LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-TR-213599

Design and Implementation of an Anomaly Detector

A. Bagherjeiran, E. Cantu-Paz, C. Kamath

July 14, 2005

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.

Design and Implementation of an Anomaly Detector

Abraham Bagherjeiran
Lawrence Livermore National Laboratory
bagherjeiran2@llnl.gov

Erick Cantú-Paz
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
cantupaz@llnl.gov

Chandrika Kamath
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
kamath2@llnl.gov

UCRL-TR-213599

15th July 2005

Abstract

This paper describes the design and implementation of a general-purpose anomaly detector for streaming data. Based on a survey of similar work from the literature, a basic anomaly detector builds a model on normal data, compares this model to incoming data, and uses a threshold to determine when the incoming data represent an anomaly. Models compactly represent the data but still allow for effective comparison. Comparison methods determine the distance between two models of data or the distance between a model and a point. Threshold selection is a largely neglected problem in the literature, but the current implementation includes two methods to estimate thresholds from normal data. With these components, a user can construct a variety of anomaly detection schemes. The implementation contains several methods from the literature. Three separate experiments tested the performance of the components on two well-known and one completely artificial dataset. The results indicate that the implementation works and can reproduce results from previous experiments.

1 Introduction

In many circumstances, data arrive at a rate that greatly exceeds available processing capacity. For example, raw data from sensors or simulations may indicate the current status of a system, but transformation from these raw data to useful information is difficult because of both the quantity and length of the data. In the streaming data model [11], the data consist of points that arrive in sequential order and describe the instantaneous state of the system. Although the order is sequential, the stream may be infinitely long. In these cases, it is impractical to wait for the end of the stream to begin processing; thus, research looks to online, anytime algorithms to cope with this information tsunami.

Typical anomaly detectors detect anomalies as data points that significantly deviate from a model of normal data rather than tackle the larger data mining problem. This involves three distinct tasks: model the normal and incoming data, compare the models, and ascertain the meaning of significance. A model of the data should consume less memory than the original but preserve important characteristics of the data for subsequent comparisons. The comparison technique must address the philosophical question of the between two different interpretations of data. From both the normal data and user intuition, the user must then determine the distance beyond which two models are significantly different.

This paper describes initial work to address these three problems in a general purpose anomaly detector. The anomaly detector has no particular target application, but it is sufficiently general to be useful for several application areas. The application areas should, however, share a set of common assumptions on the data. First, the data arrive as a stream of tuples of the same size, but each tuple is completely available when it arrives–tuples are atomic. Second, tuples arrive in sequential order. Third, tuples or data windows have labels so that the anomaly detector can determine whether or not data is normal during training. This last assumption is really one of convenience because none of the components has explicit knowledge of the label, but labels are useful in the evaluation of detector performance. All three of these assumptions are common in the anomaly detection literature [24, 15, 22].

The anomaly detector consists of components that build models, compare models, and select thresholds. The models are well-known dimension-reduction and statistical models from the literature. Model comparers determine the difference between two models or between a model and a point in the data again with well-known comparison methods. Threshold selectors attempt to learn a threshold from training data and the normal model. The focus on separable components and object-oriented design facilitates code reuse. Similarly, the focus on learning thresholds provides reasonable performance without an excessive amount of effort to a user who is unfamiliar with the models or comparison methods. A more dedicated or bored user can customize or re-implement all aspects of the anomaly detector.

This paper discusses the design and implementation of an anomaly detector. Section 3 describes the design of the components of the detector. Section 4 describes its application to and evaluation on datasets in the machine learning, network intrusion detection, and stochastic process domains.

2 Related Work

Anomaly detectors are not a new invention, they have extensive use in process control and particularly in intrusion detection systems. This section categorizes previous anomaly detector work by the model and application. Detectors that use matrix-based models perform operations such as Principal Component Analysis or Singular Value Decomposition on the data and use the result to detect anomalies. Other detectors cluster the data and detect outliers as points that are sufficiently far from the clusters. This review is not comprehensive, but it lends support to the concentration of the current project on models, comparers, and thresholds.

Matrix-based anomaly detectors focus on how to compare points to models and models to other models. Several of these detectors use Principal Component Analysis (PCA) described in 3.1.2, and its typical use in modeling is to represent the sources of greatest variation in the data. Several authors [14, 24] argue that this reliance on the source of most of the variation makes the model insensitive to anomalies. Their solutions utilize the least significant sources of variation which are more sensitive to anomalies, and their experimental results yield better performance. Other uses [13, 22, 25] of PCA in anomaly detection focus on its ability to reduce the dimensionality of the incoming data. In one paper, the issue of streaming data arises and the solution of the authors is to record past reduced-dimension data and perform queries with a comparison method that directly compares PCA reductions of data [25]. The current implementation includes this approach and further describes it in Section 3.2.1. In addition to PCA, the Singular Value Decomposition (SVD) also has use in anomaly detection such as the visual detection of hand gestures under water [23].

Other models such as clustering have found some use in anomaly detection. Shah performs fuzzy clustering on intrusion detection data and uses PCA to reduce the dimensionality of the data prior to clustering [22]. In that application, a point is an anomaly if it is further away from the clusters than some predefined threshold. In a similar approach [26], a cluster analysis on DARPA 1999 data shows that many attacks connections are indeed far away from clusters of known normal data. This conclusion lends credence to the use of clustering as a model of normal data for anomaly detection in this dataset.

The main conclusion from this review of related work is that anomaly detection methods in a variety of application areas follow a basic pattern. First, data arrive at the detector. After this, the detector either recalls or creates a model of the normal data, and a comparison then occurs between the normal model and a model of the incoming data or a single point within the data. The result of this comparison, given some threshold, determines whether or not the incoming data are anomalous. The question of the best threshold is one of heuristics or hand tuning, and it is only lightly regarded in the literature. In only a few papers did the authors attempt to systematically determine the threshold. Most of these techniques control the *a priori* false alarm rate [15, 18, 24].

3 Design

This section presents an overview of the functionality of the constituent components of the anomaly detector that build models, compare models, and select thresholds. Each of the components is from the literature and makes slightly different assumptions on the data. The discus-

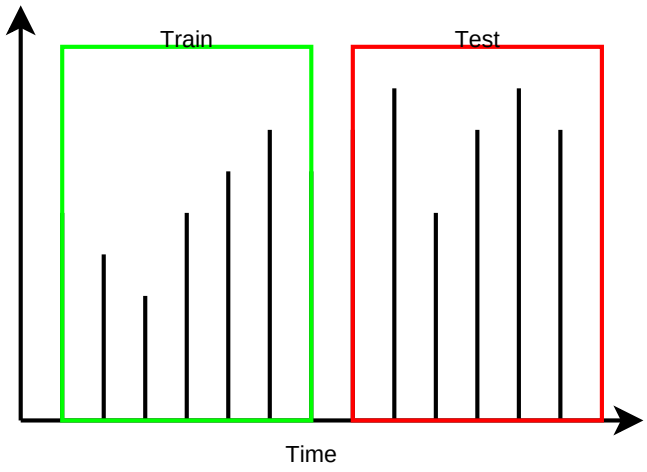

Figure 1: Data flow in the anomaly detector.

sion uses a consistent notation for all descriptions and indicates the assumptions that prevail in the anomaly detector.

Most of the model building components assume that a matrix of the data is available. Data are in a matrix whose rows correspond to the individual points and whose columns correspond to the features or attributes. For a matrix X of dimension $n \times m$, X has n points each of which has m features and is called x. If a transformation of the original data onto a different basis occurs, the resultant matrix is X' of which a point is x'. When the description is more understandable for column vectors rather than row vectors, the matrix $\mathbf{X}$ is in **boldface** type and has points $\mathbf{x}$. In these situations, the following identities hold: $\mathbf{X} = X^T$ and $X = \mathbf{X}^T$.

The anomaly detector builds and compares models on streaming data. In this data model, data arrive in temporal order as Figure 1 shows. The detector divides a large chunk of data into two non-overlapping sliding or fixed-length time windows. Components that build models may or may not support sliding windows. Incremental model builders support sliding windows with a circular queue of points in which newer points overwrite older ones. Non-incremental or batch model builders only support building models on fixed-length windows or chunks. A user can, of course always use an incremental model builder as a batch model builder. To compare th e models, the detector assumes that the first time window an its model represent normal data. If the second window is of length one, the model is the point itself. Otherwise, the model is one of the models from Section 3.1 and is usually of the same variety as the first one. Given the models, the detector uses one of the comparison methods in Section 3.2 to compare them. The design of the model comparers is sufficiently general to compare dissimilar models that similarly represent the data according to a set of common assumptions.

Figures 2 and 3 show the hierarchical relationships among the components. A user can combine the components to create a variety of anomaly detectors. The implementation includes several example general-purpose detectors. The experiments utilize these detectors to demonstrate the variety of combinations.

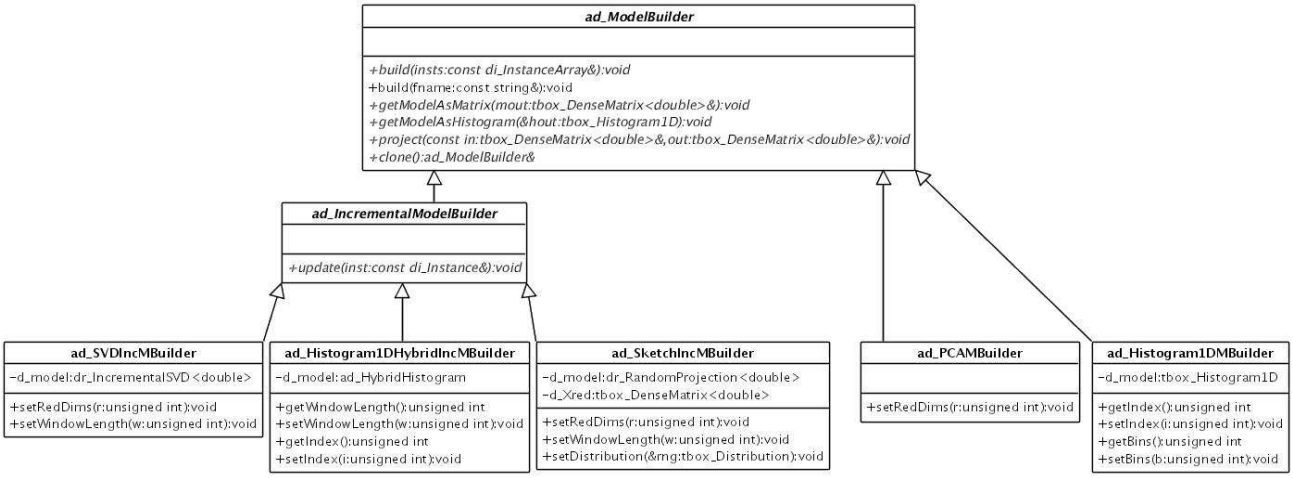

Figure 2: The model builder hierarchy.

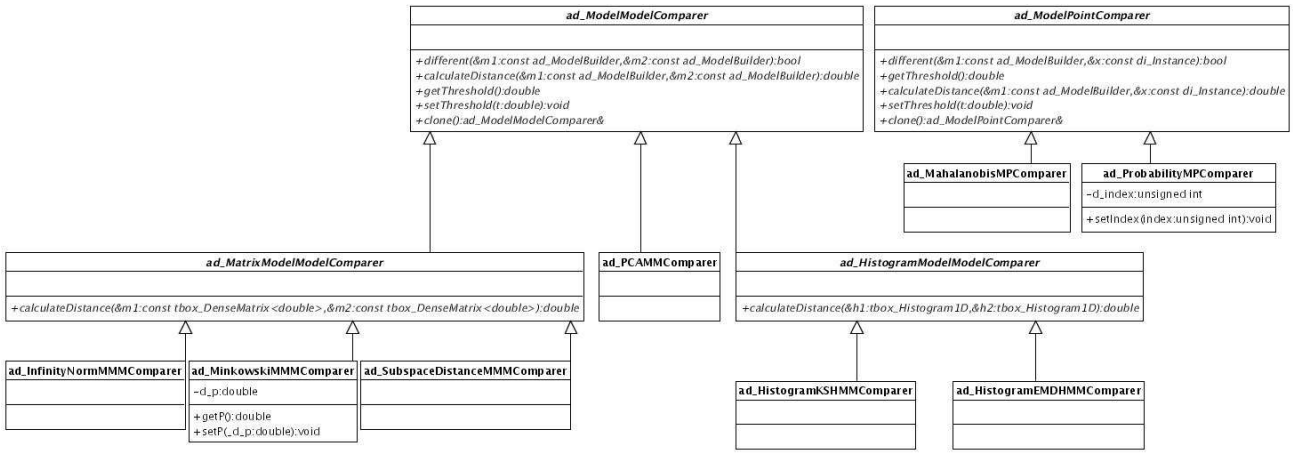

Figure 3: The model comparer hierarchy. Matrix-based model-to-model comparers extract a matrix representation of the models prior to comparison just as histogram-based model-to-model comparers extract a histogram representation of the models.

3.1 Building Models

3.1.1 Singular Value Decomposition

A Singular Value Decomposition (SVD) of a matrix $\mathbf{X}$ yields three matrices $\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}$ such that $\mathbf{X} = \mathbf{USV}^T$. The preceding equation is the traditional representation of the relation; equivalently, one can also express the relation as $X = \mathbf{X}^T = \mathbf{VSU}^T$. The matrix $\mathbf{S}$ is a diagonal matrix whose entries are in decreasing order and are called the singular values. The columns of matrices $\mathbf{U}$ and $\mathbf{V}$ are the left and right singular vectors of $\mathbf{X}$. The columns of $\mathbf{U}$ define an orthonormal subspace onto which one can transform original data. The transformation of a point x into the subspace of the data is:

$$x' = x\mathbf{U}$$

One can obtain the transformation of the original matrix as:

$$X' = \mathbf{VS}$$

Since one can reconstruct the original data from the singular values and vectors, the transformation of the input data does not actually require the presence of the original data in the multiplication.

One can use the SVD of a matrix to create a reduced-dimension representation of the matrix. If the singular values are sorted in decreasing order of the diagonal entries, one can use the k largest singular values and the accompanying singular vectors to approximate the original data. With utilization of this smaller diagonal matrix, the transformation

$$X' = \mathbf{VS}_k$$

is a reduced-dimension model of the data where $\mathbf{S}_k$ contains the k largest singular values and $\mathbf{V}$ contains the corresponding singular vectors. With respect to the mean-squared error of the reduced representation in comparison with the original data, the reduced representation is optimal [5].

The current implementation employs an algorithm [5] that incrementally constructs an SVD in near-linear time with respect to the number of points. The incremental SVD algorithm starts with an SVD of a small set of points and augments it as new points arrive. With the continual removal of the smallest singular value and accompanying singular vectors after each iteration, the incremental algorithm maintains a reduced-dimension representation of the original data storing only the matrices of the decomposition. This leads to significant time and memory savings for incremental data operations such as the manipulation of a single point in the original matrix and the addition of a new point to the matrix. These properties make it ideal for the incremental construction of models from data.

3.1.2 Principal Component Analysis

Principal Component Analysis [7] is a well-known dimension reduction technique with well-understood statistical properties. The model is the projection of the data onto a few of the eigenvectors of the sample covariance matrix of the data. The standard algorithm requires a

chunk of data in matrix form X, computes the scatter matrix $S = (n-1)\Sigma$ where Σ is the sample covariance matrix and the eigenvectors of the scatter matrix U. The projection of the data is:

$$X' = (X - \bar{X})^T U_k$$

where $\bar{X}$ is the mean of the original data, U_k contains the k largest eigenvectors of S. The interpretation of X' is that the points represent the original data with respect to the direction of the most variation of the data. Since the eigenvectors describe progressively less of the variance of the data, the use of fewer than the original number of dimensions yields a reduction in space without a simultaneous large decrease in the accuracy of the approximation with respect to the original data.

3.1.3 Sketch

As one might infer from the name of the model, a sketch [9] is intended to be a reduced-dimension, easy to compute, but not extremely accurate representation of the data. At its core, a sketch is the result of a random projection of the data; it is a transformation from the original data as a matrix onto a set of random vectors. The current implementation draws the elements of the vectors from a distribution that returns -1 and 1 with equal probability, but the user can substitute another distribution. With appropriate scaling, the transformation from the normal data onto the random vectors is reversible, so that one can obtain an approximation of the original data. The benefits of using a sketch are that distance between points is preserved in the transformed space [12] and that only a few random vectors are necessary in practice to achieve this preservation.

One can build a sketch on a chunk of data or on individual points since the random vectors are completely unrelated to the data. Given a matrix X with dimension $n \times m$, the algorithm generates a random matrix R with dimension $m \times k$ where k is the number of reduced dimensions in the result. The entries of R are from the previously mentioned distribution. The transformation is $X' = \frac{1}{\sqrt{k}} XR$ [2], and the constant scaling factor $\frac{1}{\sqrt{k}}$ is necessary to preserve the distances. Other scaling factors are, however, noted in the literature [3].

3.1.4 Histogram

A histogram [28] is an approximation of the probability distribution of a scalar random variable. The histogram model is non-incremental in that one typically builds a histogram on a chunk of data and then rebuilds it on a different chunk of data. The histogram in the current implementation divides the range of values of a random variable into a predetermined number of equal-width partitions. Each of these partitions counts the number of occurrences of the value of the random variable within the partition. The user can configure the number of partitions and the index into the data that the histogram models.

To build a histogram given a chunk of data, one must determine the maximum and minimum values of the data and then divide this interval into a predetermined number of partitions with initial counts of zero. For each value in the data, one should find the partition that contains the value and increment its count. After this, the probability of a value is approximately the ratio of the frequency of the value to the total number of values.

3.1.5 Hybrid Histogram

A hybrid histogram [19] is an extension to the concept of a histogram to a fixed-length sliding time window of data rather than the entire stream or a chunk. The hybrid histogram adapts storage based on the temporal and spatial density of the incoming data and utilizes an auxiliary data structure called the exponential histogram.

Exponential histograms [6] count binary digits in a data stream by counting the number of ones that occur in smaller time windows of exponentially increasing size. As a stream of ones and zeros flows into the exponential histogram, the algorithm allocates a new bucket to each incoming one with a time stamp. When three of these buckets have the same value, they are merged to form a new bucket with the time stamp of the most recent addition. Eventually, some of the buckets will have timestamps that are older than the time limit in which case the algorithm deallocates the buckets. The result of this bucket maintenance is a provably close approximation to the true number of ones that occur in a binary data stream within a given time limit.

Analogous to the traditional histogram, the hybrid histogram maintains counts of data values. In contrast to the traditional histogram, the hybrid histogram creates adaptive partitions in both spatial and temporal intervals of the random variable. These adaptive partitions are exponential histograms constrained to specific intervals of the value of the random variable and time regions. Adaptive partitions whose beginning time is zero are active meaning that the algorithm will increment the count whenever a value arrives. Partitions whose beginning time is not zero are inactive meaning that the algorithm will never increment the count. Eventually, the all the inactive partitions will disappear when their buckets expire. Adaptation of the partitions arises when the count of an active partition significantly deviates from the average of all partitions with data. When the count greatly exceeds the average, the partition may inaccurately represent the true count of the data. The algorithm will split these partitions to form two new, empty partitions each of which occupies half of the original interval. The new partitions become active and the old partition becomes inactive. When the count on pairs of spatially adjacent active partitions is significantly less than the average, the algorithm will merge the pair to save space. The new, active merged partition has the width of the sum of its constituent intervals, and the old partitions become inactive. The number of partitions is proportional to the number of input data items during the time window and is typically less than this number because of partition merging.

To determine the frequency of a particular value of the random variable, the algorithm accumulates the count of each partition with respect to the proportion of the interval it occupies. In the example histogram of Figure 4, the active partitions are along the horizontal axis. To find the frequency of a point x, one first locates the active partition that contains it. One plots an imaginary rectangle along the bounds of the partition and along the temporal axis into the past. The rectangle will partially intersect existing partitions, and the proportion of the intersected area with respect to the total area of the partition is that partition's contribution of its frequency to the total frequency. More formally, given value x that is inside an active partition p, the frequency of x is:

$$freq(x) \quad = \quad \sum_{\{p_i | p_i \cap p \neq \emptyset\}} \frac{|p_i \cap p|}{|p_i|} freq(p_i)$$

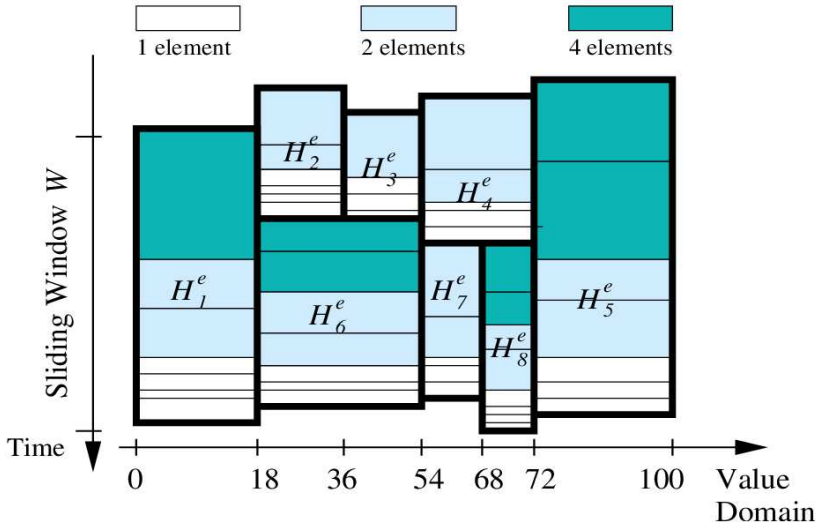

Figure 4: A hybrid histogram from [19].

3.2 Comparing Models

3.2.1 PCA Comparer

The PCA Comparer [25] compares two reduced-dimension representations of data chunks after preforming Principal Component Analysis on them. The data models must both be of size $n \times k$. The distance is defined by the following formula:

$$D_{PCA} = \frac{tr\left(X_1'^T X_2' X_2'^T X_1'\right)}{k}$$

where X_1' is the reduced-dimension model of the normal data, X_2' is the model of the incoming data, and k is the number of reduced dimensions in the model.

3.2.2 L_∞ Comparer

The infinity norm comparer computes the infinity norm of the distance [27] between two matrix representations of data of size $m \times n$. The matrices can, of course, be the original data or any two matrices of the same size. The formula is:

$$d\left(X_1', X_2'\right) = \max_i^m \sum_{j=1}^{n} \left|X'_{1_{ij}} - X'_{2_{ij}}\right|$$

where X'_{ij} is the entry at the ith row and jth column in the model matrix.

3.2.3 Minkowski Comparer

The Minkowski comparer computes the Minkowski distance [7] between two vectors extended to matrices of size $m \times n$. The formula is:

$$d\left(X'_1, X'_2\right) = \sqrt[p]{\sum_{i=1}^{m}\sum_{j=1}^{n}\left|X'_{1_{i,j}} - X'_{2_{i,j}}\right|^p}$$

where p is a parameter that is greater than or equal to one. This distance has other names. For $p = 1$, it is the Manhattan distance and for $p = 2$ it is the Euclidean distance.

3.2.4 Subspace Distance Comparer

This comparer computes the distance between two subspaces of the real-valued space of the models [10]. The formula is:

$$d\left(\mathbf{X}'_1, \mathbf{X}'_2\right) = \max_i^n \min_j^m \left\|\mathbf{X}'_{1_{:i}} - \mathbf{X}'_{2_{:j}}\right\|$$

where i indexes the columns of X'_1 and j indexes the columns of X'_2.

3.2.5 Kolmogorov-Smirnov Comparer

The Kolmogorov-Smirnov comparer Kolmogorov-Smirnov two-tailed goodness-of-fit [17] test on cumulative distributions. The comparer assumes that the models are histograms. To compute the distance, the algorithm converts each of the histograms into a cumulative distribution function which is simply the sum of the frequencies in each of the bins in the histogram. The maximum absolute value of the distance at each bin between the distribution functions D is the distance of importance to the algorithm. It must satisfy the following formula to be statistically significant:

$$D > \sqrt{-\frac{1}{2}\left(\frac{1}{m_1} + \frac{1}{m_2}\right)\ln\frac{(1-\alpha)}{2}}$$

where m_1 is the number of bins in the first histogram and m_2 is the number of bins in the second histogram. The parameter α is a significance parameter which serves as the threshold for this comparer.

3.2.6 Earth Mover's Comparer

The Earth Mover's comparer computes the Earth Mover's Distance (EMD) [21] between two histograms. The EMD between two histograms is the minimum work necessary to transform one histogram into another. Work is defined as the distance between the medians of the bins of a histogram weighted by their probabilities. The name brings to mind an accurate metaphor of the amount of earth movement necessary to completely cover the smaller histogram with

the earth from the larger histogram if the weights were piles of earth. Although it is only used to compare histograms in the detector, the EMD is sufficiently general to compare other models. Since this comparer currently compares normalized histograms, the distance metric is equivalent to the Mallows distance [16], but the implementation is based on a publicly available implemenation of the EMD [20].

3.2.7 Mahalanobis Comparer

The Mahalanobis comparer computes the distance from a point to the centroid of a cluster accounting for the variance. The formula [7] is:

$$d(\bar{\mathbf{x}}, \mathbf{x}') \quad = \quad (\mathbf{x}' - \bar{\mathbf{x}})^T \Sigma^{-1} (\mathbf{x}' - \bar{\mathbf{x}})$$

where $\mathbf{x}'$ is the column vector of a data point x projected into the model space, $\bar{\mathbf{x}}$ is the mean of the normal data, and Σ^{-1} is the inverse of the sample covariance matrix of the normal data. In the one-dimensional case, for example, the Mahalanobis distance is simply the standardization of a variable with respect to a mean and standard deviation: $d(\mu, x) = \frac{x-\mu}{\sigma}$. Although it makes the assumption that the points are from a normal probability distribution, it is still useful in practice.

3.2.8 Probability Comparer

The probability comparer determines the distance between a point and a histogram. It interprets the distance as the probability that the point does not exist as in the following formula:

$$d(x) \quad = \quad 1 - p(x)$$

where $p(x)$ is the probability of the point from the histogram. In situations where the point is outside the bounds on the bins of the histogram, the probability is zero and the distance is 1.

3.3 Selecting a Threshold

The basic functionality of an anomaly detector is to build a model of data, compare models, and claim that an anomaly exists when the incoming data significantly deviate from the normal model. So far, this paper has addressed the first two tasks. The last task is frequently overlooked in the literature [22, 18, 15]. The current implementation uses two methods to determine the threshold from training data. The first method compares a model to each of its training points and builds a histogram on the distances. The threshold is then set to a prescribed quantile of the distances. Although the idea of a threshold threshold does not reduce the number of parameters, it allows the user to set the same or similar thresholds across all detectors. The interpretation of an anomaly as being further from the model than 99% of the training data seems more intuitively understandable than distances of .8, 1×10^{-7}, and 2.54 for three different combinations of model builders and comparers. Despite the relative ease with which one can define the threshold for model-to-point comparers, a similar method for model-to-model comparers is more difficult. Since models are on time windows and training data are all that are available for determining the threshold, the current method splits the data in half and compares models of the halves. The threshold is the maximum distance between the models.

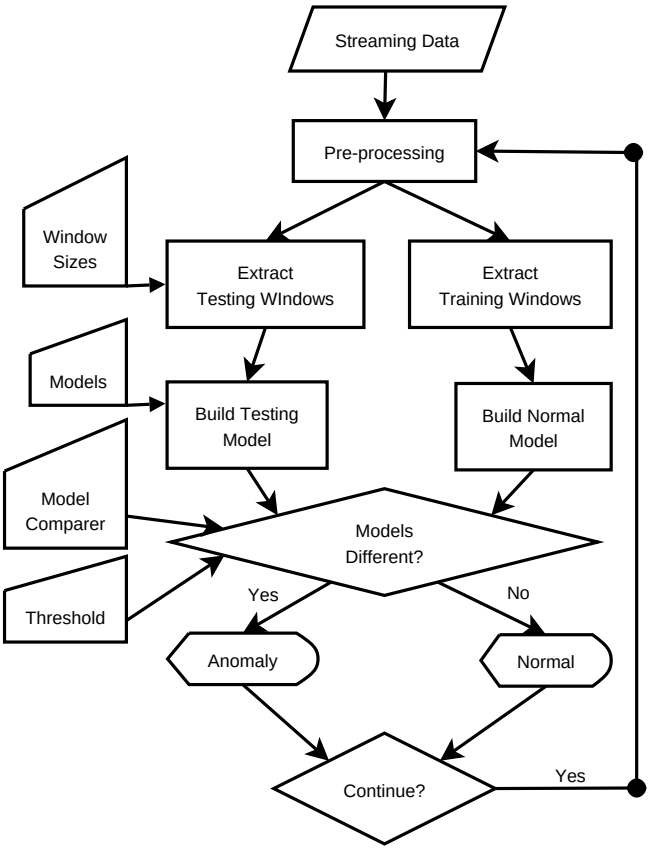

Figure 5: Anomaly detector flowchart.

3.4 Assembling the Detector

The last step in the design and implementation of an anomaly detector is to combine the components into a useful anomaly detector. This section describes the currently implemented example detectors. Figure 5 shows the basic operation of an anomaly detector with the previously described components. This section describes the operation of three different anomaly detectors: the model-to-model, model-to-point, and voting model-to-point detectors. Each of the detectors assumes that an additional component exists to manipulate the data. This component is responsible for retrieving the data, creating the necessary time windows, and interacting with the anomaly detector.

The model-to-model detector trains models on a chunk of normal and and possibly anomalous incoming data, determines the difference, and returns the detection results given a threshold. In the training phase, a threshold selector determines a threshold based on multiple comparisons on random halves of the data. In the testing phase, the detector compares the model of the testing data with the stored model of the training data using a provided model-to-model comparer. For evaluation of the detector, the label of a time window is normal only when no attack points are present.

The model-to-point detector compares models to single points. Aside from this, the function

of the model-to-point detector is analogous to that of the model-to-model detector.

The voting model-to-point detector makes a decision based on a vote. As a chunk of training data arrives, the detector trains several models on it. After training, it selects a separate threshold for each of the models based on the threshold selection algorithm and a predetermined quantile parameter. During testing, the detector compares the point with each of the models using a potentially different comparer for each. This testing procedure results in one vote from each model-comparer pair. If the voting results show a 50% majority of votes for an anomaly, the detector claims that the point is anomalous; otherwise, the point is normal.

4 Experiments

This section describes experiments that help to determine whether the implementation works and whether it can reproduce known experiments from the literature. The first experiment tests the sketch model on machine learning data to determine whether it can successfully reduce the dimensionality of the data without extremely adverse affects on classification accuracy. Secondly, a test on network intrusion data compared the detection rates of several of the model-to-point comparers in an attempt to replicate another previous experiment. Lastly, an artificially generated time-variant process tested the detection rate of the model-to-model comparers.

The first experiment is a replication of an previous one to determine the effect of a random projection of data on classification accuracy. The previous experiment [8] performed a projection of training and testing data onto a set of random basis vectors prior to classification. After projection, the experiment classified the data with one and five nearest-neighbor, decision tree, and support vector machine classifiers [8]. The datasets are the ad, colon, ionosphere, and spambase datasets from the UCI Machine Learning Repository [4]. In the current experiment, the data were standardized using the mean and standard deviation, and the experiment performed ten runs of ten-fold cross-validation [7] and did include the support vector machine classifier. Results from the previous experiment showed that random projections did not lower the accuracy results on distance based classifiers as much as one might intuitively expect. The current experiment closely followed the projection methodology and used the same data sets. The results as shown in Table 1 demonstrate that for data sets with a large number of examples (ad, ionosphere, and spambase) the accuracy results for the current experiment support the conclusion of the previous experiment.

The second experiment follows the spirit another previous experiment but uses different data and adds several anomaly detection methods. The previous experiment [15] compared four model-to-point comparers on a customized version of the Lincoln Labs 1998 DARPA evaluation dataset [18]. The bases for comparison were the detection rate of attacks and the amount of time necessary to detect an attack once it began. The current experiment follows the training and testing methods of the previous experiment but uses a the KDD Cup 1999 dataset [1] which is also based on the DARPA 1998 evaluation dataset [18]. The main differences between the two datasets are that the KDD data are publicly available, labeled, and already formatted. The other dataset included temporal and aggregate information from the original DARPA dataset that is unavailable in the KDD data. The KDD data consist of several thousand sequentially ordered connection records from one week of simulated network traffic. Each connection includes forty-

Features	Classifier	ad	colon	ionosphere	spambase
5	1-NN	91.99	1.88	56.67	5.44
5	stdev	1.88	5.44	1.81	.77
5	5-NN	89.56	61.33	79.03	76.66
5	stdev	2.25	5.2	2.62	.93
5	DT	95.96	62	64.06	63.05
5	stdev	.4	6.26	2.39	1.34
15	1-NN	92.07	49.67	86.86	89.01
15	stdev	2.36	4.57	1.59	.44
15	5-NN	89.31	57.33	79.54	86.43
15	stdev	2.38	5.74	2.32	.67
15	DT	95.83	58.33	66.86	72.7
15	stdev	.33	5.35	3.55	.89
30	1-NN	92.3	54	87.26	90.83
30	stdev	2.18	6.04	1.98	.33
30	5-NN	90.47	55.33	78	88
30	stdev	2.08	6.91	2.34	.51
30	DT	95.9	55.67	77.66	74.99
30	stdev	.29	7.05	2.24	.87
All	1-NN		53.33	86.57	91.24
All	stdev		6.7	1.86	.37
All	5-NN		62	79.37	88.79
All	stdev		5.29	2.15	.49
All	DT		64.67	78.4	76.51
All	stdev		5.07	2.36	.86

Table 1: Results for the machine learning experiment replication. The features column indicates the number of reduced features used prior to classification. The DT classifier is the C4.5 decision tree. Results are noticeably absent for the ad dataset because the experiment consumed so much memory that the program terminated without completing the measurement.

one features of which thirty-four are numeric. Most of the matrix-based models in the current experiment only use the numeric features; furthermore, the current experiment only utilizes the published ten-percent sampling of the connections for all tests. In the tests, the data are split by class label. A random sample reduces the over ninety thousand normal connections to five sets of one- and six-thousand connections. In combination with a single attack class, the five-thousand-connection training and one-thousand-connection testing normal sets form a single evaluation set. Since the data include a total of twenty-two attack classes and the experiment has five runs, a total of one-hundred ten individual test files form the basis of the experiment. On each of these test files, the experiment compared the all model builders from Section 3.1 with the Mahalanobis Comparer from Section 3.2.7 for the matrix-based models and the Probability Comparer from Section 3.2.8 for the histograms. The experiment tested each builder-comparer combination alone on the data and with the majority vote of histogram, SVD, Sketch, PCA, and matrix model builders with 5 reduced dimensions. The threshold arises from selection method described in Section 3.3 with quantiles in the range of 50% to 100%. The model-specific parameters include the number of reduced dimensions for models that reduce data dimensionality and the feature index for the histograms. For the number of dimensions, the values were from 5 to 25 in increments of 5, and indices ranged over the entire range of indices in the dataset. The number of combinations of experiment runs yielded an extremely large experiment that took several days of computing time on a 2.5 GHz processor. To abate the large amount of time necessary, the Mahalanobis distance maintained a copy of the the inverse of the covariance matrix so that it did not have to recalculate this matrix for every point. This single performance enhancement should not adversely affect the results. Table 3 shows the true positive rate for the various attack classes for each of the detectors except for the histograms because the results were much worse than for the others. Figure 6 shows a comparison of the detection performances for various model builders that use 15 reduced dimensions in their computations.

The third and final experiment uses completely artificial data because the author could not find publicly available, properly formatted, and correctly labeled temporally ordered data. Each of the one hundred features in the data is statistically independent and distributed according to one of three one-dimensional Gaussian distributions. To allow for some temporal component to the data, the particular distribution depends on a simple first-order Markov chain [7] where each state yields the parameters of a normal distribution as Figure 7 illustrates. The illustrated model is parameterized for different anomaly rates. The aa parameter is the self-transition to the anomaly state and provides an expectation of how many anomalous points appear in a given time window if the window contains an anomaly. The na parameter specifies the probability that a window will be anomalous. For the current experiment, this parameter is set to generate roughly the same number of attack windows versus normal ones. With these parameters, the experiment tests the false positive rate of the different detectors given the number of anomalies in the window. To generate the data, the generator starts in normal state 1 and generates one hundred five samples from a normal distribution with the indicated mean and standard deviation. After the initial generation, the generator moves to the different states with the shown probabilities. The anomalous state 3 has small transition probabilities and significantly higher mean and smaller standard deviation. The resultant data consist of 105,000 points from these distributions, and the window length is one hundred four points. In this highly

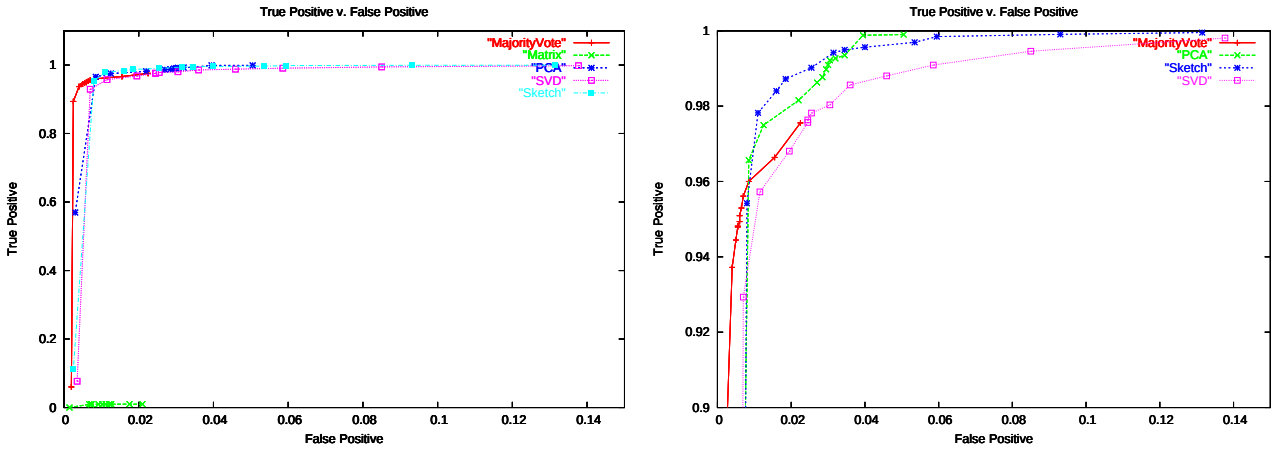

Figure 6: ROC curves for several detectors where the points are computed by the weighted average of class true positive given the number of connections in each class.

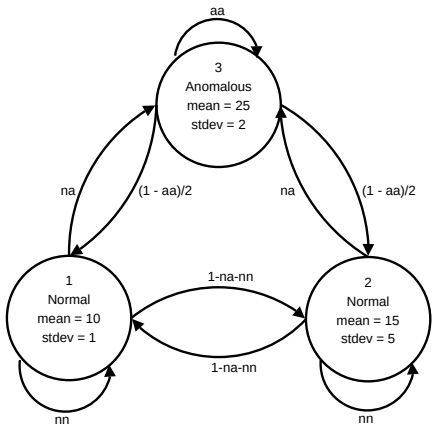

Figure 7: The Markov chain that generated the artificial data in the third experiment. Transition parameters allow for tuned datasets.

Connections	Attack Class
2203	back
30	buffer_overflow
8	ftp_write
53	guess_passwd
12	imap
1247	ipsweep
21	land
9	loadmodule
7	multihop
107201	neptune
231	nmap
3	perl
4	phf
264	pod
1040	portsweep
10	rootkit
1589	satan
280790	smurf
2	spy
979	teardrop
1020	warezclient
20	warezmaster

Table 2: Number of connections in each class for the KDD Cup 1999 network intrusion dataset [1] using 10% of the data.

Class k	Vote	PCA 10	PCA 15	PCA 20	PCA 5	Data	SVD 10	SVD 15	SVD 20	SVD 5	Sktch 10	Sktch 15	Sktch 20	Sktch 5
back	0.5084	0.5020	0.5007	0.5286	0.5263	0.4893	0.9959	0.9930	0.9796	0.9986	0.9961	0.9927	0.6770	0.9977
buffer over-flow	0.0833	0.7833	0.7833	0.4667	0.7833	0.0167	0.1000	0.2333	0.3667	0.0167	0.1000	0.6667	0.7667	0.0167
ftp write	0.0625	0.5625	0.5625	0.4375	0.3750	0.0625	0.0000	0.0000	0.3750	0.0625	0.0625	0.5000	0.3750	0.0625
guess passwd	0.0000	0.9811	0.9906	0.5094	0.9811	0.0000	0.0000	0.0000	0.4906	0.0000	0.0755	0.9811	0.5283	0.0000
imap	0.7500	0.9583	0.9583	0.9583	1.0000	0.0833	0.7500	0.7917	0.7083	0.0833	0.7917	0.8750	0.8750	0.0833
ipsweep	0.0000	0.5842	0.5922	0.5902	0.1556	0.0000	0.0000	0.0000	0.0004	0.0000	0.0000	0.0172	0.2594	0.0000
land	0.0476	1.0000	1.0000	1.0000	1.0000	0.0000	0.0476	0.0476	0.0476	0.0000	0.1190	1.0000	1.0000	0.0000
load mod-ule	0.0000	0.6111	0.7222	0.3889	0.6667	0.0000	0.0000	0.2222	0.2778	0.0000	0.0000	0.6111	0.6111	0.0000
multihop	0.5000	0.5000	0.5000	0.2143	0.6429	0.2857	0.5000	0.5714	0.5714	0.2857	0.5714	0.7143	0.7143	0.2857
neptune	0.9766	1.0000	1.0000	0.9952	1.0000	0.0000	0.9766	0.9817	0.9891	0.0000	0.9891	0.9995	0.9999	0.0000
nmap	0.0000	0.4502	0.4502	0.4502	0.4502	0.0000	0.0000	0.0000	0.0065	0.0000	0.0022	0.4459	0.4502	0.0000
perl	0.0000	1.0000	1.0000	0.5000	1.0000	0.0000	0.0000	0.3333	0.0000	0.0000	0.0000	1.0000	1.0000	0.0000
phf	0.0000	1.0000	1.0000	0.5000	1.0000	0.0000	0.0000	0.0000	0.1250	0.0000	0.0000	0.0000	1.0000	0.0000
pod	0.0000	0.9811	0.9811	0.4905	0.9811	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.8106	0.1648	0.0000
port-sweep	0.1500	0.1981	0.1813	0.2572	0.2736	0.0010	0.1505	0.1644	0.2365	0.0745	0.1548	0.2303	0.3313	0.0740
rootkit	0.0000	0.4500	0.4500	0.3500	0.5000	0.0000	0.0000	0.2500	0.3000	0.0000	0.2000	0.4000	0.3000	0.0000
satan	0.4953	0.5689	0.8518	0.9537	0.5931	0.0000	0.8807	0.9037	0.9308	0.0000	0.8880	0.8927	0.9276	0.0000
smurf	0.0000	0.0000			0.0000	0.0000	0.9994			0.0000	0.9994			0.9991
normal	0.0040	0.0120	0.0125	0.0120	0.0125	0.0073	0.0135	0.0115	0.0080	0.0085	0.0135	0.0110	0.0105	0.0114

Table 3: Table of the average true positive rates for each of the attack classes in one run of the network intrusion detection experiment with threshold of 98%. The values for the normal class are the false positive rate. Classes and results that are not visible were not available at the time of completion of this report. In the table, the Sketch model builder is abbreviated with Sktch.

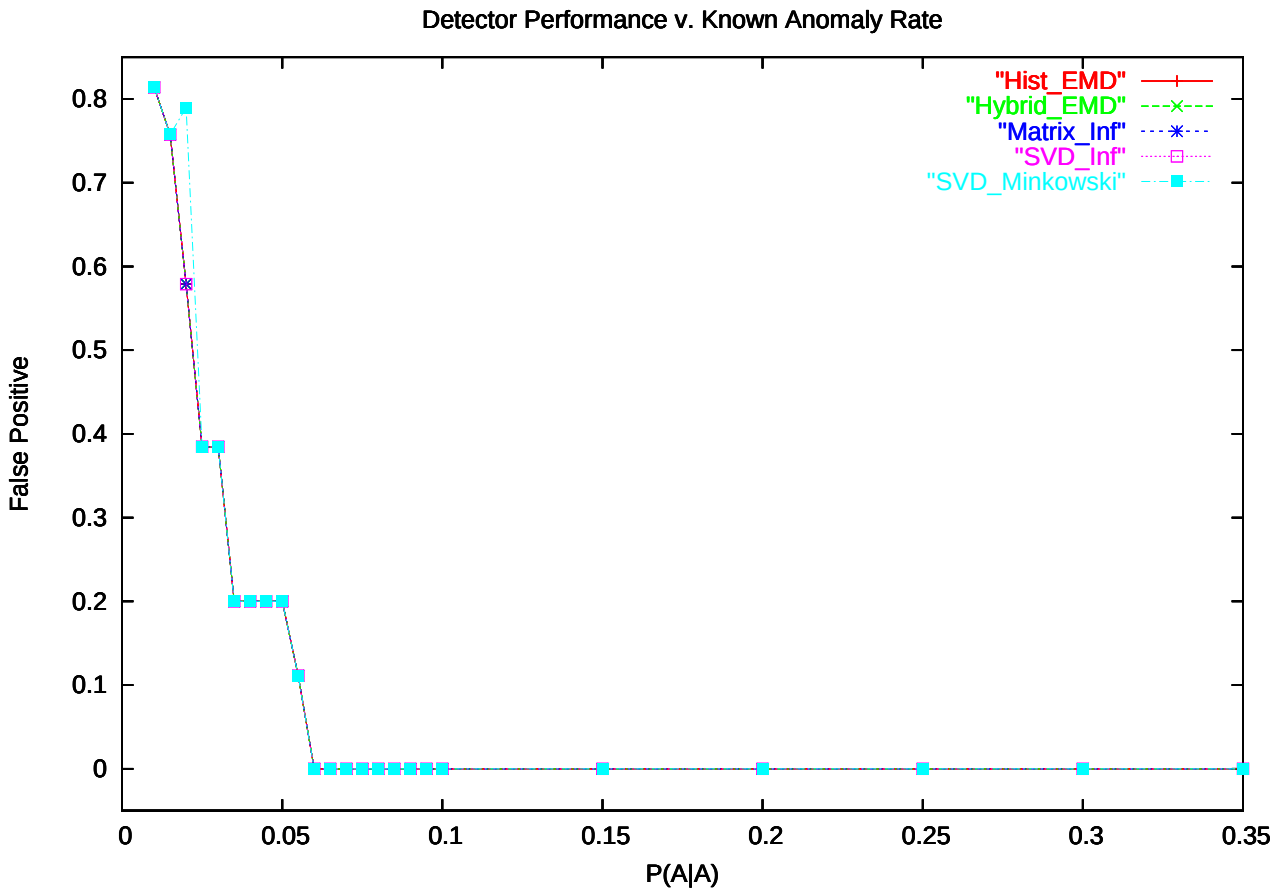

Figure 8: The curve for the performance of the detector with several combinations of model builders and model comparers for the third experiment. The horizontal axis lists the parameter aa from Figure 7, and the vertical axis lists the false positive rate where the positive class is the anomaly.

contrived example, one consequently expects the anomaly detector to perform very well. Figure 8 shows the curves of the detector performances for a few models and comparers in which all curves appear to be exactly the same. Although the true positive performance measures are not shown, they are generally very good except for the PCA and Sketch models. The typical performance results for these two anomalous anomaly detectors are either always not detecting an anomaly or always detecting an anomaly. As a likely explanation, implementation error is to blame. The comparers to do require that the two models be of the same data, and the anomaly detector does not check that they are of the same data. The problem is in the implementation of the experiment rather than the implementation of the components which is the primary focus of the experiment.

5 Conclusion

This paper describes the design and implementation of a general-purpose anomaly detector for the streaming data model [11]. In this model, data points continually arrive in consecutive order, so any anomaly detector must work on the data as they arrive. The anomaly detector that is the subject of this paper builds a model on normal data, compares this model to incoming data against a threshold to determine when the incoming data are anomalous. Models compactly represent the data but still allow for effective comparison. The implementation selects several transformation and histogram models from the literature. Model comparison methods determine the distance between two models of data or the distance between a model and a point, and the implementation again selects several matrix-based and probabilistic comparison methods from the literature. Since threshold selection is overlooked in the literature, the implementation includes two methods to estimate thresholds from normal data. With these individually useful components, a user can construct a variety of anomaly detection schemes. A series of three tests several of these schemes to determine their performance on two well-known and one artificial dataset. The results verify the conclusions from the previous experiments.

Although useful, the current implementation is just a start in the development of an anomaly detector. Future work should look to more components for the detector, an interactive graphical user interface, better test data, and methods for control in addition to detection. The problem of anomaly detection is important and challenging, but it is still only a small component of an autonomous control system. Such a system should be able to use an anomaly detector to detect changes in a process and then decide what to do to the process to restore normalcy or realize that the changed process is normal.

Acknowledgments This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Abraham Bagherjeiran is a graduate student at the University of Houston. This work was done when he was a student intern at LLNL.

References

[1] KDD Cup 1999 data. Available at: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, October 1999.

[2] Dimitris Achlioptas. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 274–281. ACM Press, 2001.

[3] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: Applications to image and text data. In *KDD 2001*, 2001.

[4] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[5] Matthew Brand. Fast online SVD revisions for lightweight recommender systems. In *Proceedings of the Third SIAM International Conference on Data Mining*, 2003.

[6] Datar. Maintaining stream statistics over sliding windows. *SIAM Journal of Computer Science*, 31(8):1794–1813.

[7] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* Wiley-Interscience, 2nd edition, 2001.

[8] Dmitriy Fradkin and David Madigan. Experiments with random projections for machine learning. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 517–522. ACM Press, 2003.

[9] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Quicksand: Quick summary and analysis of network data. Technical Report 2001-43, DIMACS, 2001.

[10] Gene H. Golub and Charles Van Loan. *Matrix Computations.* Johns Hopkins UP, 3rd edition, 1996.

[11] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical Report 1998-011, Digital Systems Research Center, 1998.

[12] William B. Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. In *Conference in Modern Analysis and Probability*, pages 189–206, 1984.

[13] Sukhbinder Kumar, Elaine B. Martin, and Julian Morris. Detection of process model changes in PCA based performance monitoring. *Proceedings of the American Control Conference*, 2002.

[14] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. Technical Report BUCS-TR-2004-008, Boston University, 2004.

[15] Aleksandar Lazarevic, Levent Ertoz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the Third SIAM International Conference on Data Mining*, 2003.

[16] E. Levina and P. Bickel. The earth mover's distance is the mallows distance: some insights from statistics. In *Proceedings of the Eighth IEEE International Conference on Computer Vision*, volume 2, pages 251–256, 2001.

[17] Bernard W. Lindgren. *Statistical Theory.* Macmillan, 3rd edition, 1976.

[18] Richard P. Lippmann, David J. Fried, Isaac Graf, Joshua W. Haines, Kristopher R. Kendall, David McClung, Dan Weber, Seth E. Webster, Dan Wyschogrod, Robert K. Cunningham, and Marc A. Zissman. Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, volume 2, pages 12–26, 2000.

[19] Lin Qiao, Divyakant Agrawal, and Amr El Abbadi. Supporting sliding window queries for continuous data streams. In *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*, 2003.

[20] Y. Rubner and C. Tomasi. Code for the earth movers distance. web page available at: http://www.cs.duke.edu/ tomasi/software/emd.htm.

[21] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. A metric for distributions with applications to image databases. In *Proceedings of the 1998 IEEE International Conference on Computer Vision*, 1998.

[22] Hiren Shah, Jeffrey Undercoffer, and Anupam Joshi. Fuzzy clustering for intrusion detection. In *Proceedings of the 12th IEEE International Conference on Fuzzy Systems*, April 2003.

[23] Cyrus Shahabi and Donghui Yan. Real-time pattern isolation and recognition over immersive sensor data streams. In *Proceedings of the 9th International Conference on Multi-Media Modeling*, pages 93–113, 2003.

[24] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and Li Wu Chang. A novel anomaly detection scheme based on principal component classifier. In *Proceedings of the IEEE Foundations and New Directions of Data Mining Workshop, in conjunction with the Third IEEE International Conference on Data Mining (ICDM'03)*, 2003.

[25] Ashish Singhal and Dale E. Seborg. Matching patterns from historical data using PCA and distance similarity factors. In *Proceedings for the Americal Control Conference*, 2001.

[26] Carol Taylor and Jim Alves-Foss. "Low cost" network intrusion detection. In *Proceedings of the New Security Paradigms Workshop*, pages 89–96, September 2001.

[27] David S. Watkins. *Fundamentals of matrix computations.* Wiley, 1991.

[28] Robert L. Winkler. *Statistics, Probability, Inference, and Decision.* Holt, Rinehart and Winston, 1975.