Fast Lightweight Outlier Detection in Mixed-Attribute Data

Matthew Eric Otey, Srinivasan Parthasarathy and Amol Ghoting
Department of Computer Science and Engineering
The Ohio State University
Contact: srini@cse.ohio-state.edu

Abstract

In recent years, researchers have proposed many techniques for detecting outliers in data sets. However, most of these techniques assume that the data set is static and consists of homogeneous attribute types. However, these assumptions do not hold for many real-world data sets. To address these weaknesses, we present a technique for outlier detection in dynamic mixed-attribute data. Our technique is capable of finding outliers in a single pass of the data, and can do so with low memory requirements. Our approach uses a combination of classifiers and statistical tests to discover anomalous values of categorical and continuous attributes. Our empirical results demonstrate that while our technique only shows marginal improvements in detection rates, its execution speed and memory usage are far better than those of current state-of-the-art outlier detection techniques.

1 Introduction

A common problem in data mining is that of automatically finding outliers in a database. This is an important problem, since they can be indicative of bad data or malicious behavior. Examples of bad data include skewed data values resulting from measurement error, or erroneous values resulting from data entry mistakes. A common example of data indicating malicious behavior occurs in the field of network traffic analysis, where anomalous IP packets may indicate either a possible intrusion or attack, or a failure in the network [20]. Efficient detection of such outliers reduces the risk of making poor decisions based on erroneous data, and aids in identifying, preventing, and repairing the effects of malicious or faulty behavior.

Additionally, many data mining algorithms and statistical analysis techniques may not work well in the presence of outliers. Outliers may introduce skew or complexity into models of the data, which make it difficult to fit an accurate model to the data. Statistical measures of the data may be skewed because of erroneous values, or the noise of the outliers may obscure the truly valuable information residing in the data set. Accurate, efficient removal of outliers may greatly enhance the performance of statistical techniques and data mining algorithms [5]. As can be seen, different domains have different reasons for discovering outliers: They may be noise that we want to remove, since they obscure the true patterns we wish to discover, or they may be the very things in the data that we wish to discover. As has been said before, "One person's noise is another person's signal" [14].

Over the years, a large number of techniques have been developed for outlier detection. However, real-world data sets present a range of difficulties that limit the effectiveness of these techniques. One of these difficulties is that the data set may be dynamic. Another is that the data set may contain a mixture of attribute types (i.e. categorical and continuous attributes).

Many interesting data sets are dynamic or evolving, meaning that data is constantly being added. In many cases, the processes generating this new data may be non-stationary, meaning that a model of the data built at one point in time may become invalid in the future. An example of such a data set is a network data stream. In network intrusion detection, the models of network traffic and intrusion patterns are in a state of constant change, as new network services, virii, and intrusion techniques are constantly being developed and modified. Since it is too expensive to rebuild a model of the data for each new data point, an efficient outlier detection technique must be able to build an accurate model of the data incrementally.

Also, the features in a data set may be a mixture of categorical (nominal) and continuous types. Many techniques for outlier detection assume that the data set contains attributes that are either all continuous or all categorical. However, many real-life data sets contain a mixture of types. For example, a data point representing a network connection contains continuous attributes (e.g. the duration of the connection) and categorical attributes (e.g. the protocol used). Having different attribute types make it difficult to find relations between two or more attributes and to define distance or similarity metrics. When processing data sets with a mixture of attribute types, many techniques either discretize the continuous attributes, or

convert the categorical attributes into continuous attributes by applying some (arbitrary) ordering, which can lead to a loss in information and an increase in noise. An efficient outlier detection system needs to quantify the relations between features of different types.

We have previously proposed an algorithm called LOADED to discover outliers in dynamic data sets containing a heterogeneous mixture of attribute types [6]. However, LOADED suffers from high memory requirements. In this paper we propose an alternative to LOADED named RELOADED that drastically reduces the memory requirements of LOADED, as well as improves on its scalability, at a small cost to accuracy.

The rest of this paper is organized as follows. In Section 2 we discuss previous work in the domain of outlier detection. In Section 3 we present our new RELOADED outlier detection algorithm. In Section 4 we present the results of our evaluation of RELOADED. Finally in Section 5 we draw conclusions about our results and present directions for future work.

2 Related Work

There are several approaches to outlier detection. One approach is that of statistical model-based outlier detection, where the data is assumed to follow a parametric (typically univariate) distribution [1]. Such approaches do not work well in even moderately high-dimensional (multivariate) spaces, and finding the right model is often a difficult task in its own right. Simplified probabilistic models suffer from a high false positive rate [16, 17]. Also, methods based on computational geometry [10] do not scale well as the number of dimensions increase. To overcome these limitations, researchers have turned to various non-parametric approaches including distance-based approaches [2, 11], clustering-based approaches [7, 20], and density-based approaches [4, 19]. Here we consider these methods in more detail.

An approach for discovering outliers using distance metrics was first presented by Knorr *et al.* [12, 13, 11]. They define a point to be a *distance outlier* if at least a user-defined fraction of the points in the data set are further away than some user-defined minimum distance from that point. In their experiments, they primarily focus on data sets containing only continuous attributes.

Related to distance-based methods are methods that cluster data and find outliers as part of the process of clustering [9]. Points that do not cluster well are labeled as outliers. This is the approach used by the ADMIT intrusion detection system [20]. A clear limitation of clustering-based approaches to outlier detection is that they require multiple passes to process the data set.

Recently, density-based approaches to outlier detection have been proposed [4]. In this approach, a local outlier factor (LOF) is computed for each point. The LOF of a point is based on the ratios of the local density of the area around the point and the local densities of its neighbors. The size of a neighborhood of a point is determined by the area containing a user-supplied minimum number of points ($MinPts$). A similar technique called LOCI (Local Correlation Integral) is presented in [19]. LOCI addresses the difficulty of choosing values for $MinPts$ in the LOF-based technique by using statistical values derived from the data itself. However, both the LOF- and LOCI-based approaches do not scale well with a large number of attributes and data points.

Most distance-based methods for detecting outliers take time that is at least quadratic in the number of points in the data set, which may be unacceptable if the data set is very large or dynamic. Bay and Schwabacher [2] present a method called ORCA for discovering outliers in near linear time. The central idea is to perform pruning by keeping a monotonically decreasing score for each point in the data set. If the score falls below a certain threshold, then further processing on the data point is not necessary. In the worst case (when there are no outliers), the algorithm still takes quadratic time, but in practice the algorithm runs very close to linear time. Such an approach assumes that the data set is randomized, and randomization is performed on disk prior to running the algorithm. ORCA can handle mixed-attribute data sets by using the Euclidean distance for the continuous attributes and the Hamming distance for the categorical attributes. However, the Hamming distance is not always effective as it does not consider dependencies between categorical attributes.

A comparison of various anomaly detection schemes is presented in [15]. Its focus is on how well different schemes perform with respect to detecting network intrusions. The authors used the 1998 DARPA network connection data set to perform their evaluation, which is the basis of the KDDCup 1999 data set used in our experiments [8]. They found detection rates ranging from a low of 52.63% for a Mahalanobis distance-based approach, to a high of 84.2% for an approach using support vector machines.

2.1 LOADED

The LOADED (Link-based Outlier and Anomaly Detection in Evolving Data sets) algorithm was first presented in [6]. It is designed explicitly for dynamic data with heterogeneous attributes. The central data structure used to model the data is an augmented lattice of all itemsets formed from the categorical attributes of the data. Each node in the lattice is augmented with the support count of the corresponding itemset, and the correlation matrix computed from the continuous attributes of all data points in the data set containing that itemset. Such a data structure ensures that the dependencies between all attributes, regardless of type, can be modeled. Each data point is assigned an anomaly score based on the support of all

its itemsets and how well the continuous attributes agree with the relevant correlation matrices. The basic algorithm makes a single pass of the data set, incrementally updating the lattice for each data point processed. The algorithm is also able to make a second pass of static data sets, which allows for better detection rates. Finally, it is also possible to constrain the size of the lattice to conserve memory, at a small cost to accuracy.

3 Outlier Detection Algorithm

As mentioned above, the original LOADED algorithm maintains a complete itemset lattice in memory, where each node of the lattice is augmented by a correlation or covariance matrix. Since each matrix is quadratic in size with respect to the number of continuous attributes, and the itemset lattice is exponential in size with respect to the number of categorical attributes, LOADED requires a relatively large amount of memory.

To address this memory consumption issue, we propose the RELOADED (*RE*duced memory LOADED) algorithm. RELOADED, like LOADED, uses a set of covariance matrices to discover anomalous values of continuous attributes, but unlike LOADED, it uses a set of classifiers to detect anomalous values of categorical attributes. To characterize the relationships between the categorical and continuous attributes, the covariance matrices are conditioned on the values of the categorical attributes, while the classifiers take into account the values of the continuous attributes. We can then define an anomalous data point as one that has a subset of attributes that take on unusual values given the values of the other attributes. The details of our approach are given below.

3.1 Predicting Values of Categorical Attributes

Consider a data point $P = (P_c, P_q)$ where P_c is a vector of m categorical attributes and P_q is a vector of n continuous (quantitative) attributes. To discover anomalous values of categorical attributes, we use a set of m classifiers to predict the values of each of the m categorical attributes and then compare against their actual values. The classifier C_i used to predict the categorical attribute P_{ci} is trained on data points of the form $P^i = (P_c^i, P_q)$, where P_c^i is the vector of categorical attributes with the ith element removed. In turn, the value of the ith attribute is used as the class label. For example, assume that our data set has four attributes:

$$P = (N_1, N_2, Q_1, Q_2)$$

where the N's and Q's are respectively the categorical and continuous attributes. We use two classifiers C_1 and C_2 to predict the values of N_1 and N_2:

$$\hat{N}_1 = C_1(N_2, Q_1, Q_2)$$

$$\hat{N}_2 = C_2(N_1, Q_1, Q_2).$$

If for attribute i, the predicted value $\hat{N}_i$ disagrees with that of the true value N_i, we increase the value of our anomaly score for that point. In this work, we utilize Naïve Bayes Classifiers, as they are relatively light-weight classifiers, and can be trained incrementally, which is useful for our single-pass approach to anomaly detection. To train each classifier, we only need to calculate and store the probability mass function for each attribute, and since we assume that the distributions of the continuous attributes to be normal, we only need to store the mean and standard deviation for each continuous attribute.

3.2 Finding Anomalous Values of Continuous Attributes

Again, consider the data point $P = (P_c, P_q)$. We wish to discover to what degree P_q violates the covariances conditioned on the values of the categorical attributes. First, note that P_c can be considered as a set of attribute-value pairs:

$$P_c = \{(attribute_1, value_1), \ldots, (attribute_m, value_m)\}.$$

Note that LOADED maintains correlation or covariance matrices for all subsets of all the P_c in the data set, which is the root cause of its high memory consumption problem. To reduce memory consumption, RELOADED only maintains a covariance matrix for each unique attribute-value pair in the data set. To illustrate the magnitude of this reduction, consider a data set with m categorical attributes, where the ith attribute can take on r_i distinct values. In this case, RELOADED must maintain $\Sigma_{i=1}^{m} r_i$ covariance matrices in the worst case, while LOADED must maintain on the order of $2^m \Pi_{i=1}^{m} r_i$ matrices in the worst case.

As stated above, for each unique attribute-value pair in the data set, there is a covariance matrix C. Let C^d be the covariance matrix for the attribute-value pair d, where C_{ij}^d is the covariance between continuous attributes i and j. To determine the degree to which P violates C_{ij}^d, we calculate the covariance score $c_{ij}^{P,d}$ for each pair of continuous attributes i and j of P:

$$c_{ij}^{P,d} = (P_{qi} - \mu_i^d) \times (P_{qj} - \mu_j^d) \tag{1}$$

where μ_i^d and μ_j^d are the means of continuous attributes P_{qi} and P_{qj} for all points in the data set containing the attribute-value pair d. We can then compute the violation score V of P_q as:

$$V_\tau(P_q) = \sum_i \sum_j v_\tau(P_{qij}) \tag{2}$$

where

$$v_\tau(P_{qij}) = \begin{cases} 0 & \text{if } | \frac{c_{ij}^{P,d}}{C_{ij}^d} | \leq \tau \\ 1 & \text{otherwise.} \end{cases} \tag{3}$$

Procedure: Reloaded
Input: $DataSet$, τ
Output: AnomalyScores
1 For each point $P = (P_c, P_q) \in DataSet$:
2 $\#WrongPred = 0$
3 $VScore = 0$
4 For each attribute-value pair $d \in P_c$:
5 P'_c = P_c with d removed
6 Train $Classifier_d$ on P'_c
7 $Pred_d$ = $Classifier_d(P'_c)$
8 if ($d \neq Pred_d$)
9 $\#WrongPred++$
10 Use P_q to update $Covariance_d$
11 $VScore+ = V_\tau(P_q)$
12 End For
13 $AnomalyScore[P] = f(\#WrongPred, VScore)$
14 End For
end

Figure 1. The RELOADED algorithm

The function v_τ captures how much the covariance score of a pair of continuous attributes deviates from the covariance learned from the data set. If a pair has a deviation greater than τ, the violation score is incremented.

3.3 The RELOADED Algorithm

The RELOADED algorithm for anomaly detection can be seen in Figure 1. The version presented in the figure is the single-pass version, which trains the classifiers and computes the covariance matrices incrementally. Therefore, the decision as to whether a given point is an anomaly or not is based only on the previously processed data points. The algorithm operates as follows. For each point in the data set, and for each categorical attribute d of that data point, the appropriate classifier is trained. That classifier, in turn, is used to predict the appropriate value of d. If the prediction is wrong, the count of incorrect predictions is incremented. Next, the continuous attributes of the data point are used to incrementally compute the covariance matrix corresponding to the attribute-value pair d. The cumulative violation score of the data point is incremented by the result of the V_τ function (see Equations 2 and 3).

When all categorical attributes of a data point have been processed, the anomaly score of the data point is computed as a function f of the count of incorrect predictions and the violation score. The function f allows one to weight anomalies in the categorical and continuous attributes differently. One can see that if there are m categorical attributes and n continuous attributes, $VScore$ can achieve a maximum value of $m \times n^2$, while there can only be m incorrect predictions of the categorical attribute values, meaning a simple summation or product of the two values would be much more sensitive to anomalous continuous values. In our implementation, we use the following function for point P_i:

$$AnomalyScore[P_i] = \frac{(\frac{\sum_{j=1}^{m} i - W_j}{i})}{m} + \frac{V_\tau}{mn^2} \quad (4)$$

where W_j is the cumulative number of incorrect predictions of categorical attribute j for previous i data points. Such a formula ensures that the categorical and continuous attributes have equal weight, and that categorical attributes with little or no correlation with the other attributes will not adversely affect the anomaly score (such attributes will have a large value for W_j, since they are often mispredicted). Finally, we must choose a threshold for $AnomalyScore$ in order to discriminate between the outliers and normal points. We do this by incrementally computing the mean and standard deviation of the $AnomalyScore$s seen so far, and flag any point as an outlier if it is more than s standard deviations greater than the current mean.

We note that it is straight forward to convert the single-pass algorithm presented in Figure 1 into a two-pass algorithm for static data. We simply make two passes over the data set. In the first pass, we omit steps 2-3, 7-9, 11 and 13, while in the second pass we omit steps 6 and 10. We also note that the version of RELOADED presented above places equal weight on all previously processed points. Since older points are usually less relevant than more recent ones, we can implement a sliding-window variant that periodically instantiates new classifiers and covariance matrices and disposes of older ones.

3.4 Computational and Space Complexity

Consider a data set with N data points, having m categorical attributes and n continuous attributes, and let each of the m categorical attributes take on a maximum of k distinct values. The processing of a data point has two stages. In the first stage, m Naïve Bayes Classifiers are run to predict the values of each of the m categorical attributes based on the remaining attributes. If the the classifier's probability mass functions are stored in hash tables, the time it takes to train a single classifier on a single point is $O(m+n)$. For classification of a point, for each of the m categorical attributes there are at most k classes we must consider. The time necessary for computing the probability of a point belonging to a class is $O(m+n)$, and so the time required to predict the values of each of the m categorical attributes is $O(mk(m+n)) = O(km^2 + kmn)$.

In the second stage we must incrementally compute the covariance matrix for the set of continuous attributes for each attribute-value pair of a given point. The time it takes to incrementally compute the covariance matrix is $O(n^2)$ for a given attribute-value pair. Since there are m such attribute-value pairs, the the total time taken to compute all covariances is $O(mn^2)$. Therefore the upper bound on the amount of time taken for each data

point is $O(mn^2 + km^2 + kmn)$, which gives an upper bound of $O(N(mn^2 + km^2 + kmn))$ for the algorithm running on the entire data set. As can be seen, the algorithm is linear in the number of data points and the number of categorical attribute values, and quadratic in the number of categorical and continuous attributes. For comparison, LOADED has a worst-case execution time of $O(Nn^2(km)^m)$ [6]. Like RELOADED, it is linear in the number of data points and quadratic in the number of continuous attributes, but it is exponential in the number of categorical attributes. For comparison, the ORCA outlier detection algorithm (see Section 2) has a worst-case complexity of $O(N^2(n + m))$, though empirically it appears to have complexity $O(N^d(n + m))$, where d is between 1 and 2 [2].

In terms of space complexity, RELOADED requires $O(m(km+n))$ space to hold the classifiers, and $O(kmn^2)$ space to hold the covariance matrices. The space requirements of our algorithm are thus independent of the size of the data set, and quadratic in the number of continuous and categorical attributes. This compares very favorably to LOADED, which requires $O(kmq^2 2^m)$ space in the worst case [6].

4 Experimental Results

4.1 Setup

We evaluate RELOADED's performance and compare it to that of LOADED and ORCA using a machine with a 2.8 GHz Pentium IV processor, and 1.5 GB of memory, running Mandrake Linux 10.1. Our implementations are in C++ and are compiled using gcc with O2 optimizations. Since ORCA finds the top-k outliers in a data set, we set k equal to the number of outliers in the data set. Also, we set LOADED to use only 4 lattice levels in all experiments. We compare these algorithms using the following data sets.

4.1.1 KDDCup 1999 Intrusion Detection Data

The 1999 KDDCup data set [8] contains a set of records that represent connections to a military computer network where there have been multiple intrusions and attacks. This data set was obtained from the UCI KDD archive [3]. The training data set has 4,898,430 data instances with 32 continuous attributes and 9 categorical attributes. The testing data set is smaller and contains several new intrusions that were not present in the training data set. Since these data sets have an unrealistic number of attacks, we preprocess them such that intrusions constitute 2% of the data set, and the proportions of different attacks is maintained. In network traffic, packets tend to occur in bursts for some intrusions. Therefore, intrusion instances are not randomly inserted into the data, but occur in bursts that are randomly distributed in the data set. The processed training data set contains 983,561 instances with 10,710 attack instances, while the processed testing data set contains 61,917 instances with 1,314 attack instances.

4.1.2 Adult Data

The Adult data set [3], contains 48,842 data instances with 6 continuous and 8 categorical attributes. Since the algorithms we test differ in their abilities to handle missing data, we removed all records containing missing data, leaving 32,561 records. The data was extracted from the US Census Bureau's Income data set. Each record contains an individual's demographic attributes together with a class label indicating whether the person made more or less than 50,000 dollars per year.

4.1.3 Synthetic Data

Since there are very few publicly available large mixed-attribute data sets, we wrote a synthetic data set generator to produce data to compare performance with existing algorithms, and with varying data set characteristics. The generator can produce data sets with a user-supplied number of continuous attributes and categorical attributes. The data points are generated according to a user-supplied multi-modal distribution. The exact details can be found in [18]. To create actual data sets for our experiments, we first generate a set of normal points from one distribution, and then separately generate a much smaller set of outliers from another distribution. The two sets are then randomly mixed to produce the final data set. However, we cannot guarantee that the distribution of the outliers is significantly different from that of the normal points. Therefore, the synthetic data sets are chiefly used for memory and execution time scaling experiments. In our experiments, we consider a synthetic data set containing a 1% mixture of outliers.

4.2 Detection Rate

Our first set of experiments compares the detection rates of RELOADED to both LOADED and ORCA. In particular we compare the detection rate of the two-pass version of RELOADED to that of the two-pass version of LOADED, and ORCA. We do not include results for ORCA on the KDDCup training data set, as it cannot complete in a reasonable amount of time.

Detection rates for all the different algorithms are reported in Table 1 (Note that "n/a" indicates that the attack was not present in that particular data set). Since the intrusion packets tend to occur in bursts in our data set, we mark an intrusion as detected if at least one instance in a burst is flagged as an outlier. This is realistic, since a network administrator needs to be alerted only once that an intrusion is underway. Consequently, the detection (true positive) rates in the table are in terms of the number of intrusion bursts detected. The highest detection

Attack	KDDCup Testing			KDDCup Training	
	RELOADED	LOADED	ORCA	RELOADED	LOADED
Apache2	**100%**	**100%**	0%	n/a	n/a
Back	n/a	n/a	n/a	0%	**98%**
Buffer Overflow	72%	90%	**100%**	0%	**91%**
FTP Write	n/a	n/a	n/a	0%	**33%**
Guess Password	50%	**100%**	0%	34%	**100%**
Imap	n/a	n/a	n/a	50%	**100%**
IP Sweep	**100%**	28%	0%	**90%**	37%
Land	**100%**	0%	0%	**100%**	**100%**
Load Module	n/a	n/a	n/a	0%	**100%**
Multihop	63%	70%	**75%**	0%	**94%**
Named	67%	**100%**	40%	n/a	n/a
Neptune	n/a	n/a	n/a	**100%**	98%
Nmap	n/a	n/a	n/a	64%	**91%**
Perl	n/a	n/a	n/a	0%	**100%**
Phf	80%	20%	**100%**	0%	0%
Pod	96%	**100%**	18%	**81%**	54%
Port Sweep	**100%**	**100%**	3%	93%	**100%**
Root Kit	n/a	n/a	n/a	0%	**33%**
Saint	**100%**	**100%**	1%	n/a	n/a
Satan	n/a	n/a	n/a	**80%**	72%
Sendmail	17%	**50%**	**50%**	n/a	n/a
Smurf	**98%**	21%	0%	**78%**	22%
Snmpgetattack	0%	**52%**	0%	0%	0%
Spy	n/a	n/a	n/a	0%	**100%**
Teardrop	n/a	n/a	n/a	**40%**	30%
Udpstorm	0%	0%	0%	n/a	n/a
Warez Client	n/a	n/a	n/a	4%	**43%**
Warez Master	n/a	n/a	n/a	0%	**25%**
Xlock	50%	50%	**66%**	n/a	n/a
Xsnoop	**100%**	**100%**	**100%**	n/a	n/a

Table 1. Detection rates for the KDDCup data sets. The best detection rates are in bold.

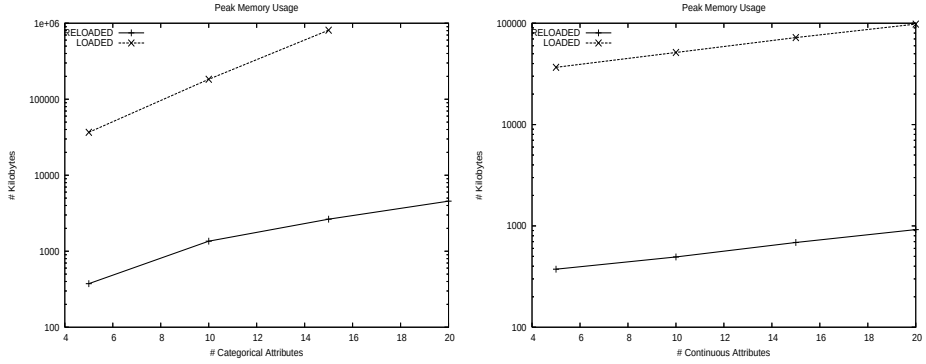

Figure 2. Peak memory usage with increasing numbers of (a) categorical and (b) continuous attributes.

Algorithm	KDDCup (Test)	KDDCup (Train)	Adult
RELOADED	623	852	291
LOADED	49,328	595,280	58,316
ORCA	599	n/a	390

Table 2. Peak heap usage in kilobytes.

rates for each intrusion type are in bold. We report false positive rates in terms of the number of normal packets marked as outliers. LOADED has a false positive rate of 0.35%, which is good considering its high detection rates for many of the intrusions. ORCA has a false positive rate of 0.43%, but this is not as significant considering its low detection rates. RELOADED has detection rates comparable to LOADED on many intrusions, and does very well on a handful of intrusions (e.g. IP sweep and smurf) on which both LOADED and ORCA do poorly. RELOADED has higher false positive rates of 1.5% for the testing data set and 3.6% for the training data set, which is to be expected since it builds a less intricate model in order to save on memory.

4.3 Memory Usage

We first compare the memory usage of RELOADED to that of both LOADED and ORCA when they are run on the KDDCup and Adult data sets. For RELOADED and LOADED we use single-pass approaches, as the amount of memory used does not vary with the number of passes. We set ORCA to find the top 1,314 outliers in the KDDCup testing data set and the top 30 outliers in the Adult data set. As mentioned before, ORCA does not finish in a reasonable amount of time on the KDDCup training data set. We measure memory usage by looking at the peak heap usage measured in kilobytes. The results can be seen in Table 2. Both RELOADED and ORCA consume less than one megabyte of memory, while LOADED uses *two to three orders of magnitude more memory*, even when we constrain the lattice to 4 levels. If we examine the cache performance on RELOADED when run on the KDDCup testing data set, we find that it has 0.0003 L2 cache misses per instruction, which is indicative of good temporal locality. This is due to the fact that RELOADED's model of the data is compact enough to fit in L2 cache.

Unlike ORCA, LOADED and RELOADED have greater than linear space complexity with respect to the number of categorical and continuous attributes, and so we empirically test how their peak memory usage scales as the number and types of attributes vary. In Figure 2(a) we plot peak memory usage versus the number of categorical attributes, while setting the number of continuous attributes equal to 5. It is evident that the memory requirements of LOADED are very large and grow exponentially as the number of categorical attributes increase, while those of RELOADED grow much more slowly. Note that we cannot run LOADED on data sets with more than 15 categorical attributes, as our machines do not have sufficient memory. In Figure 2(b) we set the number of categorical attributes equal to 5 and then vary the number of continuous attributes. The peak memory requirements of both RELOADED and LOADED increase at about the same rate, which is expected as they both use space that is quadratic in the number of continuous attributes. Note that even for 5 categorical attributes, LOADED requires significantly more memory to maintain the itemset lattice.

4.4 Execution Time

In our next set of experiments we compare the execution times of RELOADED, LOADED and ORCA. We use the single-pass versions of both RELOADED and LOADED. In our first experiment, we measured the execution times on the KDDCup testing data set. RELOADED takes 47 seconds to complete, compared to 109 seconds for LOADED and 303 seconds for ORCA. Next, we examine how execution time scales with the number of data points processed. We use synthetic data with 10 categorical and 5 continuous attributes. The results can be seen in Figure 3(a). For small data sets, ORCA out-performs both LOADED and RELOADED, but since it does not scale linearly, this advantage is lost for larger data sets. As we expect, the execution times of both RELOADED and LOADED scale linearly with the number of points. Note that LOADED does not scale as well as RELOADED, as it must compare each point with and update $\Sigma_{i=1}^{4}\binom{10}{i} = 385$ covariance matrices and itemsets, whereas RELOADED need only compare each point with and update 10 classifiers and covariance matrices.

While ORCA's time complexity is linear in both the number of categorical and continuous attributes, RELOADED's and LOADED's complexity is not, and so in our next two experiments we compare how the execution of times of both RELOADED and LOADED scale for data sets with varying numbers of categorical and continuous attributes. In our first experiment, we set the number of continuous attributes equal to 5 and vary the number of categorical attributes from 1 to 15. The results can be seen in Figure 3(b). Though we limit the size of LOADED's lattice, its execution time still increases exponentially with the number of categorical attributes, while that of RELOADED increases quadratically. In our second experiment we examine the scalability of both algorithms with respect to the number of continuous attributes. In this experiment we set the number of categorical attributes equal to 5 and vary the number of continuous attributes from 1 to 25. The results can be seen in Figure 3(c). For smaller numbers of continuous attributes, LOADED is more efficient than RELOADED, but RELOADED scales better for larger numbers of continuous attributes.

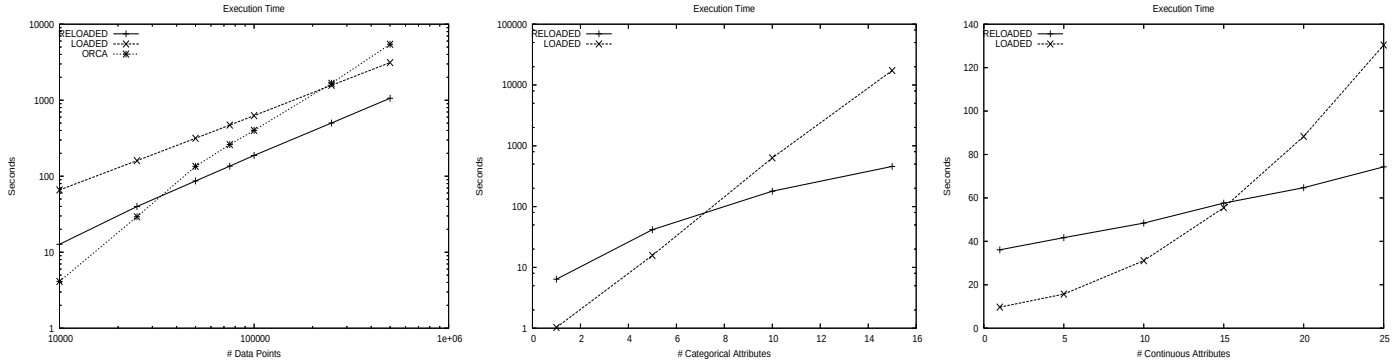

Figure 3. Plots of execution time versus (a) data set size; (b) increasing categorical attributes; (c) increasing continuous attributes.

5 Conclusions and Future Work

In this paper we have presented a general-purpose outlier detection algorithm named RELOADED that is capable of discovering outliers in dynamic mixed-attribute data. It is designed to minimize both memory consumption and execution time. Our experimental results show that for about the same memory usage, RELOADED outperforms ORCA in terms of speed and detection rates. While RELOADED does not outperform LOADED in terms of detection rates, it does in terms of memory usage and speed. RELOADED's low execution times and memory consumption make it a good candidate for embedded outlier detection systems, such as might be found in network interface card-based intrusion detection [17], or sensor networks. In future, we plan to examine alternatives to the Naïve Bayes Classifiers we currently use. Finally, we also plan to look at a hybrid RELOADED/LOADED approach to increase detection rates.

References

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

[2] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proc. of 9th annual ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[3] C. Blake and C. Merz. UCI machine learning repository, 1998.

[4] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying density-based local outliers. In *ACM SIGMOD Intl. Conf. Management of Data*, 2000.

[5] D. Gamberger, N. Lavrač, and C. Grošelj. Experiments with noise filtering in a medical domain. In *ICML*, 1999.

[6] A. Ghoting, M. E. Otey, and S. Parthasarathy. Loaded: Link-based outlier and anomaly detection in evolving data sets. In *Proceedings of the IEEE International Conference on Data Mining*, 2004.

[7] S. Guha, R. Rastogi, and K. Shim. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000.

[8] S. Hettich and S. Bay. KDDCUP 1999 dataset, UCI KDD archive, 1999.

[9] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[10] T. Johnson, I. Kwok, and R. T. Ng. Fast computation of 2d depth contours. In *ACM SIGKDD*, pages 224–228, 1998.

[11] E. Knorr and *et al*. Distance-based outliers: Algorithms and applications. *VLDB Journal*, 2000.

[12] E. Knorr and R. Ng. A unified notion of outliers: Properties and computation. In *ACM SIGKDD*, 1997.

[13] E. Knorr and R. Ng. Finding intentional knowledge of distance-based outliers. In *VLDB*, 1999.

[14] E. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proc. Int'l Conf. on VLDB*, 1998.

[15] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of outlier detection schemes for network intrusion detection. In *SIAM Data Mining*, 2003.

[16] M. V. Mahoney and P. K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *ACM SIGKDD*, 2002.

[17] M. Otey, S. Parthasarathy, A. Ghoting, G. Li, S. Narravula, and D. Panda. Towards nic-based intrusion detection. In *Proceedings of 9th annual ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[18] M. E. Otey, A. Ghoting, and S. Parthasarathy. Fast distributed outlier detection in mixed-attribute data sets. Technical Report OSU-CISRC-6/05-TR42, Department of Computer Science and Engineering, The Ohio State University, 2005.

[19] S. Papadimitriou, H. Kitawaga, P. B. Gibbons, and C. Faloutsos. LOCI: Fast outlier detection using the local correlation integral. In *ICDE*, 2003.

[20] K. Sequeira and M. Zaki. Admit: Anomaly-based data mining for intrusions. In *ACM SIGKDD 02*, pages 386–395, 2002.