vectors, the Parzen windows method estimates the true distribution $p(\mathbf{x})$ by:

$$\hat{p}_N(\mathbf{x}) \triangleq \frac{1}{N}\sum_{n=1}^{N}\varphi\left(\frac{\mathbf{x}-\mathbf{x}^{(n)}}{\sigma}\right) \tag{9}$$

$$= \frac{1}{N}\sum_{n=1}^{N}\frac{1}{(2\pi)^{d/2}\sigma^d}\exp\left(-\frac{||\mathbf{x}-\mathbf{x}^{(n)}||^2}{2\sigma^2}\right) \tag{10}$$

With Gaussian kernels, points that are far away from the test point are virtually irrelevant, as the contribution of these points decreases exponentially with the square of the distance. The width of the kernel is determined by the variance σ^2. If σ is too small, then the estimated distribution of the data would be overfitting the data, in the form of peaks around each data point. If σ is too large, then the estimated distribution would suffer from low resolution, as distributions of different classes may overlap and blend separate classes into one single class. With a limited number of data, one must seek a compromise between these two extremes and empirically fix σ to minimize the classification error. (This trade-off is illustrated in Figure 14.) But in the case where an unlimited number of examples is available, one can let $\sigma \to 0$ and achieve an asymptotically close estimate to the true distribution of the data. Specifically, for all $\mathbf{x}$, when the number of examples N goes to infinity, $p_N(\mathbf{x})$ converges to $p(\mathbf{x})$ in the mean square sense, where

$$\lim_{N\to\infty} \mathbb{E}\left[p_N(\mathbf{x})\right] = p(\mathbf{x}) \tag{11}$$

$$\lim_{N\to\infty} \mathbf{var}\left[p_N(\mathbf{x})\right] = 0 \tag{12}$$

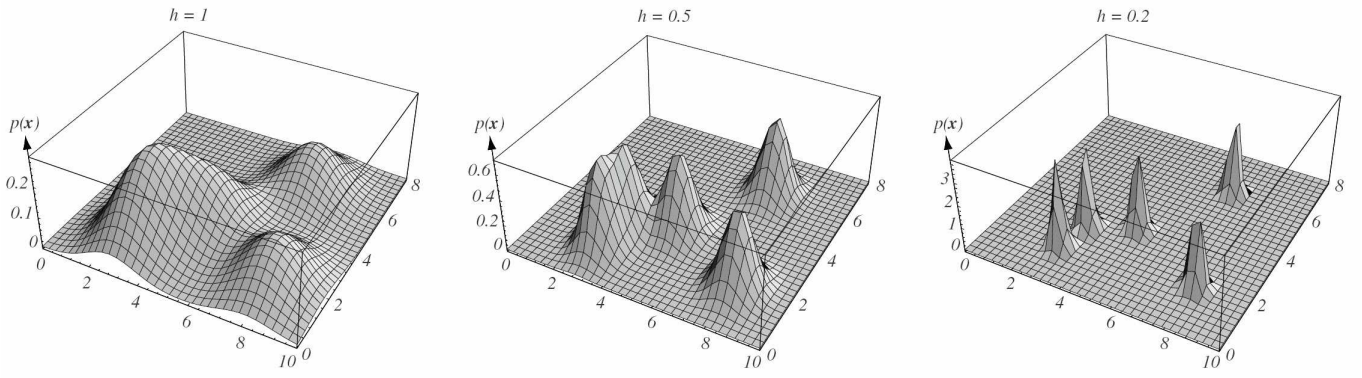

Figure 14: Three Parzen-windows estimate of the data, based on the same five testing examples. The vertical axes are scaled to show the structure of each distribution. In this figure, h has the same function as σ from our discussion. Reproduced from Richard O. Duda, Peter E. Hart and David G. Stork, *Pattern Recognition*. ©2001 by John Wiley & Sons, Inc.

For classification, the generative paradigm is followed: $p(\mathbf{X}|C)$ is first estimated from the data using Parzen windows density estimation and $P(C)$ is either estimated by a simple frequency distribution (if C is finite) or a subjective prior distribution that reflects one's belief about the distribution of classes. (In our equations, we assume the frequency approach for estimating $P(C)$.) Then $P(C|\mathbf{X})$ is computed from the Bayes rule, as shown in Equation 8. At this point, the data should have been partitioned into different classes, based on the shape of the probability distribution (Equation 9) as estimated by the Parzen windows method. As a result, the following estimates can be obtained for each class k:

$$p(\mathbf{X} = \mathbf{x}|C = k) \approx \frac{1}{N_k} \sum_{n\in\mathcal{G}_k} \varphi\left(\frac{\mathbf{x} - \mathbf{x}^{(n)}}{\sigma}\right) \tag{13}$$

$$P(C = k) \approx \frac{N_k}{N} \tag{14}$$

where N is the number of data vectors, of which N_k vectors belong to class k. Note that the sum for $p(\mathbf{X} = \mathbf{x}|C = k)$ is taken over the indices in $\mathcal{G}_k$, which correspond to those data vectors that belong to class k. In other words, $|\mathcal{G}_k| = N_k$. Combining these two expressions results in locally weighted averaging of the data:

$$P(C = k|\mathbf{X} = \mathbf{x}) \approx \frac{\sum_{n=1}^{N} \varphi\left(\frac{\mathbf{x}-\mathbf{x}^{(n)}}{\sigma}\right) \mathbf{I}\left(\mathbf{x}^{(n)} \mapsto k\right)}{\sum_{n=1}^{N} \varphi\left(\frac{\mathbf{x}-\mathbf{x}^{(n)}}{\sigma}\right)} \tag{15}$$

where $\mathbf{I}\left(\mathbf{x}^{(n)} \mapsto k\right)$ is an indicator function that evaluates to 1 if $\mathbf{x}^{(n)}$ belongs to class k and evaluates to 0 otherwise. Lastly, $\mathbf{x}$ is assigned to the class k with the highest probability $P(C = k|\mathbf{X} = \mathbf{x})$.

In fact, one can interpret the Parzen classifier as a generalization of the k-nearest-neighbor method. In the k-nearest-neighbor classifier, the class of a test point $\mathbf{x}$ is determined by the majority vote of the classes from $\mathbf{x}$'s k nearest neighbors. Instead of examining just the k-nearest neighboring vectors, the Parzen classifier considers every vector in the data set and weights their votes by a kernel function centered on the test point $\mathbf{x}$. Although the method considers every data vector, not every vector actually contributes to the majority vote, since vectors that are located outside of the kernel function will have 0 weight.

In the limit of infinite amount of data, the Parzen window estimate of the data distribution approaches the true distribution. In practice, many data vectors may be required for a reasonable estimate of the data distribution. This demand for data grows exponentially with the dimensionality of the data, limiting this method's applicability due to its severe computation and memory requirements.

The method of Parzen windows has been applied with success to novelty detection for intrusion detection in [Yeung and Chow, 2002]. In this work, novelty detection is formulated as a hypothesis test, where the log-likelihood of the test vector, $L(\mathbf{x})$, and the log-likelihood of an arbitrary vector $\mathbf{y}$ sampled from the normal class, $L(\mathbf{y})$, are compared. If $P(L(\mathbf{y}) \leq L(\mathbf{x})) > \psi$ for some false alarm rate $\psi \in (0, 1)$, then $\mathbf{x}$ is labelled as belonging to the normal class. Otherwise, $\mathbf{x}$ is labelled as anomalous. The study used the 1999 KDD Cup data set [Hettich and Bay, 1999], which contains a standard set of data to be audited, including a wide variety of intrusions simulated in a military network environment. The study compared the proposed Parzen-based intrusion detection system against the KDD Cup winner, using the true acceptance rate (TAR) and the true detection rate (TDR) as the performance metrics. The TAR measures the percentage of normal instances in the test set that were correctly classified as normal, while the TDR measures the percentage of intrusions in the test set that were correctly classified as intrusions. To estimate the distribution for the normal class, 3000 randomly generated examples were used as training data. The empirical results are shown in Table 5, where the Parzen-based detector outperformed the KDD Cup winner in the detection of intrusions, with similar or much higher values for the TDR. On the more difficult types of intrusion (types 3 and 4 shown in Table 5), the KDD Cup winner performed poorly while the Parzen-based detector was able to dominate by a clear performance margin.

Table 5: Comparison of the Parzen-based intrusion detector and the KDD Cup winner. The higher performance score is shown in boldface. Reproduced from [Yeung and Chow, 2002].

Method	TAR	TDR			
	Normal	Intrusion Type 1	Intrusion Type 2	Intrusion Type 3	Intrusion Type 4
Parzen-based	97.38%	**99.17%**	96.71%	**93.57%**	**31.17%**
KDD winner	**99.45%**	87.73%	**97.69%**	26.32%	10.27%

5.2 Mixture of Gaussians

A Gaussian mixture density is a d-dimensional probability distribution, defined by the weighted sum of M components:

$$p(\mathbf{X}|\theta) = \sum_{m=1}^{M} p(\mathbf{X}|m)P(m) \tag{16}$$

where each component $p(\mathbf{X}|m)$ is a d-dimensional multivariant Gaussian distribution with mean μ_m and covariance Σ_m:

$$p(\mathbf{X}|m) = \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{X} - \mu_m)'\Sigma_m^{-1}(\mathbf{X} - \mu)\right) \tag{17}$$

and $P(m)$ is a mixture coefficient, where $P(m) \geq 0$ for $m = 1, ..., M$ and $\sum_{m=1}^{M} P(m) = 1$. Thus, the Gaussian mixture density is parametrized by the parameters from all its components:

$$\theta = \{\mu_1, ..., \mu_M, \Sigma_1, ..., \Sigma_M, P(1), ..., P(M)\} \tag{18}$$

For classification, the data from each class k are represented by a Gaussian mixture density, where $p(\mathbf{X}|C = k) \triangleq p(\mathbf{X}|\theta_k)$ (as given by Equation 16). The parameters θ_k are typically optimized in a

maximum likelihood sense by using the expectation-maximization algorithm [Dempster *et al.*, 1977; Bilmes, 1997], or in a Bayesian sense by using Markov chain Monte Carlo methods [Gilks *et al.*, 1995] to sample the parameters from the posterior distribution. The class probability $P(C)$ is often assumed to be a discrete distribution or a multivariate Gaussian for computational efficiency, in which the parameters to $P(C)$ are typically learned using maximum likelihood or Bayesian techniques. A test vector $\mathbf{x}$ is assigned to the most probable class $c \in \{1, ..., K\}$ such that:

$$c = \underset{1 \leq k \leq K}{\arg\max} P(C = k|\mathbf{X} = \mathbf{x}) \propto \underset{1 \leq k \leq K}{\arg\max} p(\mathbf{X} = \mathbf{x}|\theta_k)P(C = k) \tag{19}$$

Gaussian mixture models have been applied with success to speaker identification [Reynolds and Rose, 1995] and active learning for anomaly detection [Pelleg and Moore, 2005]. In [Reynolds and Rose, 1995], the use of Gaussian mixture densities for modelling speaker identity was motivated by two main reasons, that (1) Gaussian components can effectively model a speaker's acoustic classes and (2) Gaussian mixtures can be used to model any arbitrary distributions, thus lending to their versatility. In this study, the classification task was to correctly classify a test segment of speech and identify the speaker from whom the speech segment was generated. The experiments were conducted on a subset of the KING speech database [Godfrey *et al.*, 1994], which contains conversation speech samples from 51 male speakers. For each speaker, there are 10 independent conversations of approximately 45 seconds. The performance metric is taken to be the percentage of correct identification. The results for the study are presented in Figures 15 and 16. In Figure 15, the performance of speaker identification is plotted against the number of mixture components in the Gaussian mixture model, for test segments and training segments of varying lengths. The empirical results show that beyond 16 components, there is only marginal improvement in the performance. Moreover, as the amount of training data is reduced, it becomes more crucial to choose the optimal number of components for the Gaussian mixture. In Figure 16, the speaker identification performance is plotted against the length of the test speech segment, for different number of speakers. The left plot presents the results for clean speech and the right plot presents the results for the telephone speech. For clean speech, near perfect identification is achieved when the test speech is 15 seconds. However, for telephone speech, due to the low signal-to-noise ratio of the audio data, there is a significant degradation in performance.

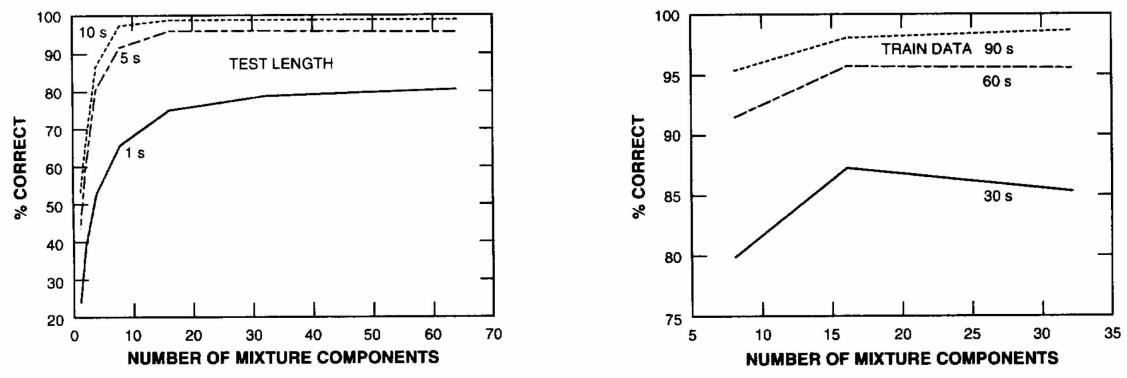

Figure 15: Speaker identification performance as a function of the number of components in the Gaussian mixture model. The left plot shows the performance for test segments of 1,5, and 10 seconds in length, in which the model is trained on 1 minute of training segment. The right plot shows the performance for training segments of 30,60 and 90 seconds of speech, where the test segment is 5 seconds. Reproduced from [Reynolds and Rose, 1995].

In addition to speaker identification, Gaussian mixture models have also been applied in an active learning framework to find rare and useful anomalies. The active learning approach proposed by [Pelleg and Moore, 2005] assumes that the distribution of the data is extremely skewed towards the normal class and that a mixture model can be used to fit the data. The active learning process proceeds in rounds, where the computer attempts to learn the model of the data from a small number of labelled training instances along with a large number of unlabelled training instances. Then it identifies a small number of difficult instances and elicits the human user to provide labels for these difficult instances. The human user labels these instances and add them to the collection of labelled

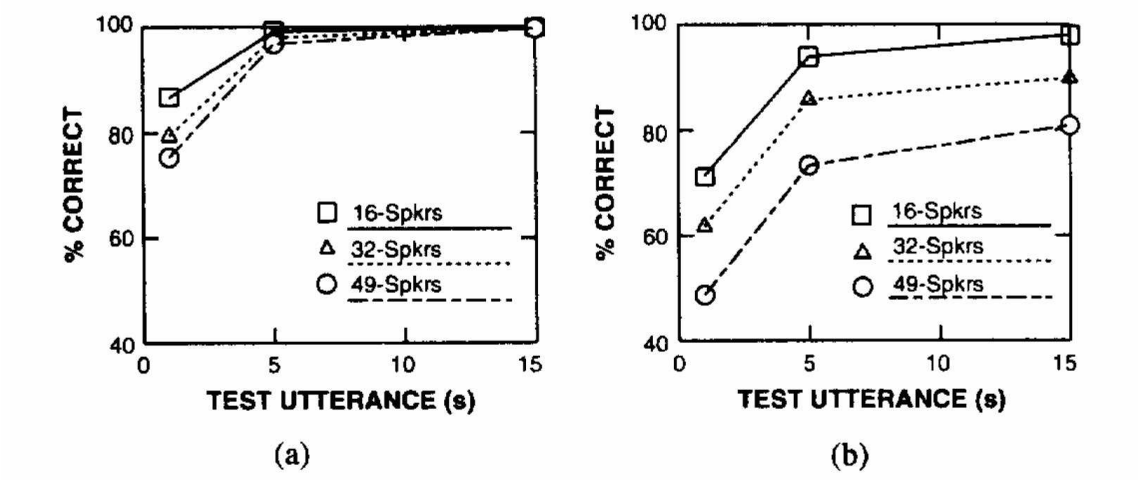

Figure 16: Speaker identification performance as a function of the test segment length, for population sizes of 16, 32 and 49 speakers. The model contains 50 components and is trained using 80-100 seconds of speech per speaker. The left shows the performance for clean speech and the right shows the performance for noisy telephone speech. Reproduced from [Reynolds and Rose, 1995].

data, and the cycle repeats. This framework is flexible in that the labels are unconstrained and the user has the freedom to add, delete and change the classes at will. As a result, this system allows for adaptive learning of a dynamic data set.

This work presents several selection methods by which unlabelled instances are chosen for the feedback process. The following criteria for *hint selection* were examined:

- LOWLIK: Choosing instances with low likelihood, i.e., data on which the model performs worst. In particular, instances are ranked in increasing order of model likelihood and the ones with low likelihoods are chosen.
- AMBIG: Choosing ambiguous instances, i.e. data that the computer has the least certainty about. In particular, instances are ranked in decreasing order of entropy and the ones with high entropy are chosen.
- MIX-AMBIG-LOWLIK: Hybrid approach of the two selection schemes described above.
- INTERLEAVE: Choosing points based on the ranking from the perspective of just one component, instead of a mixture of components.

Experiments were ran on synthetic data and real data. The last approach of INTERLEAVE seems to perform best in both synthetic and real cases. Working from the definition of a mixture model, this active learning approach allows each component to nominate its favorite queries. Empirical results show that this method works well in the presence of noisy data and extremely rare anomalies. Figures 17 and 18 shows the learning curves for various real data sets, as characterized in Table 6. The learning curves show the percentage of classes discovered as a function of the number of hints requested by the computer system.

Table 6: Properties of the real-life data sets used in the [Pelleg and Moore, 2005] study. Reproduced from [Pelleg and Moore, 2005].

Data set	# dim	# records	# classes	smallest class	largest class
ABALONE	7	4177	20	0.34%	16%
KDD	33	50000	19	0.002%	21.6%
EDSGC	26	1439526	7	0.002%	76%
SDSS	22	517371	3	0.05%	50.6%

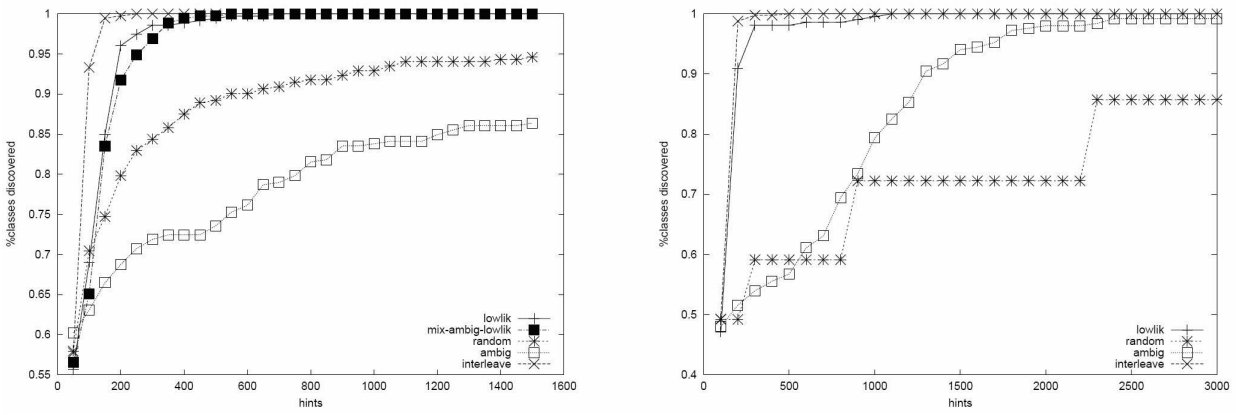

Figure 17: The learning curves for the ABALONE [Newman *et al.*, 1998] (left) and the KDD [Newman *et al.*, 1998] data sets. Reproduced from [Pelleg and Moore, 2005].

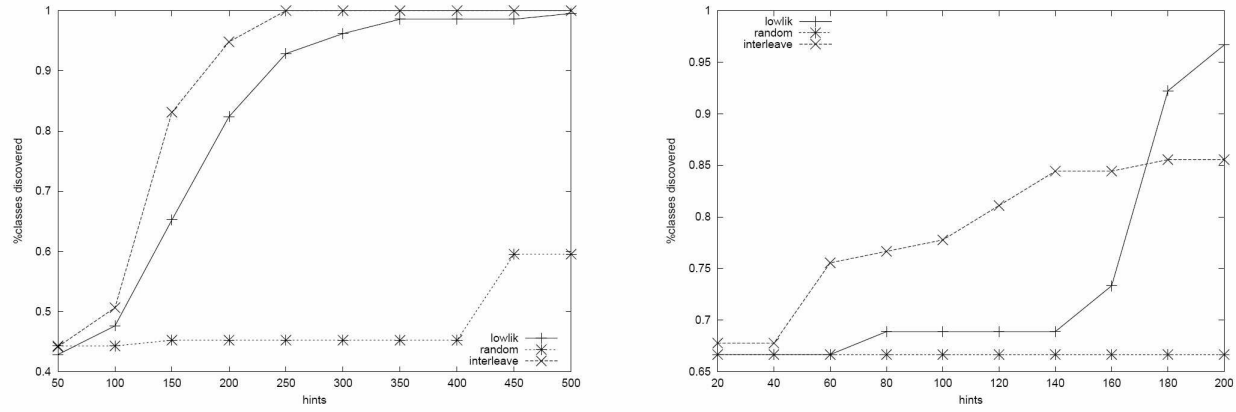

Figure 18: The learning curves for the EDSGC [Nichol *et al.*, 2000] (left) and the SDSS [Finkbeiner *et al.*, 2004] (right) data sets. Reproduced from [Pelleg and Moore, 2005].

The experiments show that this tremendously reduces the number of instances that a human user must label before presentation to an automatic learner. In fact, a user only needs to label one or two hundred examples before a rare anomaly from a huge data set is being presented to the user.

To allow for greater flexibility in the Gaussian mixture model, [Lee and Lewicki, 2000] proposed the *generalized Gaussian mixture model*, whose mixture component can deviate from normality to represent platykurtic and leptokurtic distributions. As a comparison, the generalized Gaussian mixture model was compared against the standard Gaussian mixture model and the k-means clustering algorithm, in the unsupervised classification of the data set shown in Figure 19. The number of classes is assumed known and the classification task is to learn the model parameters and to classify the data. The classification error was 4.0%$\pm$0.5% for the generalized Gaussian mixture model, 5.5%$\pm$0.3% for the Gaussian mixture model, and 18.3% for the k-means clustering algorithm. The study showed that the generalized Gaussian mixture model is equal or superior to the Gaussian mixture model, especially in data that are non-Gaussian and abound with outliers.

5.3 State estimation vs. anomaly detection

Before we discuss generative models for anomaly detection of temporal processes, it is useful to relate state estimation to anomaly detection. In the generative framework, probabilistic models encode how a system evolves over time. At each time τ, the state is denoted by $\mathbf{S}_\tau$ and the history of observations up to time τ is given by $\mathbf{y}_{1:\tau} = \{\mathbf{y}_1, ..., \mathbf{y}_\tau\}$. The posterior distribution $p(\mathbf{S}_\tau|\mathbf{y}_{1:\tau})$ represents our belief about the system at time τ given the most up-to-date information.

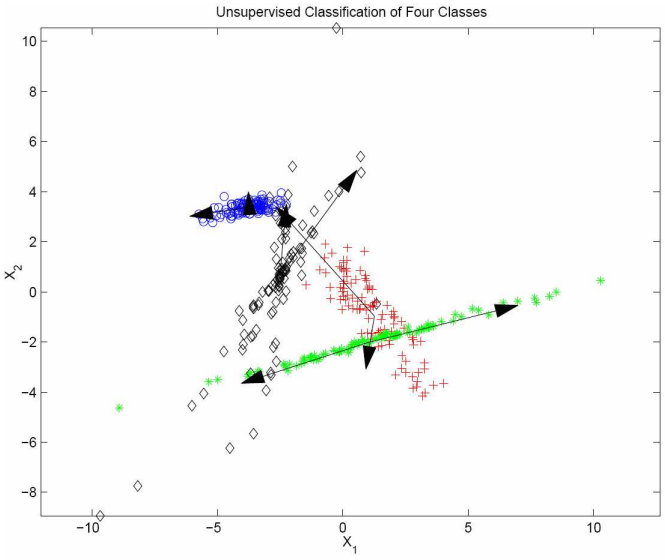

Figure 19: Example data used in the unsupervised classification task in [Lee and Lewicki, 2000]. The data in each class was generated by a random distribution. Reproduced from [Lee and Lewicki, 2000].

State estimation is a two-step process that is achieved through Bayesian updating. Given $p(\mathbf{S}_{t-1}|\mathbf{y}_{1:t-1})$, state prediction is the act of computing the predictive distribution $p(\mathbf{S}_t|\mathbf{y}_{1:t-1})$, which represents the best estimate of $\mathbf{S}_t$ given observations up to time $t-1$. When $\mathbf{y}_t$, the observation at time t, is made available, we apply this information to update the predictive distribution and obtain $p(\mathbf{S}_t|\mathbf{y}_{1:t})$, the posterior distribution at time t. The two steps of prediction and correction comprise the overall process of state estimation, as shown in Figure 20.

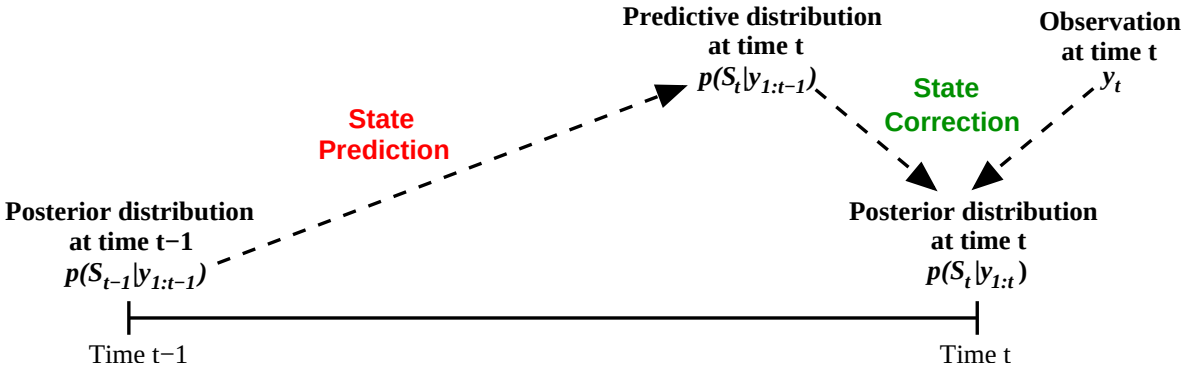

Figure 20: The iterative process of state estimation. State prediction is the act of computing $p(\mathbf{S}_t|\mathbf{y}_{1:t-1})$, the predictive distribution for the state at a future time t while using only the history of observations from time 1 to $t-1$. State correction is the act of updating the predictive distribution $p(\mathbf{S}_t|\mathbf{y}_{1:t-1})$ to yield the posterior distribution $p(\mathbf{S}_t|\mathbf{y}_{1:t})$, which incorporates the most current observation as part of the state estimate. Classification and anomaly detection involve mathematical operations on the posterior distribution $p(\mathbf{S}_t|\mathbf{y}_{1:t})$.

Both classification and anomaly detection typically involve operations on the posterior distribution. Given $p(\mathbf{S}_t|\mathbf{y}_{1:t})$, classification boils down to identifying the class (which is usually represented as part of the state) with the highest probability, and anomaly detection reduces to thresholding the posterior distribution then attributing the low-probability states to a completely new class. In addition to strictly using the posterior distribution, one can also use significant changes between the predictive distribution $p(\mathbf{S}_t|\mathbf{y}_{1:t-1})$ and the posterior distribution $p(\mathbf{S}_t|\mathbf{y}_{1:t})$ as an indication for abrupt state changes. When $p(\mathbf{S}_t|\mathbf{y}_{1:t})$ differs dramatically from $p(\mathbf{S}_t|\mathbf{y}_{1:t-1})$, this usually means that a state

transition occurred at time t and the effect of this transition was captured by the observation y_t. By using the discrepancy between $p(\mathbf{S}_t|\mathbf{y}_{1:t-1})$ and $p(\mathbf{S}_t|\mathbf{y}_{1:t})$ as an additional detection criterion, one may be better able to infer *when* exactly the system changes from a normal state to an anomalous state, and vice versa.

Thus, it is clear that generative classification/detection relies heavily on state estimation. In turn, the efficiency of state estimation depends on the class of models being used for generative modelling and the type of inference algorithms being used for reasoning about these models.

5.4 Hidden Markov models

A hidden Markov model (HMM) [Rabiner, 1989] is a graphical model that represents a Markov process whose state $\mathbf{S}_t$ is hidden and can only be inferred through (noisy) observations $\mathbf{Y}_t$. An HMM assumes that the system evolves according to a first-order Markov process in which the current state depends only on its previous state:

$$p(\mathbf{S}_t|\mathbf{S}_{0:t-1}) = p(\mathbf{S}_t|\mathbf{S}_{t-1}), \quad t = 1, 2, ... \tag{20}$$

and additionally, observations depend only on the current states:

$$p(\mathbf{Y}_t|\mathbf{S}_{0:t}) = p(\mathbf{Y}_t|\mathbf{S}_t), \quad t = 1, 2, ... \tag{21}$$

For simplicity, most HMMs assume that the probabilities $p(\mathbf{S}_t|\mathbf{S}_{t-1})$ and $p(\mathbf{Y}_t|\mathbf{S}_t)$ are stationary, in that these distributions do not change with time:

$$\begin{array}{lcll} p(\mathbf{S}_t|\mathbf{S}_{t-1}) & = & p(\mathbf{S}_{s+t}|\mathbf{S}_{s+t-1}), & s \geq 0 \\ p(\mathbf{Y}_t|\mathbf{S}_t) & = & p(\mathbf{Y}_{s+t}|\mathbf{S}_{s+t}), & s \geq 0 \end{array} \tag{22}$$

With these assumptions, an HMM characterizes the temporal process by its transition model $p(\mathbf{S}_t|\mathbf{S}_{t-1})$ and its observation model $p(\mathbf{Y}_t|\mathbf{S}_t)$. Thus, the state evolves probabilistically according to $p(\mathbf{S}_t|\mathbf{S}_{t-1})$ and is observed through noisy measurements according to $p(\mathbf{Y}_t|\mathbf{S}_t)$, as shown in Figure 21.

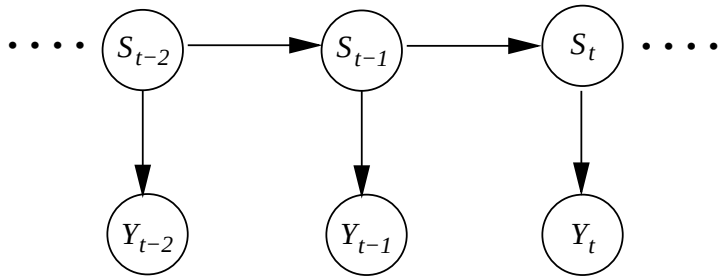

Figure 21: A hidden Markov model. The state $\mathbf{S}_t$ evolves probabilistically according to $p(\mathbf{S}_t|\mathbf{S}_{t-1})$, where the causal dependence between $\mathbf{S}_{t-1}$ and $\mathbf{S}_t$ is shown by the arrow from $\mathbf{S}_{t-1}$ to $\mathbf{S}_t$. In turn, the state $\mathbf{S}_t$ is observed through noisy measurements according to $p(\mathbf{Y}_t|\mathbf{S}_t)$, where the causal dependence between $\mathbf{S}_t$ and $\mathbf{Y}_t$ is represented by the arrow from $\mathbf{S}_t$ to $\mathbf{Y}_t$.

HMMs are well-studied and established algorithms are available for inference and parameter learning. For classification, the Viterbi algorithm [Viterbi, 1967] can be applied to any finite-state HMM to find the most probable sequence of hidden states that could have generated a given observation sequence. Since the input features can be associated with the observations and the unknown classes with the hidden states, one can use the Viterbi algorithm to classify any input vector to its most probable class. Application of HMMs for classification and anomaly detection has been quite popular in the intrusion detection community [Lane, 1999; Yeung and Ding, 2003; Ourston *et al.*, 2003; Wright *et al.*, 2004].

In [Yeung and Ding, 2003], the effectiveness of dynamic and static models for intrusion detection were compared in an empirical study. In particular, the study examined the dynamic and static models' performance on detecting intrusions from program data (based on system calls) and from user data (based on shell commands). The dynamic model is represented by HMMs, trained on data corresponding to normal behavior. In this dynamic framework, observations with low likelihood values,

in respect to the model, are associated as intrusions. For the static model, frequency counting of event occurrences is used to estimate the frequency distribution that characterizes normal behavior. In this static framework, the criterion for an intrusion is if the cross entropy between two frequency distributions, in respect to the model, exceeds a given threshold. The clear difference between the two models is that the static one does not take into account the order in which events occur, while the dynamic one does incorporate this information as part of the HMM. Like [Yeung and Chow, 2002], novelty detection is formulated as a hypothesis test and the performance metric is measured in terms of the true acceptance rate (TAR) and the true detection rate (TDR). (See Subsection 5.1 for details.)

The two modelling approaches were tested on two different types of intrusion data: the system call data sets from the University of New Mexico [Forrest and Hofmeyr, 1998] and the shell command data sets from the University of California at Irvine [Newman *et al.*, 1998]. On the first set of data, the dynamic models showed superior performance, probably due to the temporal dependencies between system calls. But on the second set of data, the dynamic models were outperformed by the static models. (Extensive results are presented in [Yeung and Ding, 2003], please refer for details.) Thus, it was inferred that temporal dependencies in shell command sequences may not be very useful and could potentially be a noisy feature that deters detection. As a result, this study suggests that HMMs should be used only for applications with strong temporal dependencies.

The observation that HMMs are well-suited for modelling of sequential processes is supported by [Ourston *et al.*, 2003], in which HMMs were applied to detecting multi-stage network attacks. These attacks typically span an extended period of time and are composed of multiple steps in which actions may interchange within each step. The requirement for interchangeable actions allows the system to model random or deceptive behavior on the part of the perpetrator, who might attempt to mask his or her actions by random changes in the action sequence. Due to the sequential nature of this domain, it makes intuitive sense to use a temporal model like an HMM, because the order of actions that constitute an attack provides invaluable information about the nature of the attack.

The experiments were conducted on a set of in-house network sensor data, in which a semi-automatic procedure was used to categorize the data into classes defined by subject experts. Each training example consists of an alert sequence, ordered temporally, that occurred over a 24-hour period. A comparison was made between the HMM approach, a decision tree algorithm and a neural network, as shown in Figure 22. The performance metric was the fraction of test examples that were correctly classified.

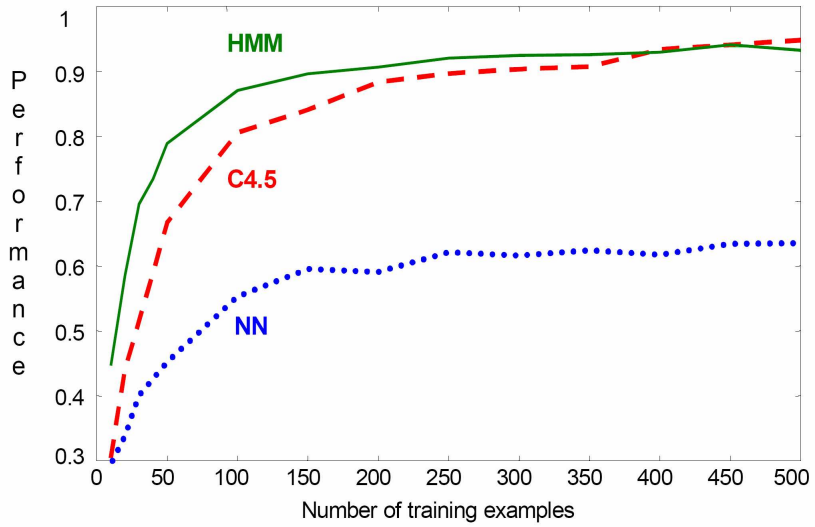

Figure 22: The detection performance, defined as the fraction of test examples that were correctly classified, as a function of the number of training examples. C4.5 refers to the decision tree approach and NN refers to the neural network approach. Reproduced from [Ourston *et al.*, 2003].

In Table 7, the confusion matrix and the precision, recall and F-measure statistics are shown. The confusion matrix displays the probability that a test example from a particular class is assigned to the

Table 7: The results of the experiment expressed as a confusion matrix and statistics of precision, recall and F-measure. Reproduced from [Ourston *et al.*, 2003].

Training examples		Test examples		Performance		Sampling iterations								
300		100		0.9255		100								
		Examples assigned to category												
		21.1	0.0	15.0	29.7	4.6	0.3	0.0	9.8	2.6	12.1	4.4	0.3	0.0
Cat	Exs in cat	Confusion matrix												
1	20.2	**0.96**	0.00	0.02	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.0	0.80	**0.20**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	15.7	0.02	0.00	**0.92**	0.01	0.03	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.00
4	32.4	0.01	0.00	0.00	**0.90**	0.06	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.00
5	2.2	0.05	0.00	0.03	0.05	**0.83**	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00
6	0.2	0.00	0.00	0.00	0.00	0.00	**0.35**	0.00	0.65	0.00	0.00	0.00	0.00	0.00
7	0.2	0.85	0.00	0.00	0.00	0.00	0.00	**0.15**	0.00	0.00	0.00	0.00	0.00	0.00
8	9.4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00
9	2.5	0.03	0.00	0.01	0.03	0.00	0.02	0.00	0.02	**0.90**	0.00	0.00	0.00	0.00
10	12.7	0.04	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	**0.93**	0.00	0.01	0.00
11	4.4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00
12	0.1	0.00	0.00	0.00	0.50	0.00	0.00	0.00	0.00	0.00	0.33	0.00	**0.17**	0.00
13	0.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**0.00**
Precision		0.92	1.00	0.96	0.98	0.47	0.25	1.00	0.96	0.87	0.97	1.00	0.06	?
Recall		0.95	0.25	0.92	0.90	0.81	0.35	0.13	1.00	0.93	0.93	1.00	0.17	?
F-measure		0.93	1.00	0.94	0.93	0.60	0.89	0.89	0.98	0.90	0.95	1.00	0.67	?

another output class. The diagonal of the confusion matrix shows the probability that a test example is correctly assigned to its class. Precision, recall and F-measure are functions of the true positives (tp), false positives (fp) and false negatives (fn), as follows:

$$\text{Precision} = P = \frac{tp}{tp+fp}, \quad \text{Recall} = R = \frac{fp}{tp+fp}, \quad \text{F-measure} = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} \tag{23}$$

where tp correspond to the number of correctly identified intrusions, fp correspond to the number of erroneously identified intrusions, and fn correspond to the number of instrusions missed by the detection system. The parameter α was set to 0.5 to allow for equal weighting between precision and recall. In general, the precision, recall and F-measure values show relatively good performance, except for classes where with training and testing data are insufficient. To quantify the value of extra training data, experiments were run with different number of training examples, and the performance, in terms of ROC curves and the area under these curves, are presented in Figures 23 and 24. As expected, it can be seen that detection generally improves with more training data.

5.5 Dynamic Bayesian networks

Dynamic Bayesian networks (DBNs) [Dean and Kanazawa, 1989] are compact representations of Markov processes. Like an HMM, a DBN is a graphical model, where state variables are represented as nodes and causal influences between variables are represented as arrows between nodes. But in contrast to an HMM where parameters are defined over the entire state $\mathbf{S}_t$, each node S_t in a DBN is associated with a conditional probability distribution $p(S_t|\mathbf{Pa}(S_t))$ that encapsulates the conditional probability of that variable given its *parents* $\mathbf{Pa}(S_t)$. (A variable's parents are the subset of the state variables that affects that variable.) Thus, DBNs generalize HMMs in that DBNs exploit conditional independencies between state variables to represent the transition model and the observation model in a factored manner, as a product of lower-dimensional probability distributions that correspond to the local dynamics:

$$p(\mathbf{S}_t|\mathbf{S}_{t-1}) = \prod_i p(S_{i,t}|\mathbf{Pa}(S_{i,t})) \tag{24}$$

$$p(\mathbf{Y}_t|\mathbf{S}_t) = \prod_j p(Y_{j,t}|\mathbf{Pa}(Y_{j,t})) \tag{25}$$

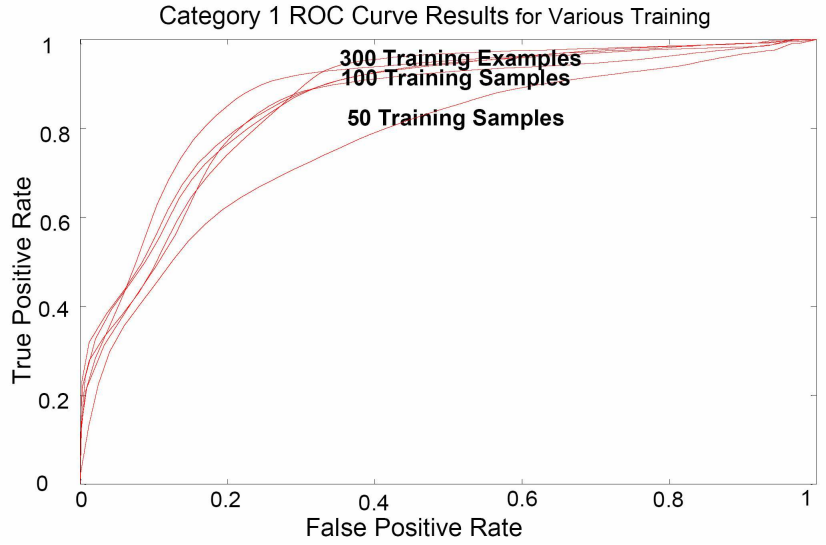

Figure 23: Detection performance, expressed as ROC curves, for training data of different sizes. The results presented are from the testing data for class 1. Reproduced from [Ourston *et al.*, 2003].

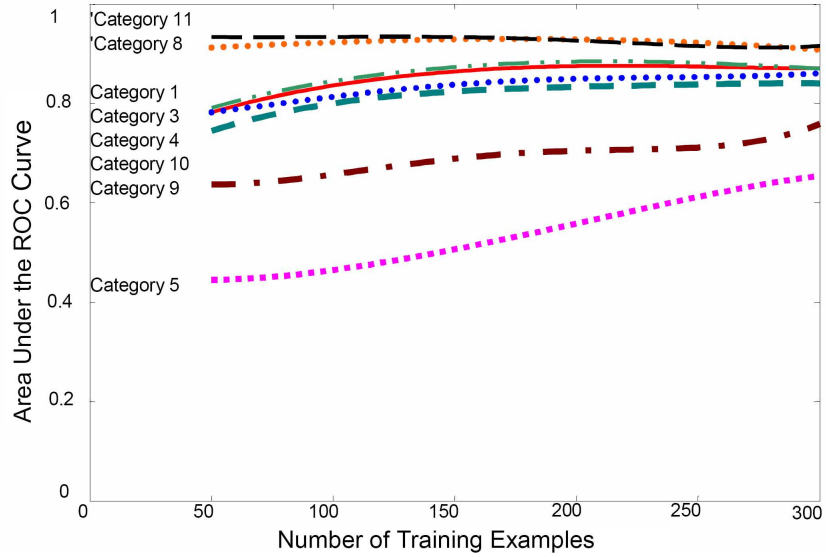

Figure 24: Detection performance, expressed as area under the ROC curves, for training data of different sizes. Classes with insufficient testing data are not shown. Reproduced from [Ourston *et al.*, 2003].

where i is the index over the state variables and j is the index over the observation variables. [For a detailed description of DBNs, please refer to the companion survey on Bayesian models for temporal processes.]

The use of DBNs is quite popular for modelling temporal processes in applications of fault detection and diagnosis. In these applications, the state $\mathbf{S}_t$ is often augmented to include fault variables $\mathbf{Z}_t$ that denote the absence or presence of different faults. Since the presence of a fault affects how the system behaves, this causal effect is represented graphically by the arrow from the fault variables $\mathbf{Z}_t$ to the system variables $\mathbf{V}_t$, shown graphically in Figure 25. Note that, in this framework, the observation variables $\mathbf{Y}$ take on the role of the input features $\mathbf{X}$, and the fault variables $\mathbf{Z}$ now represent the output classes. The variables $\mathbf{V}$ can be interpreted as auxiliary random variables that model other aspects of the system, which may aid in clarifying the relationship between $\mathbf{Y}$ and $\mathbf{Z}$.

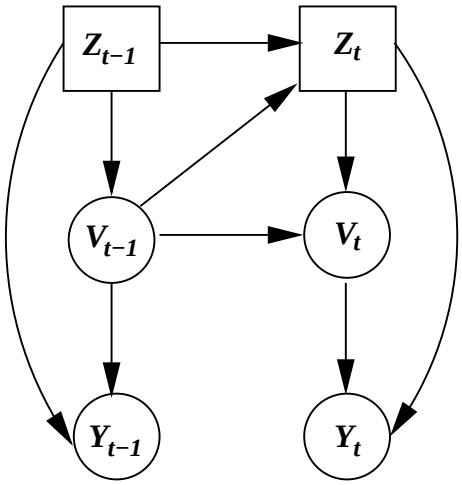

Figure 25: An example DBN of a hybrid-state system that includes fault variables. In this example, the fault variables $\mathbf{Z}_t$ are discrete-state (shown as square nodes), while the system variables $\mathbf{V}_t$ and the observation variables $\mathbf{Y}_t$ are continuous-state (shown as circular nodes). $\mathbf{V}_t$ is observed through the variable $\mathbf{Y}_t$, whose measurements may be affected in the presence of a fault, thus the arrow going from $\mathbf{Z}_t$ to $\mathbf{Y}_t$.

In this modelling paradigm, the state contains both the system and fault variables, i.e., $\mathbf{S}_t = \{\mathbf{V}_t, \mathbf{Z}_t\}$. From the history of values for $\mathbf{Y}_t$ (the observed variables), the probability distribution $P(\mathbf{S}_t|y_{1:t})$ is estimated using standard state estimation techniques. From $P(\mathbf{S}_t|y_{1:t})$, the most probable fault states $\mathbf{z}$ (in respect to $P(\mathbf{Z}_t|y_{1:t})$) are identified and are presented to an analyst for the proper diagnosis of the system.

Under this paradigm of system modelling, [Lerner *et al.*, 2000] have shown success with using hybrid-state DBNs for plant modelling and fault diagnosis. In this work, a DBN model of a multi-tank system was constructed from the specifications given in a temporal causal graph. The DBN model contained fault variables that represent:

- Measurement faults, which occur when a sensor fails, causing measurements to become extremely noisy.
- Burst faults, which occur when a pipe bursts, causing the pipe's resistance to change abruptly to some unknown value.
- Drift faults, which occur as a result of normal wear-and-tear on a pipe, in which the pipe's resistance may gradually drift to a non-calibrated value.

The entire system consists of five tanks connected in sequence by pipes. Fluid exchanges from one tank to a connecting tank occur spontaneously when the fluid level in one tank exceeds the height of the connecting pipe, as shown in the schematic from the top part of Figure 27. A fragment of the system's DBN model is presented in Figure 26, which shows the connections between the continuous system variables and the discrete fault variables (shown as D for burst/drift faults and

E for measurement faults) for a system of two tanks. For faults that persist over time, such as drift faults, the fault variable D_t will be dependent on D_{t-1}, its copy at time $t-1$, as shown graphically by the arrow from D_{t-1} to D_t.

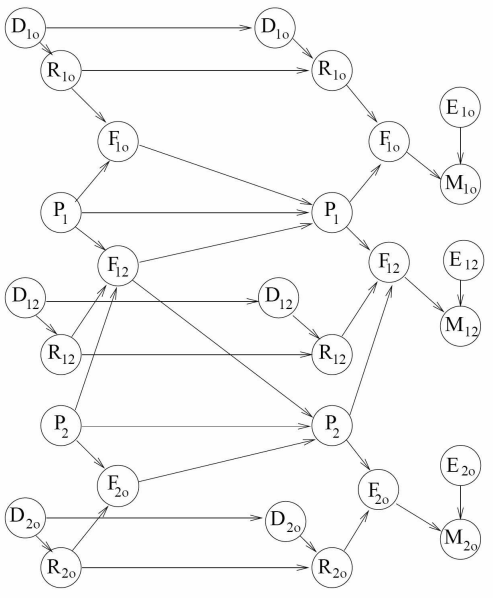

Figure 26: DBN of a two-tank system. The fault variables are denoted by D and E. Reproduced from [Lerner *et al.*, 2000].

At any point in time, the number of different faults that can occur is 2^{27}. As a result, approximate inference was performed on the DBN model, where the model was decomposed into 5 subsystems and each tank comprises a subsystem. Nonetheless, to detect momentary faults, whose direct effects are not observable upon its onset but are only observable after a short delay, a *smoothing* procedure was also required. In this inference procedure, $P(\mathbf{Z}_t|\mathbf{y}_{1:t+\tau})$ is used instead of $P(\mathbf{Z}_t|\mathbf{y}_{1:t})$ for fault detection. The intuition is that, by taking into account observations that occur τ time steps *after* the onset of the faults $\mathbf{Z}_t$, there should be more evidence to support the presence of these faults and thus detection rate should be improved. The empirical results from this study supports this intuition, as shown by the remarkable state estimation of the hidden variable "Conductance", as shown in Figure 27. Only the state estimation results are shown because the state estimation performance is directly related to fault detection performance. Faults affect the system behavior, and as a result, if the faults were not promptly detected, one would see errors in the state estimate.

The experiment was run on the full 5-tank system and the observations were generated from a handcrafted scenario where many different and multiple faults were injected between time 5 and 25. To illustrate the improvement that smoothing brings to the detection accuracy, we examine a particular fault, as presented in [Lerner *et al.*, 2000]. At time $t = 5$, a drift fault was introduced to the variable R_{23}. Upon its onset, the probability for a drift fault was only 0.012%. At time $t = 6$, the probability jumped to 71.7%, and was further increased to 99.9% after the smoothing procedure. At this point, the algorithm had correctly detected this fault and maintained a high probability until the effects of this drift wore off. Thus, this study demonstrates that DBNs are useful in fault diagnosis, and the smoothing procedure, combined with the subsystem approximations, shows much success in tracking a complex system, in the presence of faults with a small number of measurements.

In addition to fault diagnosis, DBNs have also been applied to the emergent field of privacy intrusion detection [An *et al.*, 2006]. Privacy intrusion is the act of illegitate disclosure or general misuse of private data by human agents who are entrusted with this sensitive information (e.g., employees from a government's revenue service or employees from a medical laboratory). With the growth of information technology, most business organizations collect private information about their clients and it is the responsibility of each organization to monitor and to detect possible misuses of private data by its agents. While privacy intrusion detection has been implemented as comparing the agent's

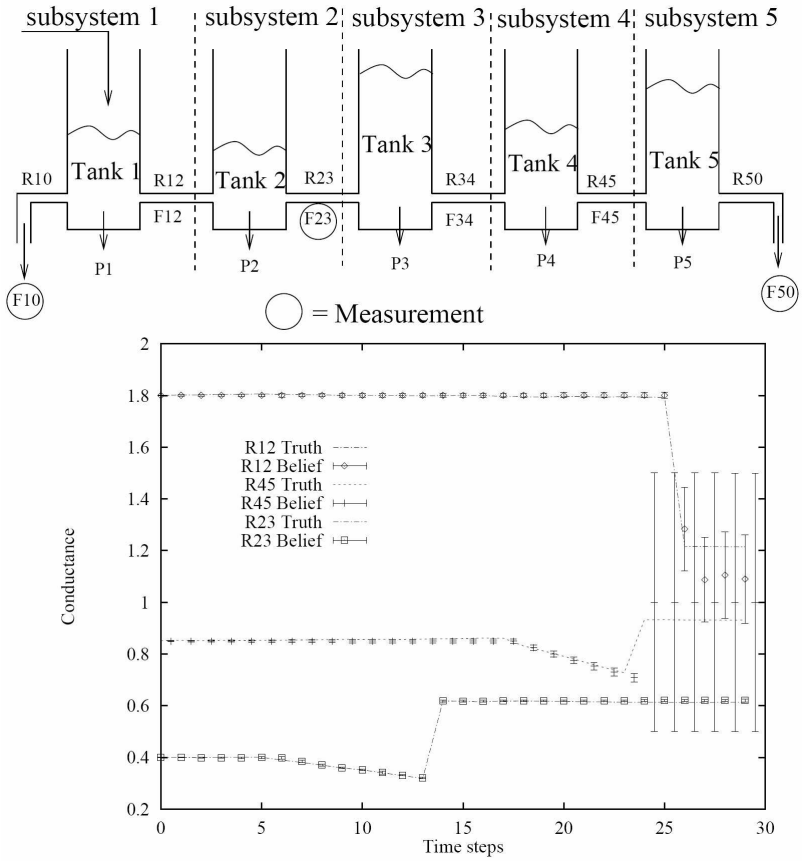

Figure 27: Schematic of the five-tank system (top) and the fault diagnosis results (bottom). Reproduced from [Lerner *et al.*, 2000].

behavior against his or her profile of normal behavior, this is not sufficient for misuse detection, because simply tracking the amount of time or the frequency in which an agent accesses a particular information database can lead to many false alarms, since the agent may have legitimate work-related reasons to do so. As a result, this study applied DBNs to combine various domain-specific features to establish a measure for the *degree of suspiciousness* for an intrusion.

In general, this is a sensible approach because the activities of an agent constitutes a stochastic process. The agent may be assigned a task that takes a prolonged period of time and the actions taken to fulfill this task are likely to be causally related. The study presented a DBN (shown in Figure 28) that was tailored for a government's revenue service, although the same modelling paradigm can be adapted for other industries.

In Figure 28, each box contains the random variables that are specific to one time slice. In each slice, the variables are defined as follows:

- F_d: the usage frequency of databases
- F_r: the usage frequency of records
- T_r: the amount of time spent on records
- M_r: indication for modification of records
- Tk: type of task performed (audit or collection/delivery)
- $Intr$: indication for intrusion of privacy
- Hrs: indication of whether work was performed during business hours
- A_r: the amount of records

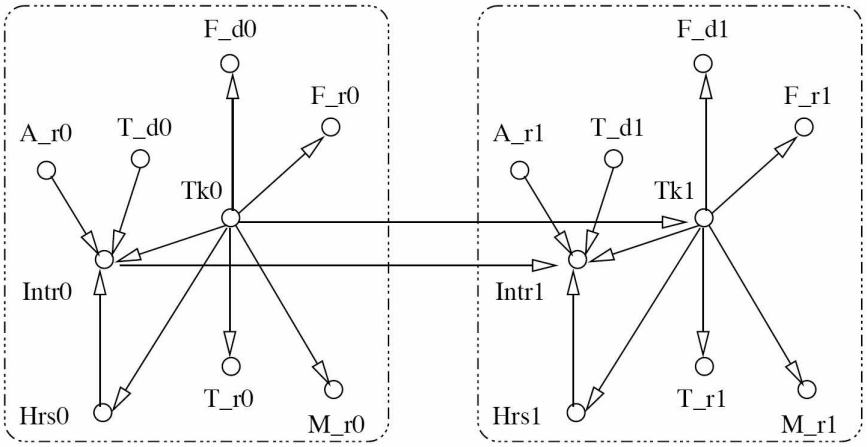

Figure 28: A DBN for privacy intrusion detection. Reproduced from [An *et al.*, 2006].

- T_d: the amount of time spent on databases

The arrow from $Tk0$ to $Tk1$ represents the evolution of the agent's tasks and the arrow from $Intru0$ to $Intru1$ represents the evolution of the agent's privacy intrusion. In this model, it is assumed that the agent will be more likely to intrude if he or she is engaging in intrusive activities currently. Analogously, the longer an agent refrains from intrusive activities, the less likely the agent will intrude in the current time slice.

The study verified the DBN by commonsense validation through probable scenarios. Probably due to sensivity of real-life data, no actual detection results were presented. Nonetheless, the approach outlined in this study is quite applicable for intrusion detection, especially if the attack involves the theft of large amount of private data. Moreover, if a large amount of data being accessed by an agent is irrelevant to the agent's job, the DBN will perform especially well because irrelevancy is directly modelled into the DBN.

Lastly, we examine a recent application of DBNs for traffic incident detection [Singliar and Hauskrecht, 2006]. Traffic incident detection is an important practical problem because the cost of highway accidents can be significantly reduced by their prompt detection. The study examined the performance of simple univariate detectors that performed thresholding on each feature, and compared the combination of these simple detectors to a support vector machine (SVM) (please refer to Subsection 4.2 for details). The SVM method was chosen because SVMs generalize the linear discriminators implemented by the thresholding detectors. The algorithms were evaluated on real-life traffic data collected from the most accident-prone segment of a highway in Pittsburg. The data consisted of roadway statistics, such as the average speed, the volume (number of passing vehicles) and occupancy (the traffic density), collected over a period that ranges from 30 seconds to 5 minutes.

It was found that an SVM approach generally outperforms the benchmark detection algorithm, the California TSC-2 model. The TSC-2 model uses a sequence of thresholds to determine the differences and proportions between lane occupancy from one time step to the next. Empirically, the benchmark model could detect only a third of the incidents at best. Compared to the SVM model, the TSC-2 typically resulted in lower ROC AUC (area under the Receiver Operator Characteristic curve) values. But at low false alarm rates, TSC-2 outperformed the SVM, due to the support from evidence in the last time step. As a result, a persistence check was incorporated as part of the SVM, which increased its performance dramatically, as shown in Figure 29.

This suggests that a temporal framework, in which a detector is "dynamized", might more appropriate for this problem domain. As a result, a DBN model (shown in Figure 30) was proposed. This model contains a single hidden discrete-state variable C (synonymous with the unknown class) and a number of conditionally independent univariate Gaussian observation variables $\mathbf{O} = O_1, ..., O_n$. In addition, there is also a binary observation variable I, which is the incident state as observed by the traffic management center. Following our notation, the conditional distribution $p(I_t|C_t)$ is

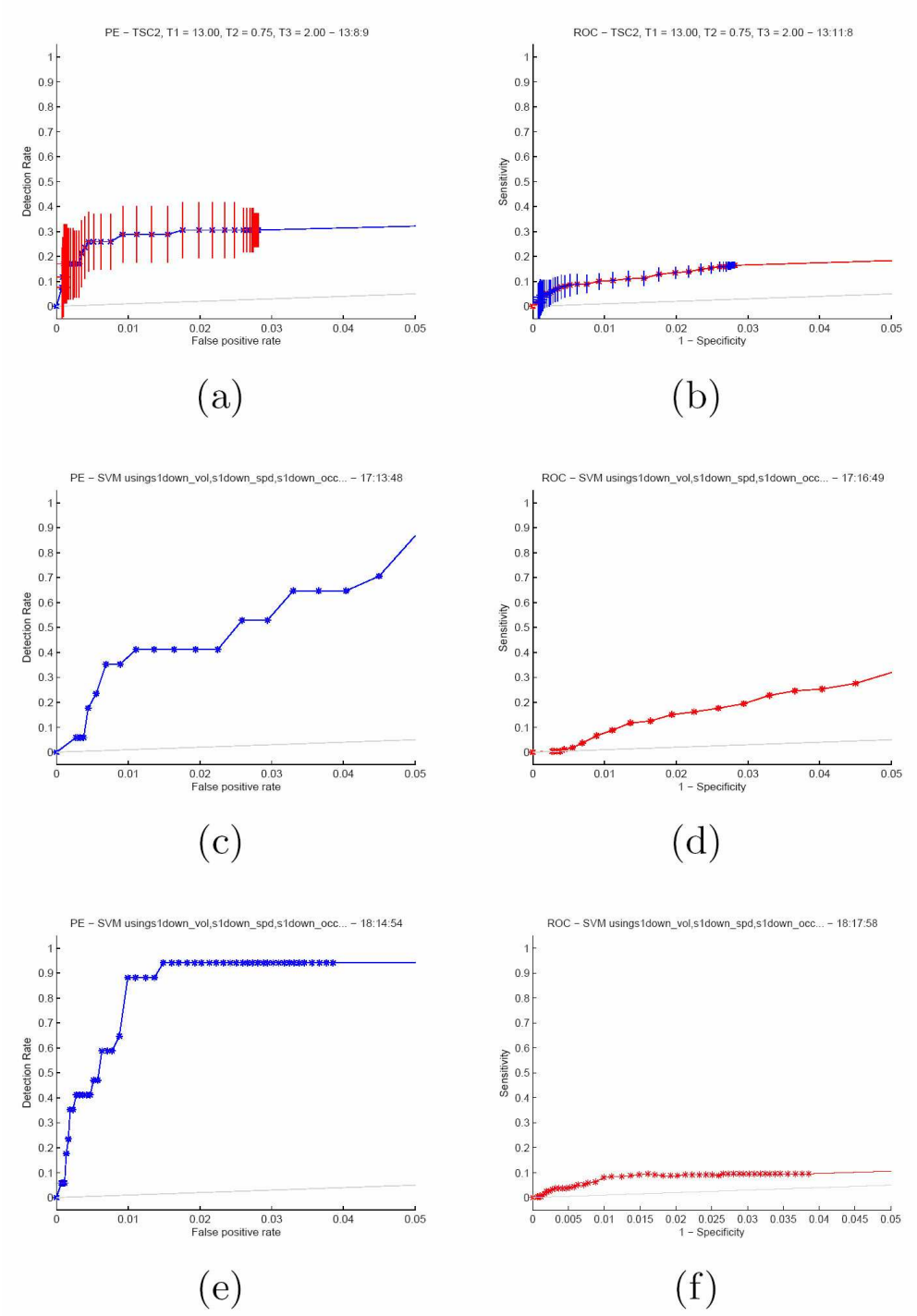

Figure 29: Comparison of SVM and the benchmark detector at low false alarm rates. The left plots are shown as detection rate (y-axis) vs. false positive rate (x-axis). The right plots are ROC curves, shown as the true positivity probability or *sensitivity* (y-axis) vs. the false positive probability defined by one minus the *specificity* (x-axis). Plots (a) and (b) show the performance of the benchmark detector, the California TSC-2. Plots (c) and (d) show the performance of the SVM. Plots (e) and (f) show the performance of the improved SVM that includes a persistence check, in which an incident must be detected in two consecutive time points before an alarm is raised. Reproduced from [Singliar and Hauskrecht, 2006].

handcrafted (see [Singliar and Hauskrecht, 2006] for details) while the conditional probability distributions $\{p(O_{i,t}|C_t)\}_{i=1}^{n}$ are learned from data. An alarm is triggered at time t if

$$p(C_t = \text{“accident effect buildup”}|\mathbf{O}_{1:t}, I_{1:t}) + p(C_t = \text{“accident steady state”}|\mathbf{O}_{1:t}, I_{1:t}) \geq \alpha \tag{26}$$

where α is some preset threshold.

The performance for the DBN-based approach is shown in Figure 31. Unfortunately, the DBN approach only achieved a AUC ROC of 0.568381, compared to the 0.810531 that was achieved by the SVM. This underperformance might be attributed to the fact that the structure of the DBN might not be the best fit for the data. With a more complex DBN model, performance can be improved. Nonetheless, this work paved the way for using DBNs in this direction of anomaly detection.

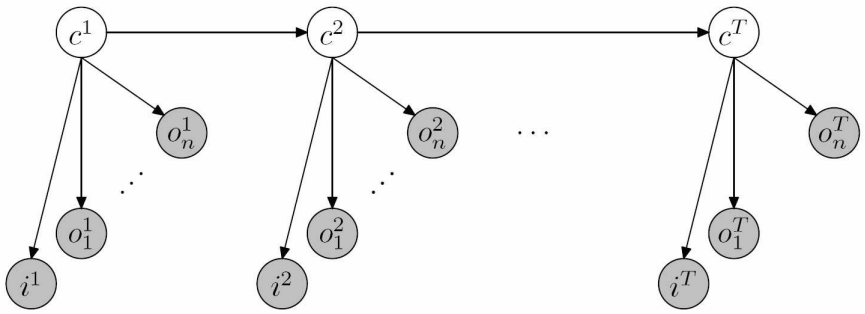

Figure 30: A DBN for traffic incident detection. Reproduced from [Singliar and Hauskrecht, 2006].

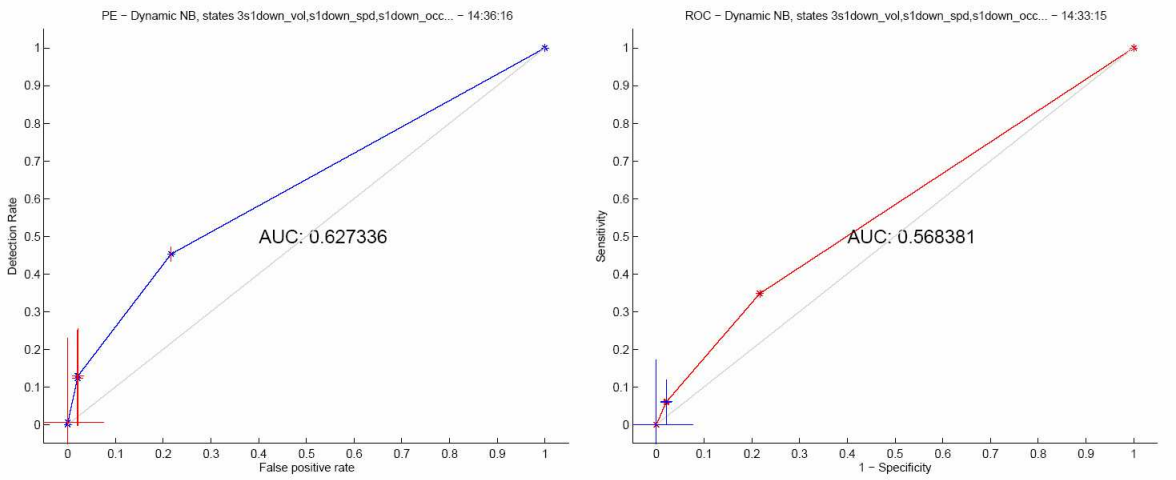

Figure 31: The performance of the DBN-based detector. The DBN used as observations the traffic sensors' measurements, as well as their differences and proportions. Reproduced from [Singliar and Hauskrecht, 2006].

6 Conclusion

In this survey, we presented an overview of popular discriminative and generative methods that have been applied to applications of classification and anomaly detection. Each approach—discriminative or generative—has its own advantages and disadvantages. The major trade-offs between the two approaches are as follows:

- Generative models tend to be biased towards those that maximize the likelihood of training data while the discriminative models are free from bias errors due to any misrepresentation of the input distribution.
- Generative models usually offer more insights about the structure of the system while discriminative models can be hard to interpret, as most treat the system dynamics as a complete black box.
- Discriminative models require more training data for the parameters to converge. As a result, in the case of sparse data and a reasonable amount of domain knowledge, it might make more sense to use generative models.

Further empirical comparisons can be found in:

- [Chan *et al.*, 2002]: in which neural networks, support vector machines, linear/quadratic discriminant analysis, Parzen windows, mixture of Gaussians, and mixture of generalized Gaussians are compared in a case study for glaucoma diagnosis;
- [Ulusoy and Bishop, 2005]: in which a discriminative model (based on logistic regression) and a generative model (based on mixture of Gaussians) are compared in the task of patch labelling and object recognition on weakly labelled data of animal pictures, as shown in Figure 32;
- [Pernkopf and Bilmes, 2005]: in which discriminative and generative parameter learning on both discriminatively and generatively structured Bayesian network classifiers are compared on a variety of benchmark data sets from [Newman *et al.*, 1998] and [Kohavi and John, 1997].

In addition, recent developments in hybridizing the two approaches have shown promising results over either one of the methods. [Bouchard and Triggs, 2004] presents a simple way of combining the two methods by interpolating linearly between the discriminative and generative objective functions during parameter learning. [Jaakkola and Haussler, 1998] combines the two methods by using a SVM as the basis for the discriminative classifier, but deriving the kernel functions from a generative model. [Raina *et al.*, 2003] presents a truly hybrid model whereby some parameters are trained to maximize the generative likelihood while other parameters are discriminatively trained to maximize the conditional likelihood. As data become more high-dimensional and complex, it is clear that one single method will not be sufficient and that an adaptive hybridization scheme will prove to be useful in the field of anomaly detection of dynamic processes. It is our hope that this survey will be able to facilitate and advocate future developments in this area.

Acknowledgments

This work was supported by FY'06 funding of the Predictive Knowledge Systems Strategic Initiative. In addition, much thanks to Vera Bulaevskaya, Barry Chen, Tina Eliassi-Rad, Bill Hanley, Gardar Johannesson, Phaedon-Stelios Koutsourelakis, and Carol Meyers for their insightful comments and discussions.

References

[Aggarwal, 2002] Charu C. Aggarwal. Towards meaningful high-dimensional nearest neighbor search by human-computer interaction. In *Proc. of the 18th International Conference on Data Engineering (ICDE)*, pages 593–604. IEEE Computer Society, 2002.

[An *et al.*, 2006] Xiangdong An, Dawn Jutla, and Nick Cercone. Privacy intrusion detection using dynamic bayesian networks. In *Proc. of International Conference on Electronic Commerce*, pages 208–215, New York, NY, USA, 2006. ACM Press.

[Beyer *et al.*, 1999] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? *Lecture Notes in Computer Science*, 1540:217–235, 1999.

[Bilmes, 1997] Jeff Bilmes. A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical Report ICSI-TR-97-021, University of California, Berkeley, April 1997.

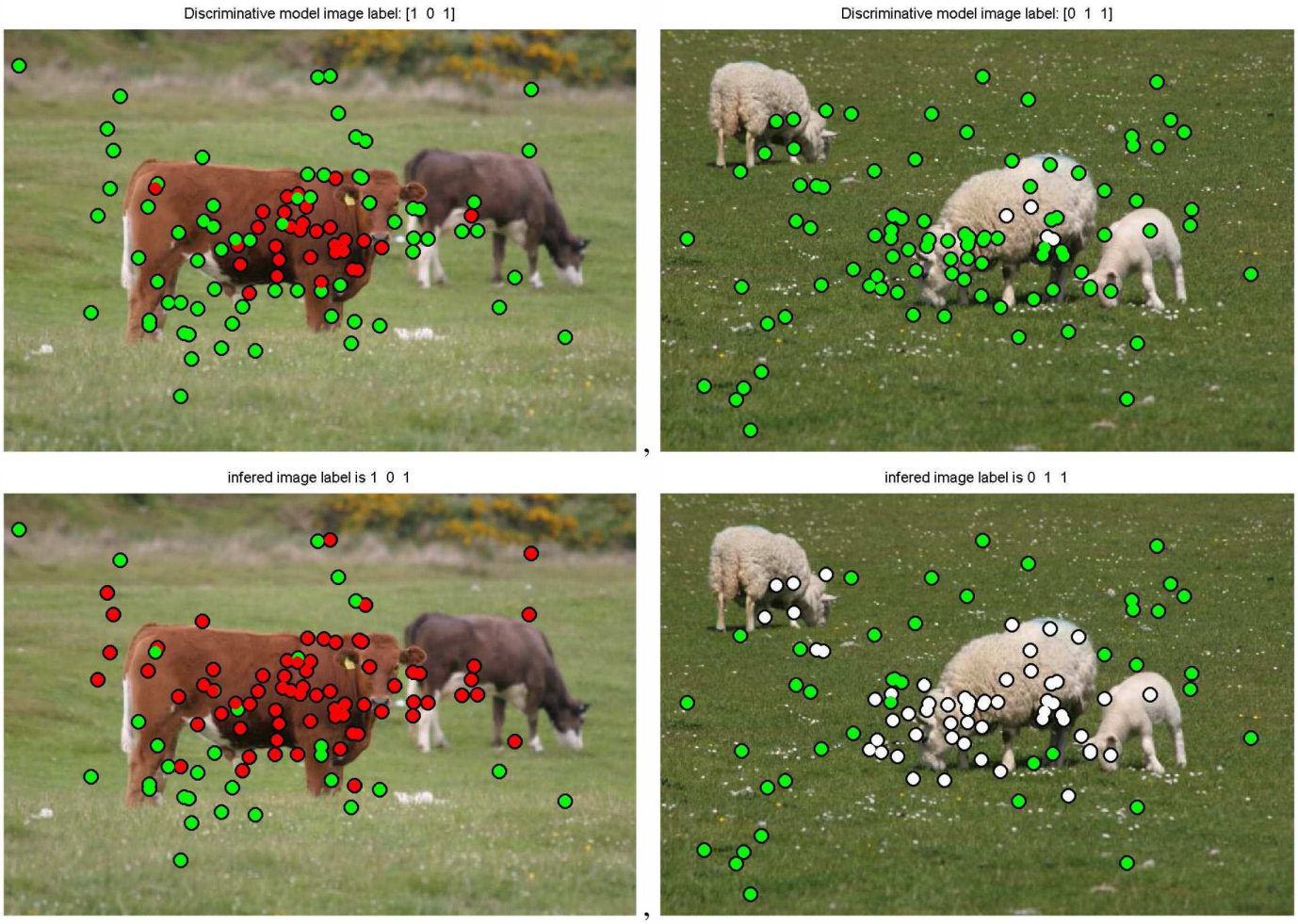

Figure 32: Patch labelling result for random patches. The discriminative results are shown on the top row and the generative results are shown on the bottom row. The red, white, and green dots denote the respective labelling of cows, sheeps and background. Reproduced from [Ulusoy and Bishop, 2005].

[Bouchard and Triggs, 2004] Guillaume Bouchard and Bill Triggs. The trade-off between generative and discriminative classifiers. In *Proceedings to IASC International Symposium on Computational Statistics (CompStat)*, 2004.

[Breunig *et al.*, 2000] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: identifying density-based local outliers. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 93–104, 2000.

[Burges, 1998] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[Chan *et al.*, 2002] Kwokleung Chan, Te-Won Lee, Pamela A. Sample, Michael H. Goldbaum, Robert N. Weinreb, and Terrence J. Sejnowski. Comparison of machine learning and traditional classifiers in glaucoma diagnosis. *IEEE Transactions on Biomedical Engineering*, 49(9):963–974, September 2002.

[Cover and Hart, 1967] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[de Kleer *et al.*, 1992] Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

[Dean and Kanazawa, 1989] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 1989.

[Dempster *et al.*, 1977] Arthur Dempster, Nan Laird, and Donald Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[Duda *et al.*, 2001] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, November 2001.

[Finkbeiner *et al.*, 2004] Douglas P. Finkbeiner, Nikhil Padmanabhan, and David J. Schlegel. Sloan digital sky survey imaging of low galactic latitude fields: Technical summary and data release. *Astronomical Journal*, 128:2577, 2004.

[Forrest and Hofmeyr, 1998] Stephanie Forrest and Steven A. Hofmeyr. UNM repository of computer immune systems data sets, 1998.

[Gilks *et al.*, 1995] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Interdisciplinary Statistics Series. Chapman & Hall/CRC, December 1995.

[Godfrey *et al.*, 1994] J. Godfrey, D. Graff, and A. Martin. Public databases for speaker recognition and verification. In *Proc. of ESCA Workshop on Automatic Speaker Recognition, Identification and Verification*, pages 39–42, 1994.

[Haykin, 1998] Simon Haykin. *Neural Networks: A Comprehensive Foundation (2nd Edition)*. Prentice Hall, July 1998.

[Hettich and Bay, 1999] S. Hettich and S. D. Bay. The uci kdd archive: Kdd cup 1999 data set. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, 1999.

[Hu *et al.*, 2003] Wenjie Hu, Yihua Liao, and V. Rao Vemuri. Robust support vector method for anomaly detection in computer security. In *International Conference on Machine Learning and Applications*, 2003.

[Hull, 1994] J. J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994.

[Jaakkola and Haussler, 1998] Tommi S. Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Advances in Neural Information Processing Systems 11 (NIPS)*, 1998.

[Knorr *et al.*, 2000] Edwin M. Knorr, Raymond T. Ng, and Vladimir Tucakov. Distance-based outliers: algorithms and applications. *The International Journal on Very Large Databases (VLDB)*, 8(3-4):237–253, 2000.

[Kohavi and John, 1997] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1–2):273–324, 1997.

[Lane, 1999] Terran Lane. Hidden markov models for human/computer interface modeling. In *Proc. of the IJCAI-99 Workshop on Learning About Users*, pages 35–44, 1999.

[Lazerevic *et al.*, 2003] A. Lazerevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the Third SIAM International Conference on Data Mining*, 2003.

[LeCun *et al.*, 1990] Y. LeCun, B. Boser, J. S Denker, D. Henderson, R. E.. Howard, W. Howard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, pages 396–404, San Mateo, CA, 1990. Morgan Kaufmann.

[Lee and Lewicki, 2000] T. Lee and M. Lewicki. The generalized gaussian mixture model using ica. In *Proc. of International Workshop on Independent Component Analysis and Blind Signal Separation*, pages 239–244, 2000.

[Lerner *et al.*, 2000] U. Lerner, R. Parr, D. Koller, and G. Biswas. Bayesian fault detection and diagnosis in dynamic systems. In *Proc. of the 17th AAAI*, 2000.

[Liao and Vemuri, 2002] Yihua Liao and V. Rao Vemuri. Use of k-nearest neighbor classifier for intrusion detection. *Computers & Security*, 21(5):439–448, 2002.

[Lincoln Laboratory, 1998] MIT Lincoln Laboratory. Darpa intrusion detection evaluation data sets. `http://www.ll.mit.edu/IST/ideval/data/data_index.html`, 1998.

[Markou and Singh, 2003] Markos Markou and Sameer Singh. Novelty detection: a review–part 2: Neural network based approaches. *Signal Process*, 83(12):2499–2521, 2003.

[Markou and Singh, 2006] Markos Markou and Sameer Singh. A neural network-based novelty detector for image sequence analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1664–1677, 2006.

[Mitchell, 2005] Tom Mitchell. Generative and discriminative classifiers: Naive bayes and logistic regression, September 2005. Draft chapter to the second edition of *Machine Learning*.

[Newman *et al.*, 1998] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998.

[Ng and Jordan, 2001] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems 14 (NIPS)*, 2001.

[Nichol *et al.*, 2000] R. C. Nichol, C. A. Collins, and S. L. Lumsden. The Edinburgh/Durham Southern Galaxy Catalogue - IX. The Galaxy Catalogue. *ArXiv Astrophysics e-prints*, August 2000.

[Ourston *et al.*, 2003] D. Ourston, S. Matzner, W. Stump, and B. Hopkins. Applications of hidden markov models to detecting multi-state network attacks. In *Hawaii International Conference on System Sciences*, 2003.

[Pelleg and Moore, 2005] D. Pelleg and A. W. Moore. Active learning for anomaly and rare-category detection. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17 (NIPS)*, pages 1073–1080, Cambridge, MA, 2005. MIT Press.

[Pernkopf and Bilmes, 2005] Franz Pernkopf and Jeff Bilmes. Discriminative versus generative parameter and structure learning of bayesian network classifiers. In *International Conference on Machine Learning 22 (ICML)*, pages 657–664, 2005.

[Rabiner, 1989] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proc. of IEEE*, 77(2):257–286, February 1989.

[Raina *et al.*, 2003] Rajat Raina, Yirong Shen, Andrew Y. Ng, and Andrew McCallum. Classification with hybrid generative /discriminative models. In *Advances in Neural Information Processing Systems 16 (NIPS)*, 2003.

[Ramaswamy *et al.*, 2000] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *Proc. of ACM SIGMOD International Conference on Management of Data*, 2000.

[Reynolds and Rose, 1995] Douglas A. Reynolds and Richard C. Rose. Robust text-independent speaker identification using gaussian mixture speaker models. *IEEE Transactions on Speech and Audio Processing*, 3(1):72–83, January 1995.

[Ryan *et al.*, 1998] Jake Ryan, Meng-Jang Lin, and Risto Miikkulainen. Intrusion detection with neural networks. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[Schölkopf *et al.*, 2000] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12 (NIPS)*, pages 582–588, Cambridge, MA, 2000. MIT Press.

[Singh *et al.*, 2000] S. Singh, M. Markou, and J. Haddon. Detection of new image objects in video sequences using neural networks. In *Proc. of the SPIE conference on Applications of Artificial Neural Networks in Image Processing*, pages 204–213, January 2000.

[Singliar and Hauskrecht, 2006] Tomas Singliar and Milos Hauskrecht. Towards a learning traffic incident detection system. In *Proc. of Workshop on Machine Learning for Surveillance and Event Detection, International Conference on Machine Learning*, 2006.

[Steinwart *et al.*, 2005] Ingo Steinwart, Don Hush, and Clint Scovel. A classification framework for anomaly detection. *Journal of Machine Learning Research*, 6:211–232, 2005.

[Tarassenko, 1995] L. Tarassenko. Novelty detection for the identification of masses in mammograms. In *Proc. of IEEE International Conference on Artificial Neural Networks*, volume 4, pages 442–447, 1995.

[Ulusoy and Bishop, 2005] I. Ulusoy and C. M. Bishop. Comparison of generative and discriminative techniques for object detection and classification. In C. S. J. Ponce, M. Herbert, and A. Zisserman, editors, *Proc. of Sicily Workshop on Object Recognition*, 2005.

[Viterbi, 1967] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.

[Williams and Nayak, 1996] B. Williams and P. Nayak. A model-based approach to reactive self-configuring systems. In *Proc. of the 13th AAAI and the 8th IAAI*, 1996.

[Wright *et al.*, 2004] Charles Wright, Fabian Monrose, and Gerald M. Masson. Hmm profiles for network traffic classification. In *Proc. of ACM workshop on visualization and data mining for computer security*, pages 9–15, New York, NY, USA, 2004. ACM Press.

[Yeung and Chow, 2002] Dit-Yan Yeung and Calvin Chow. Parzen-window network intrusion detectors. *Proc. International Conference on Pattern Recognition*, 4:40385, 2002.

[Yeung and Ding, 2003] Dit-Yan Yeung and Yuxin Ding. Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition*, 36(1):229–243, 2003.