Title:

On-line Unsupervised Outlier Detection Using Finite Mixtures with Discounting Learning Algorithms

Contact Author:

Kenji Yamanishi

Affiliation / Address:

Internet Systems Research Laboratories, NEC Corporation
4-1-1 Miyazaki, Miyamae, Kawasaki, Kanagawa 216-8555, Japan.
e-mail: k-yamanishi@cw.jp.nec.com
phone: +81-44-856-2143
fax: +81-44-856-2231

On-line Unsupervised Outlier Detection Using Finite Mixtures with Discounting Learning Algorithms

Kenji Yamanishi*, Jun-ichi Takeuchi*, Graham Williams** and Peter Milne**
*Internet Systems Research Laboratories, NEC Corporation, Japan
(k-yamanishi@cw.jp.nec.com, tak@ap.jp.nec.com)
**CSIRO Mathematical and Information Sciences, GPO Box 664, Canberra ACT 2601, Australia
(Graham.Williams@cmis.csiro.au, Peter.Milne@cmis.csiro.au)

October 2001

Abstract

Outlier detection is a fundamental issue in data mining, specifically in fraud detection, network intrusion detection, network monitoring, etc. SmartSifter is an outlier detection engine addressing this problem from the viewpoint of statistical learning theory. This paper provides a theoretical basis for SmartSifter and empirically demonstrates its effectiveness. SmartSifter detects outliers in an on-line process through the on-line unsupervised learning of a probabilistic model (using a finite mixture model) of the information source. Each time a datum is input SmartSifter employs an *on-line discounting learning algorithm* to learn the probabilistic model. A score is given to the datum based on the learned model with a high score indicating a high possibility of being a statistical outlier. The novel features of SmartSifter are: 1) it is adaptive to non-stationary sources of data; 2) a score has a clear statistical/information-theoretic meaning; 3) it is computationally inexpensive; and 4) it can handle both categorical and continuous variables. An experimental application to network intrusion detection shows that SmartSifter was able to identify data with high scores that corresponded to attacks, with low computational costs. Further experimental application has identified a number of meaningful rare cases in actual health insurance pathology data from Australia's Health Insurance Commission.

The material in this paper was presented in part at *the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* [25], August 20-23, 2000, Boston, MA, USA

1 Introduction

1.1 Outlier Detection Problem

The problem of outlier/anomaly detection is one of the most fundamental issues in data mining. Specifically, it is closely related to fraud detection problems such as credit-card fraud detection, network intrusion detection, fraudulent cellular call detection, insurance fraud etc., since criminal or suspicious activities may often induce statistical outliers.[24]

We focus on the issue of *on-line unsupervised outlier detection.* That is, we require the following conditions for outlier detectors:

1) *Outliers are detected based on unsupervised learning of the information source.* In general statistical approaches to fraud detection one first learns an underlying model of the mechanism for data-generation from examples and then evaluates how much a given input datum deviates from the model (see e.g., [2],[12]). We require here that the learning process be unsupervised, i.e., one has to learn the model under the situation where training examples are not labeled with regard to whether each datum is fraudulent or not. This is in contrast to the supervised learning approach (see, e.g., [4],[3],[18],[23]), in which one first learns classification rules from labeled examples to predict a label for future examples. Although the supervised learning based approach is popular in fraud detection and intrusion detection, the unsupervised learning approach is not only more technically difficult but also more practically important, since in real situations (A) a sufficient number of labeled examples might not be available, and (B) a new fraud or intrusion pattern which didn't appear in past data may possibly emerge in future data. There is relatively little work (e.g. [4]) on fraud detection based on unsupervised learning.

2) *The process is on-line.* That is, every time a datum is input, it is required to evaluate how large the datum is deviated relative to a normal pattern. In contrast, most existing work on outlier detection (e.g., [2],[13],[14],[17],[18],[23]) in statistics and data mining is concerned with batch-detection processes, in which outliers can be detected only after seeing the entire dataset. The on-line setting is more realistic when one deals with the tremendous amount of data in network monitoring, fraudulent cellular phone call detection, health insurance claims, etc. It is also more natural in such situations as

the data by nature becomes available over time and it is important to identify deviations as they arise.

Note that there exists relatively little previous work (e.g. [4]) focusing on the on-line unsupervised learning based approach. This paper, on the basis of statistical learning theory, introduces SmartSifter, which meets requirements 1) and 2) and has some superiority over the program in [4]. Moreover, through SmartSifter, we offer a general framework for on-line unsupervised outlier detection applicable to a variety of data mining tasks.

1.2 Overview of SmartSifter

The approach of SmartSifter is as follows:

I) SmartSifter uses a probabilistic model as a representation of an underlying mechanism of data-generation. The model takes a hierarchical structure. A *histogram density* with a number of cells is used to represent a probability density over the domain of categorical variables. For each cell a *finite mixture model* is used to represent the probability density over the domain of continuous variables.

II) Every time a datum is input SmartSifter employs an on-line learning algorithm to update the model. For the categorical domain we have developed the *SDLE (Sequentially Discounting Laplace Estimation)* algorithm for learning the histogram density (Sec.2.1). For the continuous domain we have developed the *SDEM (Sequentially Discounting Expectation and Maximizing)* algorithm (Sec.2.2) for learning the finite mixture model. The most important feature of these two algorithms is that they gradually *discount* the effect of past examples in the on-line process.

III) SmartSifter assigns a score to each input datum on the basis of the learned model, measuring the change to the model after learning. A high score indicates a high possibility that the datum is an outlier.

The novel features of SmartSifter are summarized as follows:

a) *SmartSifter is adaptive to non-stationary sources.* In conventional statistical approaches it is usually assumed that an underlying information source for data-generation is stationary [2]. This is, however, not realistic when one deals with drifting sources or time-series data. The discounting algorithm employed by SmartSifter learns from the

source and forgets out-of-date statistics of the data using a decay factor. Thus it is expected to be adaptive to non-stationary sources.

b) *A score calculated by SmartSifter has a clear statistical/information-theoretic meaning.* In most previous work a score is calculated using heuristics such as cost [5],[9], lacking statistical justification, while the Mahalanobis distance [2] and the quadratic distance [9],[13] have been used to score outliers in some work. SmartSifter defines a score for a datum in terms of a statistical distance. This distance measures the changes in the distribution learned from the data before and after the new datum is incorporated. Thus it is natural to interpret that a datum of high score is, with high probability, an outlier.

c) *SmartSifter is computationally inexpensive.* The computational complexity of SmartSifter for calculating a score for each datum is linear in the number of parameters in the model and cubic in the number of variables. For example, SmartSifter can process about 90,000 data with four attributes in 28 seconds (Pentium III 550MHz). d) *SmartSifter can deal with both categorical and continuous variables.* To our knowledge SmartSifter is the first on-line unsupervised outlier detector that can deal with both categorical and continuous variables.

The design of SmartSifter was inspired by the work by Burge and Shawe-Taylor [4]. Our work differs from [4] in the following regards: 1) SmartSifter treats both categorical and continuous variables while [4] deals only with continuous variables. 2) While [4] uses two models in the algorithm: the long term model and the short term model, SmartSifter unifies them into one model resulting in a clearer statistical meaning and lower computational cost. 3) SmartSifter uses either a parametric representation for a probabilistic model or a kernel representation, while only a kernel representation is used in [4]. In Sec 3.1 we compare our parametric method with the kernel method to show that the former outperforms the latter both in accuracy and computation.

We empirically demonstrate the practical effectiveness of SmartSifter using the network intrusion dataset (KDD Cup 1999) [26]. Although it was used in the context of a supervised intrusion detector learning problem in KDD Cup 1999, we use it for on-line unsupervised intrusion detection. In our experiments, using a network access log dataset of 458,078 accesses including 1687 intrusions, SmartSifter detects 55% of all intrusions

within the top 1% of the datums ranked by the SmartSifter score. Further, 82% of all intrusions are included within the top 5% of SmartSifter ranked datums.

We also used a health insurance pathology dataset supplied by the Australian Health Insurance Commission to demonstrate that SS was able to identify several meaningful rare cases.

The rest of this paper is organized as follows: Section 2 describes algorithms developed for SS. Section 3 illustrates SmartSifter in action through its application to network intrusion detection (3.1), rare event detection in a medical insurance dataset (3.2), and simulation results (3.3). Section 4 contains our concluding remarks.

2 Outline of SmartSifter

2.1 Overall Flow of SmartSifter

Let $(\mathbf{x}, \mathbf{y})$ denote a datum where $\mathbf{x}$ denotes a vector of categorical variables and $\mathbf{y}$ denotes a vector of continuous variables. We write the joint distribution of $(\mathbf{x}, \mathbf{y})$ as $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x})p(\mathbf{y}|\mathbf{x})$. We represent $p(\mathbf{x})$ using a *histogram density* with a finite number of disjoint cells (Sec.2.1), and for each cell, for all $\mathbf{x}$'s that fall into it, we represent $p(\mathbf{y}|\mathbf{x})$ using an identical form of a *finite mixture model* (Sec.2.2). Hence we prepare as many finite mixture models as there are cells in the histogram density.

Consider the situation where a sequence of data is given in an on-line process: $(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2) \cdots$. The fundamental steps of SmartSifter are:

1. Given the tth input datum $(\mathbf{x}_t, \mathbf{y}_t)$ identify the cell that $\mathbf{x}_t$ falls into and update the histogram density using the SDLE algorithm (Sec. 2.1) to obtain $p^{(t)}(\mathbf{x})$. Then, for that cell, update the finite mixture model using the SDEM/SDPU algorithm (Sec.2.2) to obtain $p^{(t)}(\mathbf{y}|\mathbf{x})$. For other cells, set $p^{(t)}(\mathbf{y}|\mathbf{x}) = p^{(t-1)}(\mathbf{y}|\mathbf{x})$.
2. Calculate a score for the datum on the basis of the models before and after updating (Sec.2.3).

2.2 Categorical Variables

Below we describe how to learn the histogram density over the categorical domain. Let the number of categorical variables be n. Let the range of the ith categorical variable be $\mathcal{A}^{(i)} = \{a_1^{(i)}, \cdots, a_{u_i}^{(i)}\}$ $(i = 1, \cdots, n)$. Partition it into a finite number of disjoint sets: $\{A_1^{(i)}, \cdots, A_{v_i}^{(i)}\}$ $(i = 1, \cdots, n)$, where $A_j^{(i)} \cap A_k^{(i)} = \emptyset$ $(j \neq k)$ and $\mathcal{A}^{(i)} = \cup_{j=1}^{v_i} A_j^{(i)}$. This partitioning is done in order to reduce the complexity of the task. Within the resulting multidimensional space defined over the n categorical variables, each with v_i partitions, we identify the cell $A_{j_1}^{(1)} \times \cdots \times A_{j_n}^{(n)}$ as the $(j_1, \cdots, j_n)$th *cell*. We have $k = v_1 \times \cdots \times v_n$ cells in total. This provides a partitioning of the domain.

Given such a partitioning of the domain a *histogram density* forms a probability distribution which takes a constant value on each cell. The histogram density is specified by a parameter $\theta = (q_1, \cdots, q_k)$ where $\sum_{j=1}^{k} q_j = 1,\ q_j \geq 0$ and q_j denotes the probability value for the jth cell. If there are L_j categorical variables in the jth cell, the probability value of each symbol $\mathbf{x}$ in it is given by $p(\mathbf{x}) = q_j / L_j$.

We introduce here the SDLE algorithm. This is a variant of the Laplace law (1775) (see [15]) by which one estimates the probability value over a discrete domain with $\hat{p}(a) = (T_a + \beta)/(T + M\beta)$ for each domain element a where $0 < \beta < 1$, T is the size of a data sequence, T_a is the number of occurrences of a in the sequence, and M is the number of elements in the discrete domain. The SDLE algorithm is obtained by modifying the Laplace law so that (A) it is run on-line; and (B) it can *discount* the effect of past examples gradually.

The SDLE algorithm is specified by a discounting parameter $r_h (> 0)$ where a smaller r_h indicates that the model is influenced more by past examples (see Fig.1). The estimator $\hat{p}(a)$ in the Laplace law is replaced with

$$\hat{p}(a) = \frac{T_a + \beta}{(1 - (1 - r_h)^t)/r_h + k\beta}.$$

Usually r_h is set between 0.01 and 0.001. The Laplace law is reduced to the case of $r_h = 0$ in SDLE.

Computationally the total number of cells increases exponentially in the number of categorical variables. In order to avoid this computational difficulty we sometimes use

SDLE Algorithm

A partitioning of the domain $\{A_1^{(i)}, \cdots, A_{v_i}^{(i)}\}$ $(i = 1, \cdots, n)$, r_h, and β are given.

Step 1. /* Initialization */

Let $T(j_1, \cdots, j_n) = 0$ $(1 \leq j_i \leq v_i,\ i = 1, \cdots, n)$.

$t := 1$

Step 2. /* Parameter Updating */

while $t \leq T$ (T: sample size)

Read $\mathbf{x}_t = (x_1, \cdots, x_n)$

For each $(j_1, \cdots, j_n)$-th cell,

$$\begin{aligned} T_t(j_1, \cdots, j_n) &:= (1 - r_h) T_{t-1}(j_1, \cdots, j_n) + \delta_t(j_1, \cdots, j_n) \\ q^{(t)}(j_1, \cdots, j_n) &:= \frac{T_t(j_1, \cdots, j_n) + \beta}{(1 - (1 - r_h)^t)/r_h + k\beta} \end{aligned}$$

For each $\mathbf{x} \in A_{j_1}^{(1)} \times A_{j_2}^{(2)} \times \cdots A_{j_n}^{(n)}$,

$$p^{(t)}(\mathbf{x}) := \frac{q^{(t)}(j_1, \cdots, j_n)}{|A_{j_1}^{(1)}| \cdot |A_{j_2}^{(2)}| \cdots |A_{j_n}^{(n)}|}$$

where $\delta_t(j_1, \cdots, j_n) = 1$ if the t-th datum falls into the $(j_1, \cdots, j_n)$-th cell, and $\delta_t(j_1, \cdots, j_n) = 0$ otherwise.

$t := t + 1$

Figure 1: SDLE Algorithm

heuristics for margining cells in order to drastically reduce the number of cells. For example, we estimate the probability for each cell from training examples and then merge all cells whose probability values are lower than a predetermined threshold into a single cell.

2.3 Continuous Variables

Now we describe how to learn the model over the continuous domain. We consider two versions: a parametric version and a kernel version. The parametric version is referred to as SS while the kernel version is referred to as SS*.

2.3.1 Parametric Version

In the parametric version we employ as a finite mixture model over the continuous domain a *Gaussian mixture model* written as:

$$p(\mathbf{y}|\theta) = \sum_{i=1}^{k} c_i p(\mathbf{y}|\mu_i, \Lambda_i),$$

where k is a positive integer, $c_i \geq 0$, $\sum_{i=1}^{k} c_i = 1$ and each $p(\mathbf{y}|\mu_i, \Lambda_i)$ is a d-dimensional Gaussian distribution with density specified by mean μ_i and covariance matrix Λ_i:

$$p(\mathbf{y}|\mu_i, \Lambda_i) = \frac{1}{(2\pi)^{d/2}|\Lambda_i|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{y}-\mu_i)^T \Lambda_i^{-1}(\mathbf{y}-\mu_i)\right),$$

where $i = 1, \cdots, k$ and d is the dimension of each datum. We set $\theta = (c_1, \mu_1, \Lambda_1, \cdots, c_k, \mu_k, \Lambda_k)$. The Gaussian mixture is very popular in statistical modeling [19] because it is very expressive and its efficient learning algorithms have been extensively explored in the areas of statistics and machine learning.

First, we review the incremental EM algorithm [21] for learning Gaussian mixture models. Letting s be an iteration index, we define sufficient statistics $S_i^{(s)} (i = 1, \cdots, k)$ by

$$\begin{aligned} S_i^{(s)} &= \left(c_i^{(s)}, \bar{\mu}_i^{(s)}, \bar{\Lambda}_i^{(s)}\right) \\ &\stackrel{\text{def}}{=} \frac{1}{t} \cdot \Big(\sum_{u=1}^{t} \gamma_i^{(s)}(u), \sum_{u=1}^{t} \gamma_i^{(s)}(u) \cdot \mathbf{y}_u, \sum_{u=1}^{t} \gamma_i^{(s)}(u) \cdot \mathbf{y}_u \mathbf{y}_u^T\Big), \end{aligned}$$

where

$$\gamma_i^{(s)}(u) \stackrel{\text{def}}{=} \frac{c_i^{(s-1)} p(\mathbf{y}_u|\mu_i^{(s-1)}, \Lambda_i^{(s-1)})}{\sum_{i=1}^{k} c_i^{(s-1)} p(\mathbf{y}_u|\mu_i^{(s-1)}, \Lambda_i^{(s-1)})}. \tag{1}$$

We also define $S_i^{(s)}(v)$ $(i = 1, \cdots, k)$ for $\mathbf{y}_v$ by

$$S_i^{(s)}(v) \stackrel{\text{def}}{=} \frac{1}{t} \cdot \Big(\gamma_i^{(s)}(v), \gamma_i^{(s)}(v) \cdot \mathbf{y}_v, \gamma_i^{(s)}(v) \cdot \mathbf{y}_v \mathbf{y}_v^T\Big).$$

The incremental EM algorithm for Gaussian mixture models consists of the following E-step and M-step [21]:

E-step: Choose a datum $\mathbf{y}_u$ from the sequence $\mathbf{y}^t$. Given $\theta^{(s-1)}$, compute

$$S_i^{(s)}(u) = \frac{1}{t} \cdot \Big(\gamma_i^{(s)}(u), \gamma_i^{(s)}(u) \cdot \mathbf{y}_u, \gamma_i^{(s)}(u) \cdot \mathbf{y}_u \mathbf{y}_u^T\Big).$$

$$\text{Then, set } \; S^{(s)} = S^{(s-1)} - S^{(s-1)}(u) + S^{(s)}(u). \tag{2}$$

M-step: Compute the new estimate $\theta^{(s)}$ by

$$\mu_i^{(s)} = \bar{\mu}_i^{(s)}/c_i^{(s)} \text{ and } \Lambda_i^{(s)} = \bar{\Lambda}_i^{(s)}/c_i^{(s)} - \mu_i^{(s)}\mu_i^{(s)T}.$$

The point is that in the E-step the sufficient statistics $S^{(s-1)}$ is updated relative to an arbitrarily chosen $\mathbf{y}_u$. By repeating the iteration of the E and M steps w.r.t.s, $\theta^{(s)}$ converges.

We introduce here the SDEM algorithm by modifying the incremental EM algorithm as follows:

(A) Choose $\mathbf{y}_u$ in time order, i.e., choose $\mathbf{y}_s$ at the sth round in the E-step. Make only one iteration of the E and M steps for each s. This makes the parameters updated every time a datum is input.

(B) Introduce a discounting parameter r $(0 < r < 1)$ to modify the updating rule (2) into the following:

$$S_i^{(s)} := (1-r)S_i^{(s-1)} + r \cdot (\gamma_i^{(s)}(s), \gamma_i^{(s)}(s)\mathbf{y}_s, \gamma_i^{(s)}(s)\mathbf{y}_s\mathbf{y}_s^T),$$

This rule makes the statistics exponentially decay with a factor $(1-r)$ as the stage proceeds. Hence it can forget out-of-date statistics rapidly, and thus makes the convergence faster than the incremental EM algorithm.

Below we describe a general form of the modified algorithm:

SDEM algorithm

E-step: Given $S_i^{(s-1)}$, $\theta^{(s-1)}$ and $\mathbf{y}_s$, compute $S_i^{(s)}$ by (1) and (3).

M-step: Compute the new $\theta^{(s)}$ by (3).

Note: If the modification is (B) only, without (A), we obtain the algorithm proposed by Nolan (see e.g. [21]).

The details of the SDEM algorithm are in Fig. 2. in which a parameter α is introduced in order to improve the stability of the estimates of c_i, which is set to $1.0 \sim 2.0$. Usually $c_i^{(0)} = 1/k$ and $\mu_i^{(0)}$s are set so that they are uniformly distributed over the data space.

SDEM Algorithm $(r, \alpha, k$: given)
Step 1. /* Initialization */
Set $\mu_i^{(0)}, c_i^{(0)}, \bar{\mu}_i^{(0)}, \Lambda_i^{(0)}, \bar{\Lambda}_i^{(0)} (i = 1, ..., k)$.
$t := 1$
Step 2. /* Parameter Updating */
while $t \leq T$ (T:sample size)
Read $\mathbf{y}_t$
for $i = 1, 2, ..., k$

$$\gamma_i^{(t)} := (1 - \alpha r) \frac{c_i^{(t-1)} p(\mathbf{y}_t | \mu_i^{(t-1)}, \Lambda_i^{(t-1)})}{\sum_{i=1}^{k} c_i^{(t-1)} p(\mathbf{y}_t | \mu_i^{(t-1)}, \Lambda_i^{(t-1)})} + \frac{\alpha r}{k}$$

$$c_i^{(t)} := (1 - r) c_i^{(t-1)} + r\gamma_i^{(t)}$$

$$\bar{\mu}_i^{(t)} := (1 - r) \bar{\mu}_i^{(t-1)} + r\gamma_i^{(t)} \cdot \mathbf{y}_t$$

$$\mu_i^{(t)} := \bar{\mu}_i^{(t)} / c_i^{(t)}$$

$$\bar{\Lambda}_i^{(t)} := (1 - r) \bar{\Lambda}_i^{(t-1)} + r\gamma_i^{(t)} \cdot \mathbf{y}_t \mathbf{y}_t^T$$

$$\Lambda_i^{(t)} := \bar{\Lambda}_i^{(t)} / c_i^{(t)} - \mu_i^{(t)} \mu_i^{(t)^T}$$

$t := t + 1$

Figure 2: SDEM Algorithm

The computation time for the SDEM algorithm at each round is $O(d^3 k)$ where d is the dimension of the data and k is the number of Gaussian distributions.

The discounting parameter r is related to the degree of discounting past examples. Intuitively, smaller r is, a larger effect the SDEM algorithm has from past examples. It is easily checked that $c_i^{(t)}$ is the weighted sum of $\gamma_i^{(j)}$ w.r.t. j where the weight for j is $(1 - r)^{t-j} r$ and that $\mu_i^{(t)}$ and $\Lambda_i^{(t)}$ are the parameter values that maximize the weighted sum of log likelihood of $\log p(\mathbf{y}_j | \mu_i, \Lambda_i)$ w.r.t. j where the weight for j is $(1 - r)^{t-j} r\gamma_i^{(j)}$. In the case of $r = 1/t$, the SDEM algorithm is equivalent with the original EM algorithm.

2.3.2 Kernel Version

In the kernel version we employ as a finite mixture model over the continuous domain a *kernel mixture model*:

$$p(\mathbf{y}|q) = \frac{1}{K} \sum_{i=1}^{K} w(\mathbf{y} : q_i), \tag{3}$$

where K is given, $q = \{q_1, \cdots, q_K\}$ is called a set of *prototypes*, $w(\cdot : q_i)$ is a kernel function defined as a Gaussian density with mean q_i and variance matrix $\Sigma = diag(\sigma^2, \cdots, \sigma^2)$ for

a positive constant σ, and d is the dimension of a datum.

The difference between the parametric version and kernel one is that the coefficient vector and the variance matrix for a finite mixture are variable in the former version, while they are fixed in the latter one. In general the number of prototypes for the kernel version should be set much larger than the mixture size for the parametric one.

We introduce here the *SDPU (Sequentially Discounting Prototype Updating) algorithm* for on-line learning prototypes in the kernel mixture. For a given data sequence $\mathbf{y}^t = \mathbf{y}_1 \cdots \mathbf{y}_t$, first define:

$$f(\mathbf{y}|\mathbf{y}^t) = \sum_{\tau=1}^{t} A(t,\tau) w(\mathbf{y} : \mathbf{y}_\tau),$$

where $A(t,t) \stackrel{\text{def}}{=} r$, $A(t,\tau) \stackrel{\text{def}}{=} r(1-r)^{t-r-1}$ for $\tau \leq t-1$ and $0 < r < 1$ is a discounting factor. Note that $\sum_{\tau=1}^{t} A(t,\tau) = 1$ holds. Hence $f(\mathbf{y}|\mathbf{y}^t)$ is a weighted sum of $w(\mathbf{y} : \mathbf{y}_\tau)$ $(\tau = 1, \cdots, t)$ where the weight becomes large as τ increases. Next define the square error of $p(\mathbf{y}|q)$ to $f(\mathbf{y}|\mathbf{y}^t)$ by

$$\bar{\epsilon}^2(q : \mathbf{y}^t) = \int \Big(p(\mathbf{y}|q) - f(\mathbf{y}|\mathbf{y}^t)\Big)^2 d\mathbf{y}.$$

For a new input $\mathbf{y}_{t+1}$, the SDPU algorithm updates a prototype $q^{(t)}$ into $q^{(t)} + \Delta q^{(t)}$ by choosing $\Delta q^{(t)}$ so that $\bar{\epsilon}^2(q^{(t)} + \Delta q^{(t)} : \mathbf{y}^t \mathbf{y}_{t+1})$ is minimal under the condition that $\bar{\epsilon}^2(q : \mathbf{y}^t)$ is minimized by $q = q^{(t)}$.

Several steps (omitted here) lead that $\Delta q^{(t)}$ must satisfy linear equations

$$\sum_{j,m} C^{(t)}_{jmkl} \Delta q^{(t)}_{jm} = B^{(t)}_{kl} \quad (k = 1, ..., K, l = 1, ..., d), \tag{4}$$

where $B^{(t)}_{kl} \stackrel{\text{def}}{=} r_t \Big(K \cdot (x_{t+1,l} - q^{(t)}_{kl}) \exp(-\frac{|x_{t+1} - q^{(t)}_k|^2}{4\sigma^2}) - \sum_{i=1}^{K} (q^{(t)}_{il} - q^{(t)}_{kl}) \exp(-\frac{|q^{(t)}_i - q^{(t)}_k|^2}{4\sigma^2}) \Big)$ and $C^{(t)}_{jmkl} \stackrel{\text{def}}{=} (\delta_{ml} - \frac{(q^{(t)}_{kl} - q^{(t)}_{jl})(q^{(t)}_{km} - q^{(t)}_{jm})}{2\sigma^2}) \exp(-\frac{|q^{(t)}_k - q^{(t)}_j|^2}{4\sigma^2})$. Here $q^{(t)}_{kl}$ denotes the lth component of $q^{(t)}_k$. Thus $\Delta q^{(t)}$ is obtained by solving (4). This can be done using a standard algorithm for solving linear equations. The details of the SDPU algorithm are shown in Fig.3.

Note: The computation time for the SDPU algorithm at each round is $O(d^3K^3)$ where d is the dimension of data and K is the number of prototypes. The SDPU algorithm coincides with Grabec's algorithm [10] in the case where we let r be time-dependent as $r = 1/t$.

SDPU Algorithm (r, σ, K: given)

Step 1. /* Initialization */

Set $q_i^{(0)} (i = 1, ..., K)$ so that they are uniformly distributed.

$t := 1$

Step 2 /* Prototype Updating */

while $t \leq T$ (T:sample size)

Read $\mathbf{x}_t$

for all (j, m, k, l)

$B_{kl}^{(t)} := r\Big(K \cdot (x_{t+1,l} - q_{kl}^{(t)}) \exp(-\frac{|x_{t+1} - q_k^{(t)}|^2}{4\sigma^2})$

$- \sum_{i=1}^{K} (q_{il}^{(t)} - q_{kl}^{(t)}) \exp(-\frac{|q_i^{(t)} - q_k^{(t)}|^2}{4\sigma^2})\Big)$

(Here $q_{kl}^{(t)}$ denotes the lth component of $q_k^{(t)}$.)

$C_{jmkl}^{(t)} := (\delta_{ml} - \frac{(q_{kl}^{(t)} - q_{jl}^{(t)})(q_{km}^{(t)} - q_{jm}^{(t)})}{2\sigma^2}) \exp(-\frac{|q_k^{(t)} - q_j^{(t)}|^2}{4\sigma^2})$

Solve the linear equation: $\sum_{j,m} C_{jmkl}^{(t)} \Delta q_{jm}^{(t)} = B_{kl}^{(t)}$

$(k = 1, ..., K,\ l = 1, ..., d)$ for all (j, m)

$q_{jm}^{(t+1)} := q_{jm}^{(t)} + \Delta q_{jm}^{(t)}$

$t := t + 1$

Figure 3: SDPU Algorithm

2.3.3 Burge and Shawe-Taylor's Algorithm

The kernel version of SmartSifter can be thought of as a variant of Burge and Shawe-Taylor's algorithm [4] in the sense that they are both based on Grabec's algorithm. Major differences between them are 1) methods of discounting past examples and 2) methods of score calculation.

Below we briefly review Burge and Shawe-Taylor's algorithm. It first estimates the prototypes using the original Grabec's algorithm (that is, $r = 1/t$ in the SDPU algorithm). Unlike the SDPU algorithm, however, it doesn't solve the linear equation (4) directly but employs the following iteration method in order to update the prototypes.

$$\Delta q_{jm}^{(t,s+1)} := B_{jm}^{(t)} - r_t \sum_{k \neq j}^{K} \sum_{l \neq m}^{n} C_{jmkl}^{(t)} \Delta q_{kl}^{(t,s)}.$$

where s denotes an index indicating the number of iteration, and $\Delta q^{(t,0)} = B^{(t)}$. We define here a *profile* $v^{(t+1)} = (v_1^{(t+1)}, \cdots v_K^{(t+1)})$ of data $\mathbf{y}_{t+1}$ by

$$v_j^{(t+1)} \stackrel{\text{def}}{=} \frac{\exp(-|\mathbf{y}_{t+1} - q_j^{(t)}|)}{\sum_{j=1}^{K} \exp(-|\mathbf{y}_{t+1} - q_j^{(t)}|)} \quad (j = 1, \cdots, K)$$

where K is the number of prototypes. Note that $v_j > 0$ $(j = 1, \cdots, K)$ and $\sum_{j=1}^{K} v_j = 1$.

We then introduce two discounting parameters r_1, r_2 to define two K-discrete probability distributions $S(t)$ $\$\mathsf{H}L(t)$ by the following formula:

$$\begin{aligned} S(t) &:= (1 - r_1)S(t-1) + r_1 v^{(t)}, \\ L(t) &:= (1 - r_2)L(t-1) + r_2 S(t). \end{aligned}$$

Here we call $S(t)$ the *short term model* and $L(t)$ the *long term model*, respectively. We initially set $S_i(0) = 1/K$ and $L_i(0) = 1/K$. Note that $S_j(t) > 0$ and $L_j(t) > 0$ $(j = 1, \cdots, K)$, $\sum_{j=1}^{K} S_j(t) = 1$ and $\sum_{j=1}^{K} L_j(t) = 1$.

Here we define a score of $\mathbf{y}_{t+1}$ by

$$S(\mathbf{y}_{t+1}) = \sum_j \left(\sqrt{S_j(t)} - \sqrt{L_j(t)} \right)^2.$$

It appears difficult to give a clear information-theoretic/statistical interpretation to their score calculation based on the long/short term modeling. In contrast to Burge and Shawe-Taylor's algorithm, the kernel version of SmartSifter estimates prototypes using the modified Grabec's algorithm then, as seen in the next section, directly calculates a score for a datum as a statistical difference between the models before and after learning the kernel mixture from the datum, without using long/short term models. Hence SmartSifter is much simpler and gives a clear statistical justification to a score.

2.4 Score Calculation

Below we give a method of calculating a score for a datum. Let $p^{(t)}(\mathbf{x}, \mathbf{y})$ be the joint distribution obtained at time t, i.e., $p^{(t)}(\mathbf{x}, \mathbf{y}) = p^{(t)}(\mathbf{x}) p^{(t)}(\mathbf{y}|\mathbf{x})$.

Given an input $(\mathbf{x}_t, \mathbf{y}_t)$ at time t, we define its *Hellinger score* by

$$S_H(\mathbf{x}_t, \mathbf{y}_t) = \frac{1}{r^2} \sum_{\mathbf{x}} \int \left(\sqrt{p^{(t)}(\mathbf{x}, \mathbf{y})} - \sqrt{p^{(t-1)}(\mathbf{x}, \mathbf{y})} \right)^2 d\mathbf{y},$$

where r is a discounting parameter where we set $r_h = r$. Intuitively, this score measures how large the distribution $p^{(t)}$ has moved from $p^{(t-1)}$ after learning from $(\mathbf{x}_t, \mathbf{y}_t)$.

We also define another score called the *logarithmic loss* as

$$S_L(\mathbf{x}_t, \mathbf{y}_t) = -\log p^{(t-1)}(\mathbf{x}_t) - \log p^{(t-1)}(\mathbf{y}_t|\mathbf{x}_t).$$

From the viewpoint of information theory, the logarithmic loss can be thought of as the codelength required to encode $(\mathbf{x}_t, \mathbf{y}_t)$ under the assumption that a datum is generated according to a probability density $p^{(t-1)}$.

Letting $d_h(p^{(t)}, p^{(t-1)}) \stackrel{\text{def}}{=} \int (\sqrt{p^{(t)}(\mathbf{y}|\mathbf{x})} - \sqrt{p^{(t-1)}}(\mathbf{y}|\mathbf{x}))^2 d\mathbf{y}$, the Hellinger score can be expanded as follows.

$$\begin{aligned} S_H(\mathbf{x}_t, \mathbf{y}_t) &= \frac{1}{r^2}\left(2 - 2\sum_{\mathbf{x}} \sqrt{p^{(t)}(\mathbf{x})p^{(t-1)}(\mathbf{x})} \int \sqrt{p^{(t)}(\mathbf{y}|\mathbf{x})p^{(t-1)}(\mathbf{y}|\mathbf{x})} d\mathbf{y}\right) \\ &= \frac{1}{r^2}\left(2 - 2\sum_{\mathbf{x}} \sqrt{p^{(t)}(\mathbf{x})p^{(t-1)}(\mathbf{x})} + \sum_{\mathbf{x}} \sqrt{p^{(t)}(\mathbf{x})p^{(t-1)}(\mathbf{x})} d_h(p^{(t)}, p^{(t-1)})\right). \end{aligned}$$

Notice here that $d_h(p^{(t)}, p^{(t-1)})$ for finite mixture models is not easy to calculate strictly. We can use the following approximation formula under the condition that $||\theta - \theta'||$ is small.

$$d_h(p(\cdot|\theta), p(\cdot|\theta')) \sim \sum_{i=1}^{k} \left(\sqrt{c_i} - \sqrt{c_i'}\right)^2 + \sum_{i=1}^{k} \frac{c_i + c_i'}{2} d_h(p(\cdot|\mu_i, \Sigma_i), p(\cdot|\mu_i', \Sigma_i')).$$

where

$$\begin{aligned} & d_h(p(\cdot|\mu_i, \Sigma_i), p(\cdot|\mu_i', \Sigma_i')) \qquad (5) \\ &= \int \left(\sqrt{p(\mathbf{y}|\mu_i, \Sigma_i)} - \sqrt{p(\mathbf{y}|\mu_i', \Sigma_i')}\right)^2 d\mathbf{x} \\ &= 2 - \frac{2|(\Sigma_i^{-1} + \Sigma'^{-1}_i)/2|^{-1/2}}{|\Sigma_i|^{1/4}|\Sigma_i'|^{1/4}} \\ &\times \exp\left[(1/2)(\Sigma_i^{-1}\mu_i + \Sigma'^{-1}_i \mu'_i)^T (\Sigma_i^{-1} + \Sigma'^{-1}_i)^{-1} (\Sigma_i^{-1}\mu_i + \Sigma'^{-1}_i \mu'_i)\right] \\ &\times \exp\left[-(1/2)(\mu_i{}^T \Sigma_i^{-1} \mu_i + \mu'_i{}^T \Sigma'^{-1}_i \mu'_i)\right]. \end{aligned}$$

Consider the case where one deals with continuous variables only using a Gaussian mixture model with $k = 1$, i.e., $\frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{y} - \mu)^T \Sigma^{-1} (\mathbf{y} - \mu)\right)$. Let the estimates of μ and Σ from $\mathbf{y}^{t-1}$ be $\mu^{(t-1)}$ and $\Sigma^{(t-1)}$. Then the logarithmic loss for $\mathbf{x}_t$ is calculated as

$$\frac{1}{2}(\mathbf{y}_t - \mu^{(t-1)})^T (\Sigma^{(t-1)})^{-1} (\mathbf{y}_t - \mu^{(t-1)}) + \log(2\pi)^{d/2} |\Sigma^{(t-1)}|^{1/2},$$

which coincides with the Mahalanobis distance between $\mathbf{y}_t$ and $\mu^{(t-1)}$ within a constant multiple factor and an additional constant. In this sense the Mahalanobis distance can be viewed as a special case of the logarithmic loss.

3 Experimental Results

In this section, we experimentally evaluate SmartSifter, the method by Burge and Shawe-Taylor, and a program by Woodruff and Rocke [27]. The last one is an outlier detection program based on Mahalanobis distance and S-estimation[22].

3.1 Simulation

We compared by simulation the parametric version of SmartSifter, the kernel variant of SmartSifter, the method by Burge and Shawe-Taylor, and the program by Woodruff and Rocke, which we abbreviate as SS, SS*, BS and WR, respectively.

We used pseudo random numbers to generate data sets containing a number of outlier groups. We define a probability density for generating data by

$$p(\mathbf{y}) = (1-\epsilon)p_n(\mathbf{y}) + \epsilon p_o(\mathbf{y}),$$

where ϵ is a small positive fraction, p_n is the probability density according to which normal data are generated, and p_o is the probability density according to which outlier data are generated. We refer to the probability distribution defined by p_n as a "bulk distribution." We assume that each datum is three-dimensional. In this experiment, we let the probability density p_n be a Gaussian mixture consisting of two Gaussian components:

$$p_n(\mathbf{y}) = c_1 p(\mathbf{y}|\mu_1, \Lambda_1) + c_2 p(\mathbf{y}|\mu_2, \Lambda_2),$$

where $c_1 + c_2 = 1$ and $c_1, c_2 \geq 0$. We let p_0 be a Gaussian mixture consisting of three Gaussian components:

$$p_o(\mathbf{y}) = c_3 p(\mathbf{y}|\mu_3, \Lambda_3) + c_4 p(\mathbf{y}|\mu_4, \Lambda_4) + c_5 p(\mathbf{y}|\mu_5, \Lambda_5),$$

where $c_3 + c_4 + c_5 = 1$ and $c_3, c_4, c_5 \geq 0$. In this experiment we set $\mu_1 = (0, 0, 0)$, $\mu_2 = (0, 7, 0)$,

$$\Lambda_1 = \begin{pmatrix} 1.2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad \Lambda_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1.2 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$\mu_3 = (3.5, 0, 0)$, $\mu_4 = (3.5, 2, 0)$, $\mu_5 = (9, -3, 0)$, $\Lambda_3 = \Lambda_4 = \Lambda_5 = 0.2I$, $\epsilon = 0.003$, $c_1 = c_2 = 1/2$ and $c_3 = c_4 = c_5 = 1/3$, where I is an identity matrix.

We added to each datum an attribute *label*, whose range is {positive1, positive2, positive3}, where positive$i$ means the datum is generated by the ith component of p_o. We let group$i$ denote the set of the data whose label is positive$i$. We did not use *labels* for score calculation, but rather for evaluation. Figure 4 shows the locations of the outlier groups relative to the bulk distribution over y_1-y_2 space. Outliers in group3 are far from the

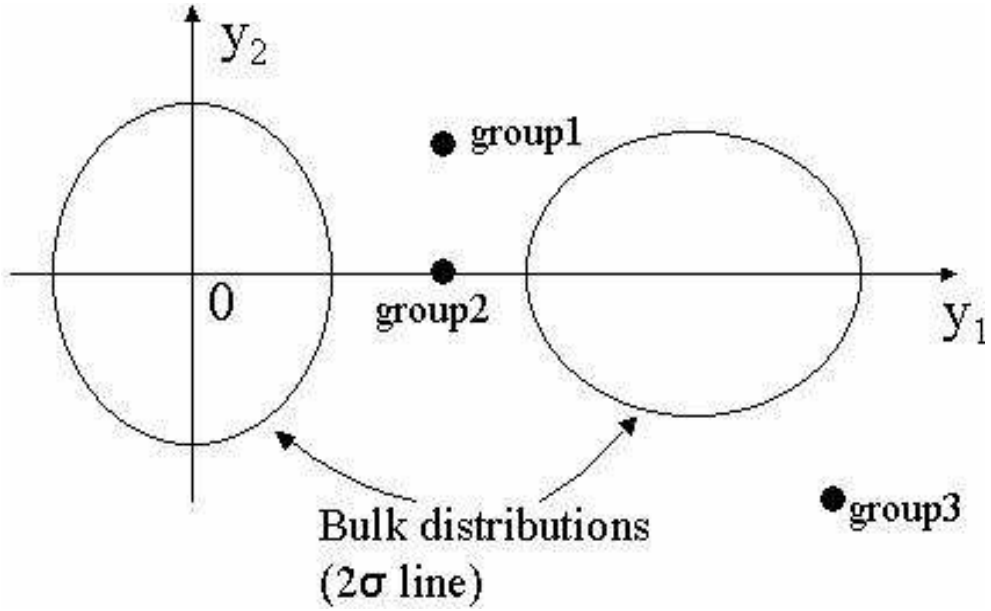

Figure 4: Location of outliers

bulk distribution with respect to the Euclidean distance, hence are expected to be easily detected by WR. On the other hand, those in group2 are located between the means of the two Gaussian components, hence are expected to be difficult for WR to detect.

We generated ten data sets according to the above densities, where each data set consists of 30,000 data, containing $90 = 30,000 \times 0.003$ outliers in average.

We applied the four algorithms to all of the data sets. For SS, we set $r = 0.001$, $k = 2$, and $\alpha = 2.0$, while for SS*, we set $r = 0.001$, $K = 27$, and $\sigma = 1.2$. We used Hellinger score for SS and SS*, For BS, we set $r_1 = 0.01$ and $r_2 = 0.02$. For WR, we set the number of iterations to 40.

Figures 5, 6, 7, and 8 show averaged coverage for the four algorithms. Here *coverage* is

the ratio of the number of detected outliers to that of outliers in the extracted data, where data were extracted in a descendent order of their scores. The horizontal axis represents the ratio of the number of the extracted data to that of a whole data. These graphs show coverage for each of the outlier groups and a whole set of outliers. For example, Figure 5 implies that SS was able to find 80% of outliers (72 outliers) approximately, in the top 10% of highest scored data.

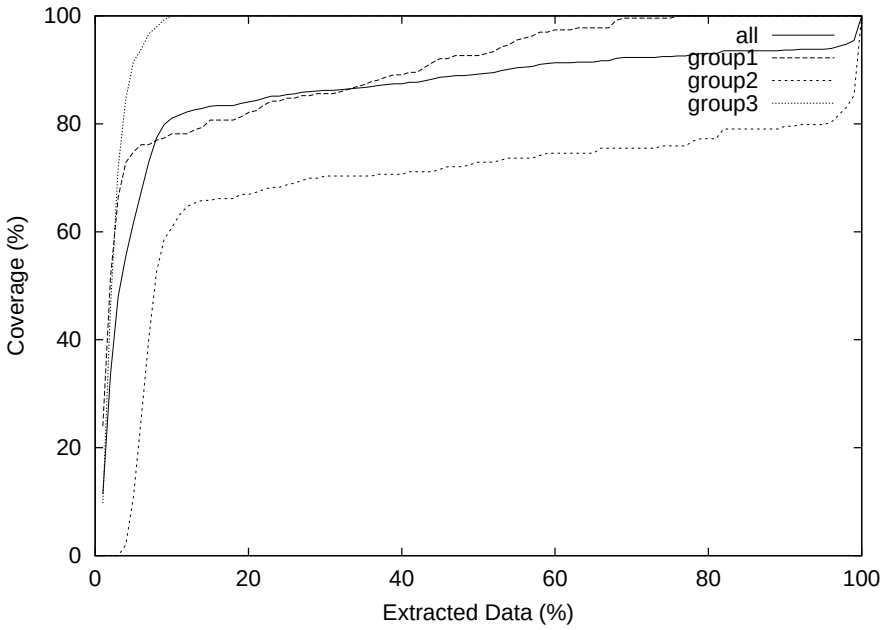

Figure 5: Coverage for SS

We observe from these graphs that SS outperformed other algorithms. Both SS and WR could fairly successfully detect outliers in group3. SS was also able to successfully detect outliers in group2, while SS*, BS and WR were not, as expected.

We investigate how well SS and WR work for the case in which the data distribution is time-dependent. We prepared the model illustrated in Figure 10. In the beginning, the model was the same as the stationary case. Then the second component of the bulk distribution and the model of group3 outliers moved at a constant velocity, with their shape and relative locations preserved. In the final stage, the second component of the bulk distribution located at the position where the group3 outliers used to be in the beginning. Figures 11 and 12 respectively show coverage for SS and WR.

We observe from Figures 11 and 12 that the coverage of group3 for WR in the non-

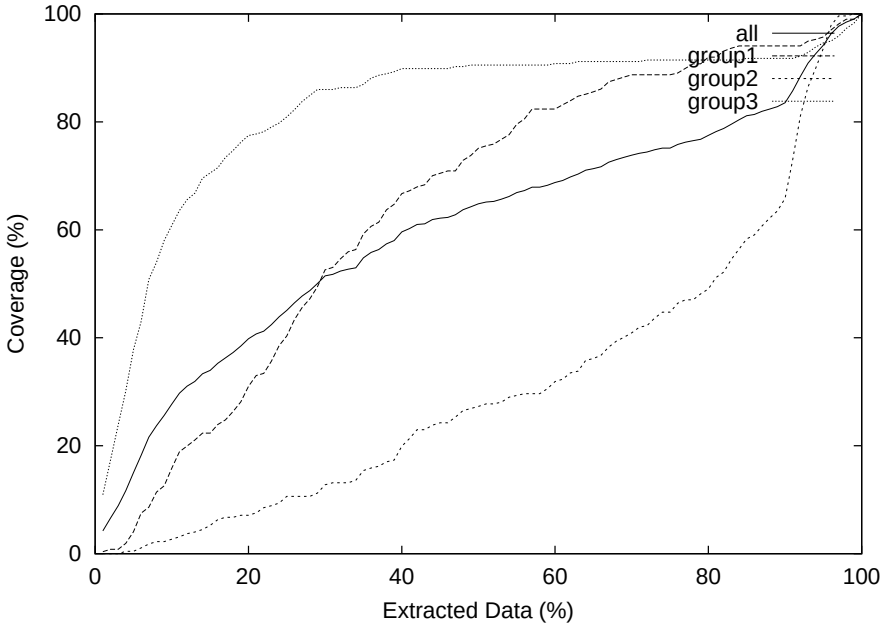

Figure 6: Coverage for SS*

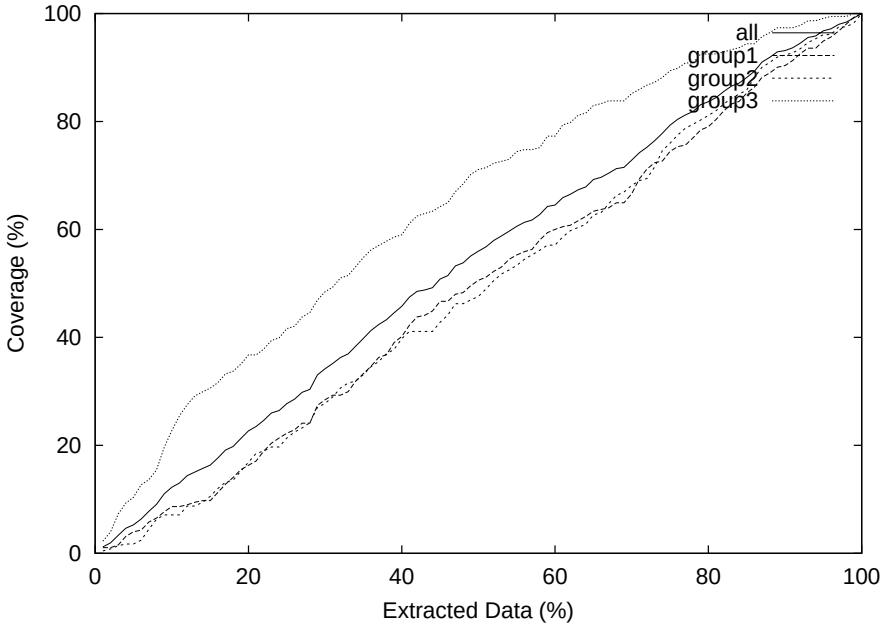

Figure 7: Coverage for BS

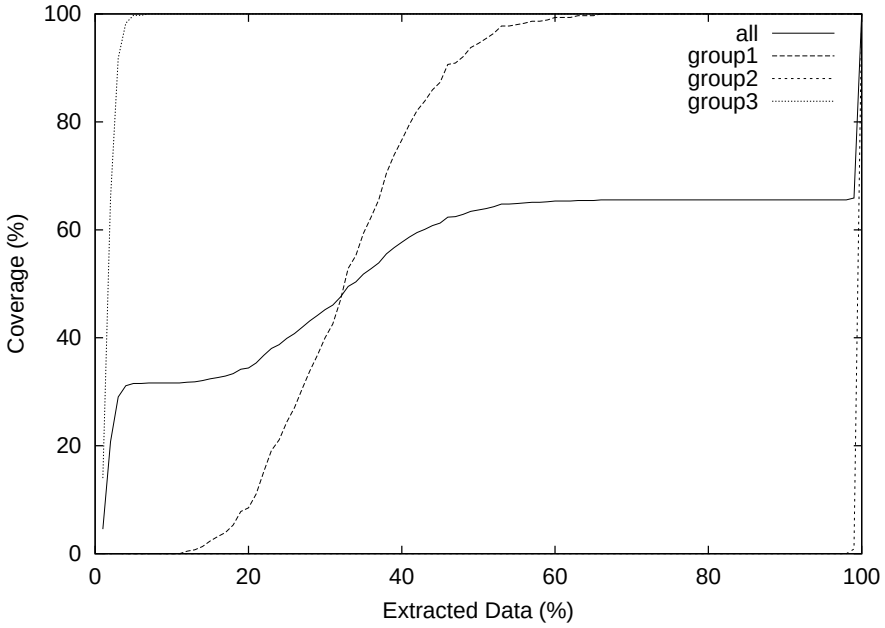

Figure 8: Coverage for WR

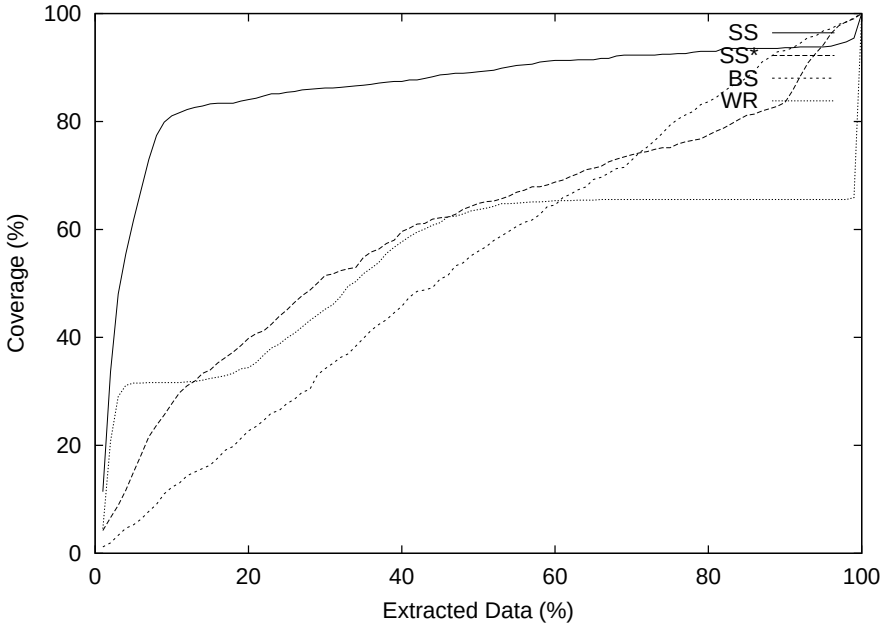

Figure 9: Coverage for the four algorithms

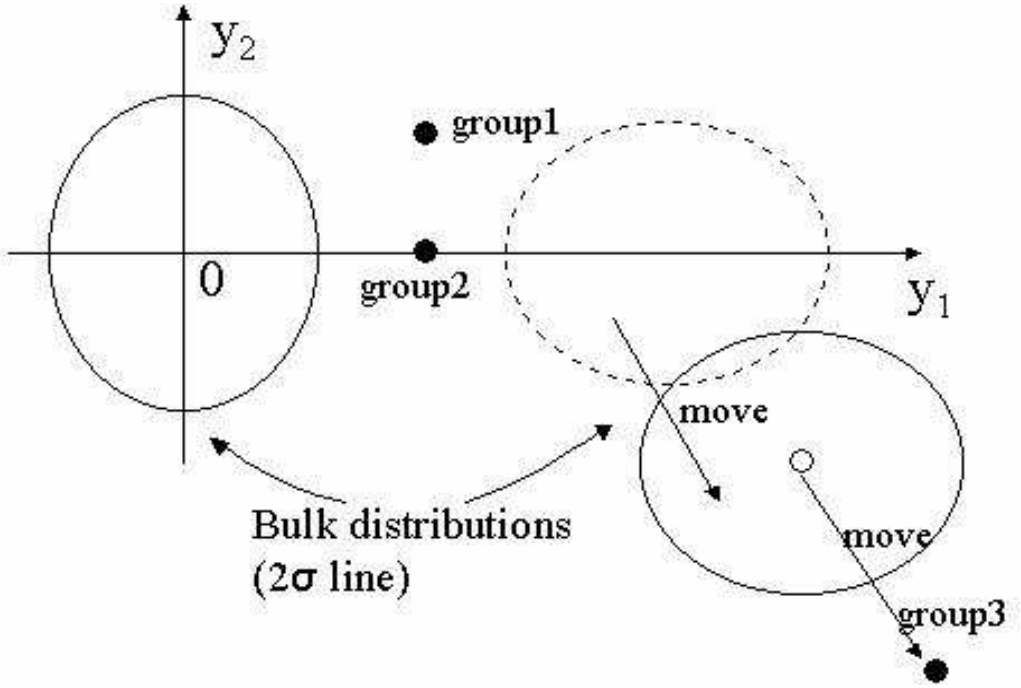

Figure 10: Location of outliers (time-dependent model)

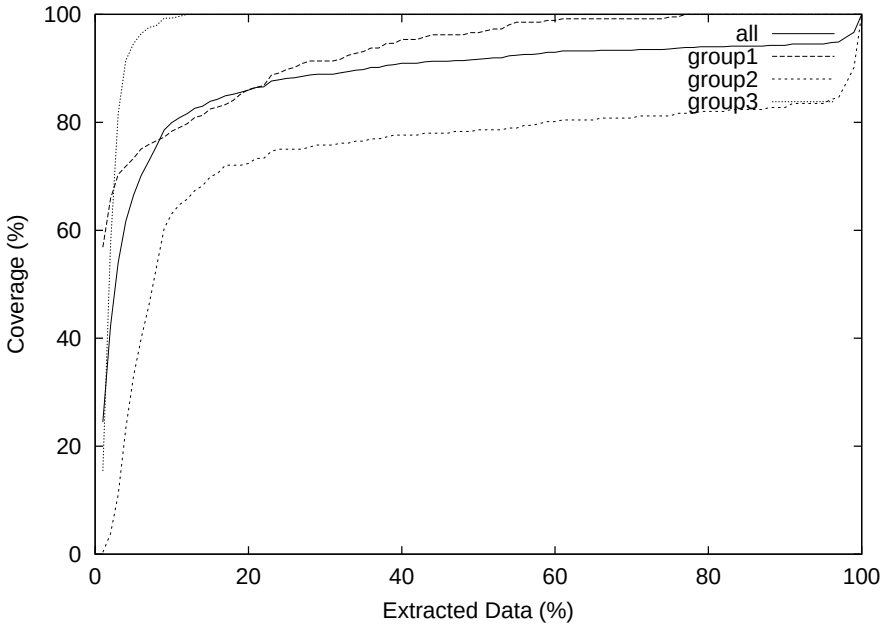

Figure 11: Coverage for SS with time-dependent data

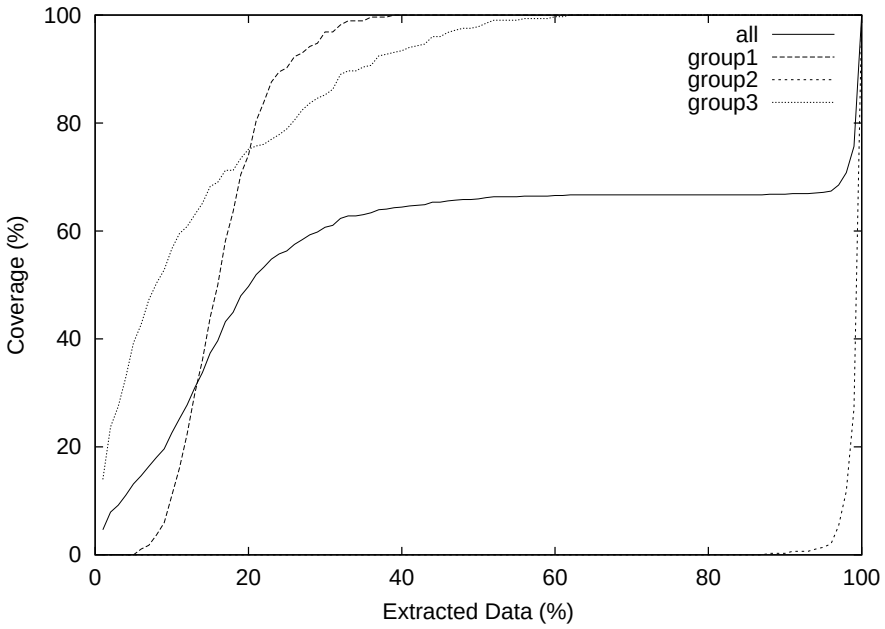

Figure 12: Coverage for WR with time-dependent data

stationary case decreased in comparison with the stationary case, while that for SS didn't change at all. This fact demonstrates that SS's function of discounting learning is significant in dealing with non-stationary sources

Table 1 shows the running time of all the algorithms. We may see that SS runs most efficiently, specifically achieving 1/2000 computation time in comparison with WR for 30,000 dataset.

algorithm	SS	SS*	BS	WR
time (second)	3.74	750	2380	7760

Table 1: Running time for 30,000 data

3.2 Network Intrusion Detection

We applied SmartSifter to the dataset KDD Cup 1999 [26] prepared for network intrusion detection. The purpose of the experiment was to detect as many intrusions as possible in an on-line setting without making use of the labels. Although in KDD Cup 1999 the data labels, each of which is concerned with if it is an intrusion or not, were used in training for supervised intrusion detection, we used them only for the evaluation of SmartSifter. Hence any supervised approach to the dataset is not fairly comparable with SmartSifter.

Each datum in KDD Cup 1999 is specified by 41 attributes (34 continuous and 7 categorical) and a label describing attack type (22 kinds: normal, back, buffer_overflow, ftp_write, warezmaster, etc.) where all labels except "normal" indicate an attack. We used four of the original 41 attributes (service, duration, src_bytes, dst_bytes) because these four were thought of as the most basic attributes. Only 'service' is categorical. The range of service is {http, smtp, finger, domain_u, auth, telnet, ftp, eco_i, ntp_u, ecr_i, other,pop_3, pop_2, ftp_data, ssh, gopher, domain, private, login, imap4, time, shell, IRC, urh_i, X11, urp_i, tftp_u, discard, tim_i, red_i, nntp, uucp, netbios_ssn, daytime, echo}. The number of service kinds is 41, and we classified them into {http, smtp, ftp, ftp_data, others} because each categorical variable belonging to "others" has a low frequency. Since the continuous attribute values were concentrated around 0, we transformed each value into a value far from 0, by $y = \log(x + 0.1)$.

The original dataset contains 4,898,431 data, including 3,925,651 attacks (80.1%). This rate of attacks is too large for statistical outlier detection. Therefore, we produced a sub dataset SF consisting of 703,066 data, including 3,377 attacks (0.48%) by picking up the data whose attribute *logged_in* is positive. Attacks that successfully *logged_in* are called *intrusions*. Note that an outlier is not necessarily an intrusion. We investigate how many intrusions are included in the outliers which we detect using SmartSifter.

We further produced from SF datasets SF10 by random sampling, which consists of 70,000 data (approximately 10% of SF). We generated ten SF10s by making different random samplings. The first 7,000 data were not scored but used only for training because the model would not be well-trained in the early stages.

We generated ten SF10s by making different random samplings. For each of them, we ran both parametric and kernel versions of SmartSifter where the former is denoted by SS while the latter by SS*. The parameters of SS are set to $r = 0.0002, r_h = 0.0003, k = 2$, and $\alpha = 2.0$, while those of SS* are set to $r = 0.0002, r_h = 0.0003, K = 10$, and $\sigma = 0.2$.

Figure 13 shows the averaged coverage for SS and SS* where the Hellinger score was used as a score. This graph shows that SS significantly outperformed SS*, as observed in simulation.

Table 2 shows the number of intrusions SS(*) detected where the Hellinger score was

top ratio	♯ of intrusions included (coverage)		
	SS with SF	SS with SF10	SS* with SF10
1%	610 (18%)	69.8 (21%)	22.0 (6.6%)
3%	2190 (64%)	262 (79%)	53 (16%)
5%	2816 (83%)	270 (81%)	71 (21%)
10%	3305 (98%)	304 (91%)	102 (31%)
total	3373	332 (average)	332 (average)

Table 2: Number of detected intrusions in SF10

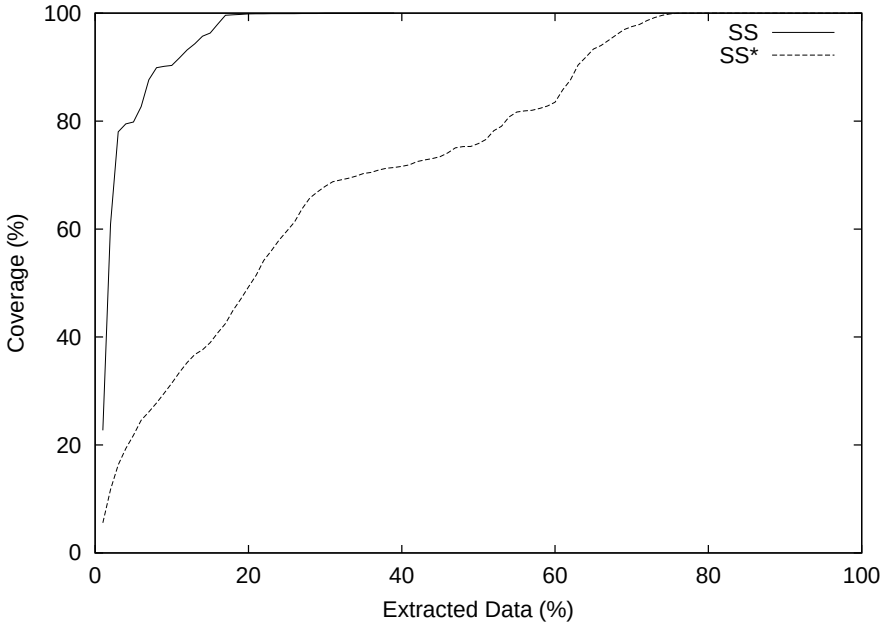

Figure 13: Coverage for SS and SS* with SF10s

used as a score. Table 3 shows the CPU time for KDDCup data. SS is superior to SS* both in accuracy and computation time. This implies that the SDEM algorithm with a Gaussian mixture works better than the SDPU algorithm using a kernel mixture. Remarkably, SS was able to detect 79% intrusions in the top 3% data of highest scores, and 81% intrusions in the top 5% data of highest scores.

algorithm	SS for SF	SS for SF10	SS* for SF10
time (second)	222	23	849

Table 3: Running time for KDDCup data

Finally, we applied SS to the dataset SF, where we used the same attributes as used for SF10. Figure 14 shows the coverage for SS. We observe that the coverage for a larger dataset SF increases more rapidly as the extracted data size increases, than for a smaller dataset SF10.

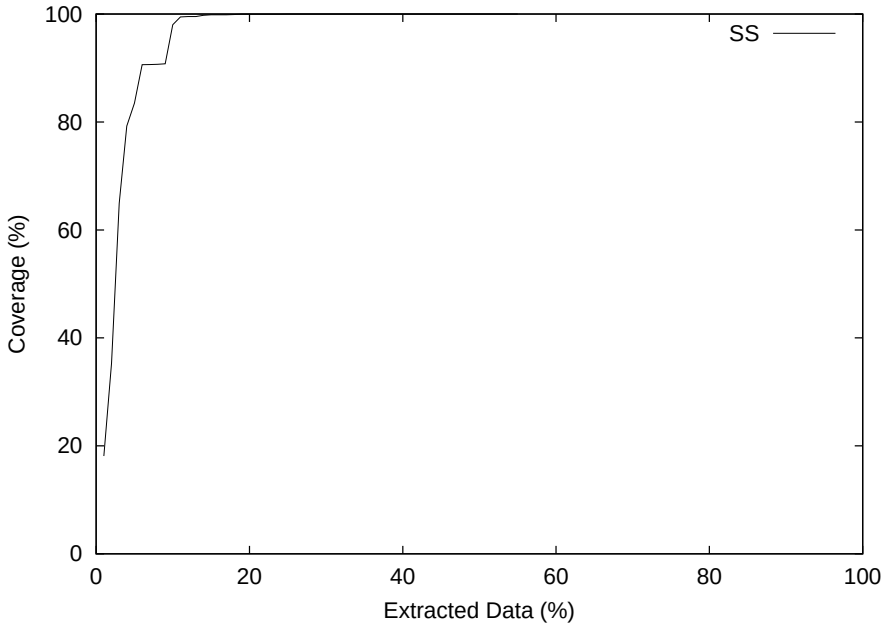

Figure 14: Coverage for SS with SF

3.3 Outliers in Medical Pathology Data

Australia's Health Insurance Commission (HIC) administers the universal health insurance scheme known as Medicare and a variety of other Government payment systems. An

important role performed by the HIC is that of preventing and detecting fraud and inappropriate servicing. The HIC is also moving towards a broader role in improving health outcomes. The universal health insurance scheme has been in operation since 1975. The transactional data for claimed medical expenses since 1975 forms a massive amount of data available for analysis.

CSIRO Australia is working with the HIC on a project exploring the utilization of pathology services. The project is exploring many aspects of the provision of pathology services through approved pathology companies. The longer term aim of the project is to identify variations in practices and provide insights that may lead to improvements in health outcomes. The current initial phase of the project is exploring for interesting and novel features within the data that will lead the way for a detailed study of health outcomes.

The dataset used for this initial phase consists of over 32 million pathology transactions (suitably de-identified to protect privacy) over a 2 year period. Associated with each transaction is information about the type of pathology test being performed (there are some 400 different types of transactionable pathology items), the rendering doctor, the pathology company performing the tests, the doctor who requested the tests to be done, and information relating to the type of doctor and patient (age, gender, location, etc.).

A straight-forward application of SmartSifter to the transactional data lead to the identification of individual transactions that are unusual. However, transaction level data is not particularly suited to the types of analyses performed in data mining. Instead, entity oriented analyses were performed after suitably transforming the dataset in various ways. SmartSifter was used to explore for unusual and rare patterns of behavior associated with individual patients (nearly 4 million), individual doctors (approximately 20 thousand), and pathology providers (approximately 150).

The types of transformations performed include aggregating relevant features of the transactions, which is a laborious and iterative task of identifying different types of aggregates and exploring whether they provide suitable results. SmartSifter was employed in this iterative process as a tool supporting feature selection, giving insights into the chosen collection of features. For this data pre-processing phase of the data mining project,

SmartSifter proved useful in identifying problems in the datasets. That is, initial applications of SmartSifter highlighted records in the datasets that contained errors. In general, their distance scores were quite significantly larger than those of other records. These errors were rectified and a cleaner dataset developed.

For illustrative purposes we demonstrate SmartSifter here to explore for pathology providers (of which there are about 150 in the dataset) which stand out from the rest in some sense. An aggregate dataset was derived from the source transactional dataset by aggregating on the pathology provider. The aggregation constructed 7 variables for each pathology provider, including percentage distributions over each of 5 pathology groups; the number of different patients; etc. SmartSifter identified providers numbered 109, 126, and 114, as having consistently high distance scores (Table 4) across two distance scores.

Pahol	Hellinger		Logarithmic	
Prov	Rank	Score	Rank	Score
109	**1**	**65.6**	**2**	**1.3**
126	**2**	**58.3**	**1**	**1.3**
114	**3**	**57.5**	**3**	**0.6**
112	4	35.5	4	-0.2
75	5	33.4	5	-0.3
50	6	25.4	6	-0.8
79	7	25.0	7	-1.2
123	8	23.7	9	-1.3
129	9	23.3	11	-1.4
51	10	21.1	8	-1.3
104	11	21.0	13	-1.6

Table 4: Summary of rare cases for pathology providers

Empirically SS has pinpointed pathology providers as significant rare-cases. This outcome can be confirmed from the sample of provider data listed in Table 5. Columns A to E record the proportion of pathology tests performed by the provider in each of 5 different categories. These categories are referred to as Chemical, Microbiology, Immunology, Tissue Pathology, and Cytology. SS has identified, in this case, two pathology providers specializing in Tissue Pathology (109 and 114) and another that does no Microbiology nor Tissue Pathology (126). The "normal" situation, from this dataset, indicates that pathology providers generally perform a spread of tests.

Of course, using such a small dataset does not exhibit the full power of SS in finding

Prov	A	B	C	D	E	F	G
107	0.64	0.23	0.03	0.08	0.03	0.27	0.02
108	0.65	0.20	0.10	0.05	0.01	0.37	0.03
109	**0**	**0**	**0**	**1**	**0**	**1**	**0.43**
110	0	0.00	0	0.57	0.43	0.89	0.10
111	0.30	0.13	0.02	0.05	0.50	0.32	0.01
112	0	0	0	0	1	0.83	0.43
113	0.34	0.14	0.02	0.02	0.47	0.38	0.02
114	**0**	**0**	**0**	**1**	**0**	**0.96**	**0.30**
115	0.36	0.09	0	0	0.55	0.45	0.45
116	0.41	0.06	0.05	0.01	0.47	0.47	0.32
117	0.29	0.12	0	0	0.58	0.39	0.35
118	0.52	0.06	0.02	0	0.39	0.41	0.34
119	0	0	0	0.5	0.5	0.5	0.5
120	0.34	0.14	0.03	0.02	0.46	0.34	0.02
125	0.55	0.23	0.02	0.17	0.03	0.56	0.07
126	**0.33**	**0**	**0.33**	**0**	**0.33**	**1**	**1**
127	0.33	0.18	0.02	0.01	0.46	0.34	0.01
128	0.29	0.14	0.02	0.02	0.52	0.39	0.03

Table 5: Sample of pathology provider data. Column names and actual provide numbers have been de-identified.

rare cases. Indeed, manually reviewing all 150 records in the pathology provider dataset is a feasible task and would identify the same observations made by SS. Yet this simple example confirms SmartSifter's ability to identify patterns in the data that a human expert finds interesting. The ongoing application of SmartSifter to the 20,000 doctor dataset and the 4,000,000 patient dataset is providing similar insights into the data. Rare cases are being efficiently highlighted and the HIC have identified situations where this will have impact.

This project with the Health Insurance Commission is still in progress and is employing a wide variety of techniques drawn from data mining and statistics in general. The use of SmartSifter has provided and continues to provide useful insights into the data. Many significant observations have been highlighted by SmartSifter, identifying candidates for further investigation.

4 Conclusion

This paper has proposed SmartSifter as a program for on-line unsupervised outlier detection. We gave a statistical theory for SmartSifter and demonstrated its effectiveness in

terms of accuracy and computation time through the experiments using: simulated data; network intrusion detection data from the 1999 KDD Cup; and rare event detection for the health insurance pathology data provided by Australian Health Insurance Commission.

Through SmartSifter we offer a general framework for on-line unsupervised outlier detection, which is expected to be further applied to a variety of data mining tasks other than fraud-detection, such as event detection [11], topic detection [1], etc.

The following technical issues remain open for future investigation:
1) Choosing the optimal number of components in the mixture models; 2) automatic clustering of categorical attributes; 3) extending SmartSifter using time series modeling; and 4) exploration of the robustneSmartSifter of the algorithm to parameter tuning.

References

[1] J. Allan, J. Carbonell, G. Doddington, J. Yamron, and Y. Yang, Topic detection and tracking pilot study: Final report, in *Proc. of the DARPA Broadcast News Transcription and Understanding Workshop*, pp:194–218, 1998.

[2] V. Barnett and T. Lewis, *Outliers in Statistical Data*, John Wiley & Sons, 1994.

[3] F. Bonchi, F. Giannotti, G. Mainetto, and D. Pedeschi, A classification-based methodology for planning audit strategies in fraud detection, in *Proc. of KDD-99*, pp:175–184, 1999.

[4] P. Burge and J. Shawe-Taylor, Detecting cellular fraud using adaptive prototypes, in *Proc. of AI Approaches to Fraud Detection and Risk Management*, pp:9–13, 1997.

[5] P. Chan and S. Stolfo, Toward scalable learning with non-uniform class and cost-distributions: A case study in credit card fraud detection, in *Proc. of KDD-98*, AAAI-Press, pp:164–168 (1998).

[6] T. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley-International, 1991.

[7] A. P. Dempster, N. M. Laird, and D. B. Ribin, Maximum likelihood from incomplete data via the EM algorithm, *Journal of the Royal Statistical Society*, B, 39(1), pp:1–38, 1977.

[8] T. Fawcett and F. Provost, Combining data mining and machine learning for effective fraud detection, in *Proc. of AI Approaches to Fraud Detection and Risk Management*, pp:14–19, 1997.

[9] T. Fawcett and F. Provost, Activity monitoring: noticing interesting changes in behavior, in *Proc. of KDD-99*, pp:53–62, 1999.

[10] I. Grabec, Self-organization of Neurons described by the maximum-entropy principle, *Biological Cybernetics,* vol. 63, pp:403–409, 1990.

[11] V. Guralnik and J. Srivastava, Event detection from time series data, in *Proc. KDD-99*, pp:33–42, 1999.

[12] D. M. Hawkins, Identification of Outliers. Chapman and Hall, London, 1980.

[13] E. M. Knorr and R. T. Ng, Algorithms for mining distance-based outliers in large datasets, in *Proc. of the 24th VLDB Conference*, pp:392–403, 1998.

[14] E. M. Knorr and R. T. Ng, Finding intensional knowledge of distance-based outliers, in *Proc. of the 25th VLDB Conference*, pp:211–222, 1999.

[15] R. E. Krichevskii and V. K. Trofimov, The performance of universal coding, *IEEE Trans. Inform. Theory*, IT-27:2, pp:199–207, 1981.

[16] T. Lane and C. Brodley, Approaches to on-line learning and concept drift for user identification in computer security, in *Proc. of KDD-98*, AAAI Press, pp:66–72, 1998.

[17] W. Lee, S. J. Stolfo, and K. W. Mok, Mining audit data to build intrusion detection models, in *Proc. of KDD-98*, 1998.

[18] W. Lee, S. J. Stolfo, and K. W. Mok, Mining in a data-flow environment: experience in network intrusion detection, in *Proc. of KDD-99*, pp:114–124, 1999.

[19] G. McLachlan and D. Peel: *Finite Mixture Models*, Wiley Series in Probability and Statistics, John Wiley and Sons, (2000).

[20] Y. Moreau and J. Vandewalle, Detection of mobile phone fraud using supervised neural networks: a first prototype, Available via: ftp://ftp.esat.kuleuven.ac.jp/pub/SISTA/moreau/reports/icann97_TR97-44.ps.

[21] R. M. Neal and G. E. Hinton, A view of the EM algorithm that justifies incremental, sparse, and other variants, ftp://ftp.cs.toronto.edu/pub/radford/www/publications.html, 1993.

[22] D. M. Rocke, Robustness properties of S-estimators of multivariate location and shape in high dimension, *the Annals of Statistics*, Vol. 24, No. 3, pp. 1327-1345, 1996.

[23] S. Rosset, U. Murad, E. Neumann, Y. Idan, and G. Pinkas, Discovery of fraud rules for telecommunications-challenges and solutions, in *Proc. of KDD-99*, pp:409–413, 1999.

[24] G. J. Williams and Z. Huang, Mining the Knowledge Mine: The Hot Spots Methodology for Mining Large Real World Databases, in *Advanced Topics in Artificial Intelligence* Lecture Notes in Artificial Intelligence, Volume 1342, pp:340–348 Springer-Verlag, 1997.

[25] K. Yamanishi, J. Takeuchi, G. Williams, and P. Milne, On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms, in *Proc. of KDD2000*, ACM Press, pp:250–254, 2000.

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

[27] http://lib.stat.cmu.edu/jasasoftware/

[28] http://www.hnc.com