LOADED: Link-based Outlier and Anomaly Detection in Evolving Datasets

Amol Ghoting, Matthew Eric Otey and Srinivasan Parthasarathy*

The Ohio State University
{ghoting, otey, srini}@cis.ohio-state.edu

Abstract

Detecting outliers or anomalies efficiently is an important problem in many areas of science, medicine and information technology. Applications range from data cleansing to clinical diagnosis, from detecting anomalous defects in materials to fraud and intrusion detection, etc. Most approaches to date have focused on detecting outliers in a continuous attribute space. However, almost all real-world datasets contain both continuous and categorical attributes. The categorical attributes are largely ignored by such approaches and often the information that is lost is quite significant. Although there has been a spate of work in outlier detection that draws inspiration from network intrusion detection, none of them consider the problem of outlier detection in both dynamic and distributed environments.

The challenge is to efficiently identify outliers in the presence of a variety of constraints including minimizing the time to respond, adapting to the data influx rate and managing the data when it is distributed across networks. To address this challenge, we present LOADED, a one-pass algorithm for outlier detection in evolving datasets containing both continuous and categorical attributes. Furthermore, LOADED is a tunable algorithm, wherein one can trade off computation for accuracy so that domain-specific (e.g. intrusion detection) response times are achieved. We then extend the proposed algorithm to a distributed grid setting where anomalies across a collection of dynamically evolving datasets can be detected. Experimental results validate the effectiveness of our schemes over several real datasets.

*This work is supported in part by an NSF CAREER grant (IIS-0347662) and an NSF ACR grant (ACI-0234273)

1 Introduction

Webster's Dictionary defines an outlier as "one which lies, or is, away from the main body." In the context of the current work this can be interpreted to mean that an outlier is a sample point or datum that is distinct from most, if not all, other points in the dataset. Recently, the problem of efficient outlier detection has attracted a lot of attention due to its wide applicability in areas such as fraud detection [4], intrusion detection [6, 27], data cleaning [25, 7], detecting and classifying anomalous defects in materials [14], and clinical diagnosis [24].

There are several approaches to outlier detection. In the statistics community, the focus has been on model-based outlier detection, where the assumption is that the data follows a parametric (typically univariate) distribution [1]. Such approaches do not work well even in moderately high dimensional spaces and finding the right model is often a difficult task in its own right. To overcome these limitations, researchers have taken to various non-parametric approaches including approaches based on computational geometry [15], clustering-based approaches [27], distance-based approaches [2, 16] and density-based approaches [23, 5]. However, there are several challenges that need to be addressed.

First, almost all the non-parametric approaches rely on some well defined notion of distance to measure the proximity between two data points. However, many real-world datasets contain both continuous and categorical attributes. Nominally valued categorical attributes have serious implications for a well-defined notion of distance. Simply disregarding categorical attributes in such scenarios may result in the loss of information important for effective outlier detection.

Second, an outlier detection scheme needs to be sensitive to response time constraints imposed by the domain. For instance, an application such as network intrusion detection requires on-the-fly outlier detection. Most existing distance-based approaches for outlier detection require multiple passes over the dataset which, if not impossible, are prohibitive in such a scenario.

Third, because of societal, market factors, or the in-

herent distributed nature of data production, the data to be analyzed is often spread across a network. For example, at the Ohio State University, network data is collected at seven distributed end-points (routers), for subsequent analysis. In such an environment, one has two choices: Either the data must be shipped to a centralized location and subsequently processed, or the data must be processed in a distributed manner. The former approach would be extremely inefficient, while the latter approach may lose out on some information.

In this paper, we present work addressing the above challenges. Specifically, we make the following contributions:

- We define a metric wherein one can effectively identify outliers in a categorical data space. Furthermore, we extend this metric such that the inherent dependence between continuous and categorical attributes can be captured in a uniform manner, instead of considering them separately.
- We present an algorithm, based on the above metric, named LOADED (Link-based Outlier and Anomaly Detection in Evolving Datasets). We also present an approximate scheme by which, this algorithm allows for more efficient outlier detection when response time is an issue.
- We extend our centralized outlier detection algorithms to operate in a distributed grid-like setting, which is of important practical concern to applications such as intrusion detection.
- We extensively evaluate our algorithms on several real datasets and confirm the utility of finding outliers in mixed attribute data. We also present results on the scalability of our centralized and distributed algorithms.

The remainder of this paper is organized as follows. We first examine related work in Section 2. In Section 3, we present our outlier detection schemes for both centralized and distributed settings. We present our experimental results in Section 4. Finally, we conclude with directions for future work in Section 5.

2 Related Work and Background

As mentioned in the previous section, statistical model-based outlier detection methods do not scale well with the number of dimensions of the data, and finding an appropriate parametric model for the data is a difficult task in itself. Simplified probabilistic models suffer from a high false positive detection rate [20, 22]. Also, methods based on computational geometry do not scale well with the number of dimensions. Here we consider other methods in more detail.

An approach for discovering outliers using distance metrics was first presented by Knorr *et al.* [16, 17, 18]. They define a point to be a *distance outlier* if at least a user-defined fraction of the points in the dataset are further than some user-defined minimum distance threshold. They primarily focused on datasets containing only real valued attributes in their experiments.

Related to the above distance-based methods are methods that cluster data and find outliers as part of the process of clustering [13]. Points that do not cluster well are labeled outliers. This is the approach used by the ADMIT intrusion detection system [27]. A clear limitation of clustering-based approaches to outlier detection is that they require multiple passes to process the dataset.

Recently, density-based approaches to outlier detection have been proposed [5]. 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) was presented in [23]. LOCI addresses the difficulty of choosing values for MinPts in the LOF technique by using statistical values derived from the data itself.

Most methods for detecting outliers take time that is at least quadratic in the number of points in the data set, often more, which may be unacceptable if the data set is very large and dynamic. Bay [2] presents a method for discovering outliers in near linear time. In the worst case, the algorithm still takes quadratic time, but in practice the algorithm runs very close to linear time.

A comparison of various anomaly detection schemes is presented in [19]. Its focus is on how well these 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 [11]. 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.

As far as we are aware, none of the above approaches explicitly or implicitly handle categorical attributes. In the context of clustering, ROCK [9], and K-prototypes[12], are two clustering algorithms that handle categorical attributes and thus one can potentially use them for outlier detection. However, ROCK handles categorical attributes only and both approaches require multiple passes to cluster data. One other shortcoming of K-prototypes is that it does not account for dependencies that span the categorical and continuous attributes.

Name	Type
Duration	Continuous
Source Bytes	Continuous
Destination Bytes	Continuous
Protocol Type	Categorical: HTTP, FTP, Telnet, etc.
Service	Categorical: TCP, UDP, etc.
Flag	Categorical: Connection Error Status
Land	Categorical: 1 (if the connection is to the same host/port), 0 (otherwise)

Table 1: Sample Attributes for a Network Transaction

2.1 Importance of Modeling Categorical Attributes

We illustrate the importance of modeling categorical attributes in the anomaly detection process using a toy example drawn from intrusion detection (see Table 1). We need to determine if a network transaction is an outlier or not. Consider the following instances of network transactions, where we know that *Instance B* is a normal transaction. *Instance A: (10 sec, 30 bytes, 40 bytes, HTTP, UDP, 0, 0)* and *Instance B: (11 sec, 30 bytes, 40 bytes, Telnet, TCP, 0, 0).* Using a distance-based approach, we use the Euclidean distance metric to measure the similarity between *Instance A* and *Instance B*. However, here if we were limited to using just the continuous attributes: *Duration, Source Bytes and Destination Bytes*; in our calculations, we would see a high similarity. However, the two transactions are very different as they are using different protocols which is represented in a categorical attribute. Moreover, *Instance A* should be flagged as an outlier since, a network transaction with the *HTTP* protocol should always use *TCP* as its service.

2.2 Key Challenges

There are several challenges to overcome. First, how does one model the categorical attributes in the outlier detection process? Put another way, what is a good distance metric in this space? One approach could be to arbitrarily assign numeric values for each nominal categorical value and ordinal categorical attributes can be assigned numbers according to their ordinality. Distance-based schemes can then treat these assigned values to be continuous. Such an approach has the drawback that distances between points projected onto this new space are meaningless (especially when projected to the space comprising categorical attributes). This problem escalates, especially if there are many categorical attributes, and can lead to poor detection and high false positive rates [19]. To address this problem, we develop a link-based approach to model similarity between data points.

The second question is how to aggregate and correlate distances captured between categorical and continuous attributes? Such correlations may be extremely important to model anomalous activity. An approach could be to find distances in the two dataspaces (continuous and categorical) separately and then combine them in some manner as has been proposed by others [12]. However, such an approach intuitively seems ad-hoc and is unlikely to capture the dependencies across the two spaces which may be essential. We address this issue by maintaining correlation matrices for the continuous attributes for different combinations of categorical attributes.

Most existing approaches for outlier detection need multiple passes over the entire dataset, since they need to capture global distances between each pair of points in the dataset. To facilitate online discovery, one needs to be able to efficiently summarize essential parts of the traffic. More importantly, how does one summarize essential parts of evolving data points in a single pass for categorical data? A categorical data record can be represented as a sequence of $(attribute, value)$ pairs. We will show that maintaining a summary of categorical data reduces to the problem of intelligently maintaining frequency counts for a small subset of the power set of feasible $(attribute, value)$ combinations in a single pass. The primary advantage of such an approach is that it allows us to capture the global neighborhood of a point in a categorical data space in a single pass, thus resulting in precise outlier detection.

3 Algorithms

3.1 Link-based Outlier Detection

We would like to formally capture the notion that a point shares a *link* with another point if the two points are considerably similar to each other. Let $p_i =< (attribute_1, value_1), \ldots, (attribute_N, value_N) >$ be a point with N categorical attributes that is represented as a set of $(attribute, value)$ pairs. We define $sim\,(p_i, p_j)$ to be a similarity function that captures the degree of similarity between a pair of points p_i and p_j. For some threshold θ, a pair of points share a link if:

$$sim\,(p_i, p_j) \geq \theta$$

Here the threshold θ is a user-defined parameter. $\theta = 1$ implies that the point will share a link with only identical points in the dataset, while $\theta = 0$ implies that the point will share a link with every point in the dataset.

Let us now define the similarity function to be the normalized number of $(attribute, value)$ pairs that the two points have in common. Thus,

$$sim\,(p_i, p_j) = \frac{\sum_{d=1}^{N} (p_i[value_d] \wedge p_j[value_d])}{N}$$

where $\wedge$ returns 1 if $p_i[value_d]$ equals $p_j[value_d]$ and 0 otherwise.

In order to estimate an anomaly score for a point, one needs to find how many other points it links to. Moreover, every link shared with another point is not the same. For instance, two points p_i and p_j may be very similar as defined by the above function, while p_i may be linked to another point p_k with similarity

value just above the threshold θ. Clearly there is a need to modulate the strength of the link and account for it. The *link strength* between p_i and p_j is simply defined as the number of $(attribute, value)$ pairs that are common to the two points, which is same as $N \times sim(p_i, p_j)$.

We define a score function that captures the degree to which a point is an outlier based on links and link-strengths as follows: Let $\ell_i(j)$ be defined as the number of points that link to a point p_i with a link strength of at least j.

$$Score_1(p_i) = \sum_{j=1}^{N} \left(\frac{1}{j} : \ell_i(j) \leq \alpha \right)$$

where α is a user defined threshold. This score function has the following characteristics:

- A point with no links to other points will have the highest possible score.
- A point that shares only a few links, each with a low link strength, will have a high score.
- A point that shares only a few links, some with a high link strength, will have a moderately high score.
- A point that shares several links but each with a low link strength, will have a moderately high score.
- Every other point will have a low to moderate score.

We note that $\ell_i(j)$ has the downward closure property, a property that facilitates efficient computation.

Lemma 1 *If $\ell_i(k) > \alpha$, then $\ell_i(j) > \alpha$ for $\forall j < k$.*

3.2 Relationship to Frequent Itemset Mining

Let $F = \{i_1, i_2, \cdots, i_I\}$ be the set of items, of cardinality I, wherein each item uniquely maps to the set of all feasible $(attribute, value)$ pairs present in the dataset. Let $S = \{s : s \in PowerSet(F) \wedge \forall_{i,j:i \neq j} s_i(attribute) \neq s_j(attribute)\}$. Set S is a set of *itemsets* such that every item within an itemset has a distinct *attribute*. The maximal itemset in S will thus be bounded by N, the total number of categorical attributes in the dataset.

In order to estimate a score for a point p_i, we need to find $\ell_i(j) \forall j \leq N$. Finding $\ell_i(j)$ reduces to the problem of finding the sum of frequency counts for all itemsets that have j $(attribute, value)$ pairs in common with p_i. Let $S_i(j) = \{s : s \in S \wedge |s| = j \wedge s \in p_i\}$. $S_i(j)$ is the set of all possible subsets of S with a size of j, with exactly j $(attribute, value)$ pairs in common with p_i. Let $freq(s)$ be the frequency of occurrence of itemset s in the dataset. The score estimation function that uses itemset frequency counts is defined as follows.

$$Score_2(p_i) = \sum_{j=1}^{N} \left(\frac{1}{j} : \left(\sum_{s \in S_i(j)} freq(s) \right) \leq \alpha \right)$$

Thus, the score estimation problem reduces to that of finding the frequency counts for all itemsets that have at least $N, N-1, \cdots, 1$ $(attribute, value)$ pairs in common with point p_i. We can further simplify our score function as follows.

$$Score_3(p_i) = \sum_{j=1}^{N} \sum_{s \in S_i(j)} \left(\frac{1}{j} : freq(s) \leq \beta \right)$$

where β is a user defined threshold. Although the score function $Score_3$ is a simplified version of $Score_2$, a suitable value for $\beta < \alpha$ gives us more fine grained control over the scoring process. Also, $freq(s)$ has the downward closure property.

Lemma 2 *If $freq(s) > \beta$ then $freq(s') > \beta$ $\forall s'$ that is a subset of s.*

3.3 Dealing with Mixed Attributes

Thus far, we have presented an approach to estimate a score for a point in a categorical data space. As motivated previously, we would like to capture the dependencies between the continuous and categorical attributes in a natural manner. If the expected dependencies between categorical and continuous data are violated by a point, then it is most likely an outlier. We choose a unified approach to capture this dependence. We incrementally maintain a correlation matrix [10] that stores the Pearson's Correlation Coefficient between every pair of continuous attributes. This matrix is maintained on a per itemset basis in S. This is clearly unrealistic in terms of memory requirements for high dimensional datasets and we relax this requirement in Section 3.5. Furthermore, we store a discretized representation of the true correlation coefficient such that each value is stored in a byte. These correlation coefficients capture dependencies between every pair of continuous attributes and continuous attributes that violate these dependencies will contribute towards the anomaly score. Moreover, as we are maintaining this matrix for each itemset, we are inherently capturing the dependence between the values in the categorical and continuous data spaces.

A point is defined to be linked to another point in the mixed data space if they are linked together in the categorical data space and if their continuous attributes adhere to the joint distribution as indicated by the respective correlation coefficients. Points that violate these conditions are defined to be outliers. We modify our score function for mixed attribute data as follows.

$$Score_4(p_i) = \sum_{j=1}^{N} \sum_{s \in S_i(j)} \left(\frac{1}{j} : (C1 \vee C2) \wedge C3 \text{ is true} \right)$$

where $C1$: $freq(s) \leq \beta$, $C2$: at least $\delta\%$ of the correlation coefficients disagree with the distribution followed by the continuous attributes for point p_i, and $C3$: $C1$ or $C2$ hold true for every superset of s in p_i. Condition $C3$ allows for more efficient processing because, when an itemset does not satisfy conditions $C1$ and $C2$, none of its subsets will be considered. The rationale for $C3$ is as follows. First, as stated by the Apriori property in lemma 2, subsets of frequent itemsets will also be frequent and thus, will not satisfy $C1$. Second, intuitively, correlation coefficients tend to be less discriminatory as the size of the itemset is reduced and thus, are less likely to satisfy $C2$.

3.4 The LOADED Algorithm

LOADED finds outliers based on the score estimation function $Score_4$. It also provides extensions to operate over dynamic data and supports intelligent itemset lattice management.

Extensions for Dynamic Data: As motivated previously, several applications (e.g. network intrusion detection) demand outlier detection over dynamic data. In order to capture concept drift and other dynamics of the data, we need to update our model to capture the most recent trends in the data. We achieve this by introducing a bias towards recent data. We maintain the itemsets together with their frequency counts in a hash table and the bias is introduced in the following two ways. First, we maintain an itemset together with its frequency count in the hash table only if it appears at least once every W points. Second, the frequency for every itemset in the hash table will be decremented by a value Δf every W points. We apply these biases using smart hash-table management coupled with a down-counting approach described in Figure 1.

For every W points, we create a new hash table with index i div W and delete the oldest hash table with index (i div $W-2$). Thus, at every instant in time, we maintain two hash tables at the most. We estimate the frequency for an itemset based on the its frequency in the two hash tables. Moreover, for every W points, relevant itemsets will have their frequencies biased with a value $-\Delta f$. The oldest hash table is deleted every W points and the two most recent hash tables will contain all the relevant itemsets with their biased frequencies.

Score estimation using LOADED: The LOADED algorithm for outlier detection is presented in Figure 2. The score function $Score_4$ requires that we iterate over all possible subsets of the maximal itemset for the point p. One approach to maintaining frequency counts for these subsets would be use the Lossy Counting approach proposed by Manku and Motwani [21].

Hash Table Management:
Let i be the index of the point that just arrived
If i mod $W = 0$,
  Then create a new hash table with index i div W
  delete the hash table with index i div $W-2$

Frequency Estimation:
When estimating the frequency for an itemset from a point with index i:
If an itemset for the i^{th} point is found in hash table with index i div W
  Increment and use the freq. from this hash table
Else
  If itemset is found in hash table with index i div $W-1$
    insert it into the hash table with index i div W,
    set freq = freq in table (i div $W-1$) $+1-\Delta f$
    and then use this freq.
  Else
    insert it into the hash table with index i div W
    with freq. = 1

Figure 1: Algorithm for hash table management and frequency estimation

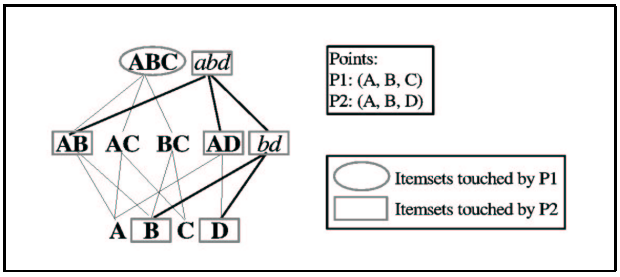

Figure 3: Lattice management in LOADED

Using this approach, we would need to update frequency counts for each possible subset on a per-point basis, which is an expensive operation. Another approach to the problem would involve using a disk-based approach [8]. Using this approach, everything below the negative border in the lattice will need to be maintained in memory and frequency counts for the remaining infrequent itemsets will be retrieved from the disk as and when needed, which too is an expensive operation. One of the ways to make the algorithm efficient is to minimize the number of itemsets in the itemset lattice that we must examine when a new point arrives. Lemma 2 states the Apriori property that if an itemset is frequent, then all of its subsets must also be frequent. An anomaly score for a point is calculated based on the number of infrequent itemsets that it contains (as per score function $Score_4$). Therefore, our algorithm need only examine those subsets of frequent itemsets that do not satisfy condition $C2$. Our algorithm works as follows.

1. For each point, we enumerate all possible itemsets with a size of $MAXLEVEL = N$ into a set G.
2. For each of these itemsets $g \in G$, we increment its frequency count in the lattice.

Procedure: Find Outliers
Input: *FrequencyThreshold, CorrelationThreshold, ΔScore, ScoreWindowSize, MAXLEVEL*
Output: Outliers discovered in one pass
For each incoming point p:
G = Enumeration of all itemsets of size $MAXLEVEL$ for point p
For each $g \in G$
Get frequency count $freq$ for g from the hash table
Update frequency and correlation coefficients for g in the hash table
If $freq <$ *FrequencyThreshold*
$score = score + \frac{1}{itemsetsize(g)}$
Add all subsets of g of size $itemsetsize(g) - 1$ into G
If these subsets of g do not have an entry in the hash table
Get precise a frequency count from $MAXLEVEL$ itemsets and insert into hash table
Else
If number of continuous attribute pairs that disagree with the correlation coefficients > *CorrelationThreshold*
$score = score + \frac{1}{itemsetsize(g)}$
Add all subsets of g of size $itemsetsize(g) - 1$ into G
If these subsets of g do not have an entry in the hash table
Get precise a frequency count from $MAXLEVEL$ itemsets and insert into hash table
End For
If *score* > (Avg. score in ScoreWindow × ΔScore)
Flag as Outlier
Else
Normal
Update the Score window of size *ScoreWindowSize* if the point is normal
End For
end

Figure 2: The LOADED Algorithm for centralized outlier detection

3. We check if g's frequency is less than *FrequencyThreshold*. If so, we increase the score by a value inversely proportional to the size of the itemset, as is dictated by the score function. Furthermore, we find all subsets of g of size $itemsetsize(g) - 1$ and insert these into G. If g became infrequent recently, there is a possibility that some subsets of g may not have an entry in the hash table. We use the following lemma to precisely estimate the frequency count for any subset of g.

   Lemma 3 *The frequency count for any itemset $g \in G$ can be precisely estimated using frequency counts from all $MAXLEVEL$ itemsets in G.*

   In this way, in the worst case, our algorithm examines the itemsets between $MAXLEVEL$ and the positive border of the lattice. The following example (Figure 3) illustrates step 3 of our algorithm. Note that in Figure 3, frequent itemsets are labeled using uppercase characters, while infrequent itemsets are labeled using lowercase characters. Consider the following points:P_1 - ABC, P_2 - ABD. The three-itemset corresponding to P_1 is frequent, so we only need to increment the frequency count of itemset ABC in the lattice. As for P_2, ABD is not frequent. Therefore, we increment the frequency count of ABD and examine its subsets AB, AD, and BD. AB and AD are frequent, so we only have to increment their respective frequency counts. Note that the frequency counts of these sets can be derived from the frequency counts of their respective supersets. However, itemset BD is not frequent. As before, we increment the frequency count of item BD and examine its subsets, B and D. B and D are both frequent, so we just increment their respective frequency counts and move on to the next point. Note that in our algorithm, we do not have to examine all of the subsets of itemset ABD.

4. If the itemset g is frequent, we check that the continuous attributes of the point are in agreement with the distribution determined by the correlation matrix using the *CorrelationThreshold* (Estimating the level of agreement is a well studied problem in statistics [26]). If so, we will not increase the score. Otherwise, we increase the score, again by a value inversely proportional to the size of the itemset. Furthermore, we find all subsets of g of size $itemsetsize(g) - 1$ and insert these into G.

5. If G is not empty go to step 2, else continue.

6. We maintain all reported scores over a recent window of size *ScoreWindowSize* and calculate the average score in the window. If the total score for

the point is greater than $\Delta Score \times$ average score, we flag the point as an outlier.

For an application such as network intrusion detection, providing an interactive response time is crucial. This enforces a one-pass processing requirement over the dataset and the score is estimated using itemset frequency counts maintained over a recent window of time. On the other hand, an application may need a higher detection accuracy, arguing for a two-pass approach. In the first pass, we find itemset frequency counts over the entire dataset and in the second pass, we find outliers based on what we have learned. The two-pass approach is beneficial as points that were mistaken as outliers during the first pass (due to a lack of information about the entire dataset) will not be flagged as outliers in the second pass.

3.5 Approximation Scheme

As it stands, our algorithm maintains and examines the itemset lattice beginning at $MAXLEVEL = N$ down to the positive border, and for each itemset in that lattice, there is a correlation matrix whose size is proportional to the the square of the number of continuous attributes. Maintaining this lattice and the corresponding matrices can consume a large amount of memory, even with the optimizations considered in LOADED. . Furthermore, the time it takes to check if a point is an outlier is proportional to the number of lattice levels between N and the positive border. For some applications, the amount of memory used by our algorithm or the time taken to determine if a point is an outlier may be too costly. In this case, it would be better to use a faster and more memory-efficient algorithm that approximates the anomaly score.

Given two 1-itemsets I_1 and I_2, the frequency for the 2-itemset $I_1 \wedge I_2$ is bounded by the smaller of the frequencies for I_1 and I_2. As a result, the frequencies of the lower order itemsets serve as good approximations for the frequencies of the higher order itemsets. We consider approximations by maintaining itemsets only up to a certain level $MAXLEVEL$. An approximate scheme such as this will provide improved execution time as a smaller subset of the lattice needs to be considered. However, the accuracy of the algorithm will decrease. In the general case, accuracy improves as $MAXLEVEL$ increases. We empirically validate this in the next section.

3.6 Distributed Algorithm

As motivated previously, several outlier detection applications need to process datasets as they are being produced, in a distributed setting. For example, in network intrusion detection, the traffic coming in to a network may be collected at several different edge routers. Because of the high volume of data passing through the routers, it is impractical to send it all to a centralized site for processing. An alternative would be for each router to detect outliers on their own data streams and then exchange messages with the other routers to confirm the global set of outliers.

To handle such cases, we present a distributed version of LOADED in Figure 4. Each node runs a the centralized LOADED algorithm, and the parameters are set uniformly across all nodes. The nodes only communicate when some user-specified event occurs. Examples of such events include: A user's query for the global outliers, a node completing the processing of a fixed number of points, or a node finding a fixed number of outliers. When such an event occurs, each node broadcasts all outliers it has found since the last event to all other nodes. Upon receiving a broadcast, a node will check all points in the broadcast to see if they are flagged as outliers locally, and will return that information to the requesting node. If, for a given point, all nodes respond that it is a local outlier, then it is flagged as a global outlier.

Note that during the validation process, every other computing node incorporates the points into their models. This fact, coupled with the fact that each potential global outlier at a node must be validated against the information maintained by all other nodes, means that the set of global outliers we discover using this method is the same as if we had found them using the centralized version.

There are trade-offs to consider when choosing which event to use. For example, since outliers are, by definition, rare, the simplest event to use that a node requests validation each time it discovers a local outlier. However, this approach requires a synchronization between nodes for any outlier found at a local node. As an alternative, we can choose an event where a node accumulates n potential local outliers before broadcasting them for validation. In this case, there is less synchronization between the nodes, but there is a greater communication cost. This increase in communication cost stems from the fact that the nodes are receiving information about global outliers less often, which increase the number of local outliers that need to be broadcast at each event.

Since we are using absolute thresholds, the following lemmas (trivial to prove), validate the correctness of our distributed algorithm and guarantee to detect all outliers detected by the sequential version of the algorithm.

Lemma 4 *Every global outlier is a local outlier at all the nodes.*

Lemma 5 *Every global normal point is a local normal point at atleast one node.*

3.7 Complexity Analysis

For a feature space with N categorical attributes and D continuous attributes, let M be the maximum number of distinct values taken by any categorical attribute. Let $MINLEVEL$ be the size of

Distributed Outlier Detection Algorithm

For each incoming point
  Determine if the point is an outlier using the centralized algorithm locally
  If it is found to be an outlier,
    Add it to the set of outliers to be verified at the next event
  If an event occurs,
    Broadcast the set of potential outliers to all other participating nodes
    Receive normal/outlier labels from all the nodes
    If all other nodes also flag a point as an outlier,
      Flag the point to be a global outlier
  If another node needs outlier verification,
    Receive the points, process them and return outlier/normal labels to that node
End For

Figure 4: Distributed algorithm for outlier detection

smallest itemset in the positive border of the lattice. In the worst case, every itemset between levels $MAXLEVEL$ and $MINLEVEL$ will be examined. For each level i in the lattice, we have $\binom{N}{i}$ feasible i sets, each set can be constructed in M^i ways. For a data set of size n,

$$\textit{Total execution time} \leq n \times D^2 \times \sum_{i=MINLEVEL}^{MAXLEVEL} M^i \binom{N}{i}$$

$$= O\left(nD^2 (MN)^{MAXLEVEL-MINLEVEL+1}\right)$$

Thus, execution times scales linearly with n, quadratically with D and exponentially with $MAXLEVEL - MINLEVEL + 1$.

4 Experimental Results

In this section, we examine LOADED over several real datasets from the standpoints of quality and performance. Specifically, we evaluate:

- The nature of the outliers discovered
- The benefits of using our approximation schemes
- The speedup gained when mining for outliers in a distributed setting.

4.1 Setup

To test LOADED[1], we selected the following real datasets. These datasets were collected from a range of domains and have a diverse set of features.

- *KDDCup 1999:* The 1999 KDDCup data contains a set of records that represent connections to a military computer network where there have been multiple intrusions and attacks by unauthorized users. The raw binary TCP data from the network has been processed into features such as *connection duration, protocol type, number of failed logins* and so forth. This dataset was obtained from the UCI KDD archive [11]. The training dataset has 4,898,430 data instances with 34 continuous attributes and 7 categorical attributes. The testing dataset is smaller and contains several new intrusions that were not present in the training dataset.

- *Adult Database:* This dataset was extracted from the US Census Bureau's Income dataset. Each record has features that characterize an individual's yearly income. Also, each record has a class label indicating whether the person made more or less than 50,000 dollars per year. This dataset was obtained from the UCI Machine Learning Repository [3]. The dataset has 48,842 data instances with 6 continuous attributes and 8 categorical attributes.

- *Congressional Votes:* This dataset is the the US Congressional voting records in 1984. Each record corresponds to one Congressional Representative's votes on 16 issues. All attributes are boolean (Yes/No), with a few missing values. In this dataset, missing values can correspond to such cases as voting present or abstaining from voting, in addition to having no record of the vote at all. A classification label of Republican or Democrat is provided with each data record. The dataset contains records for 168 Republicans and 267 Democrats and was obtained from the UCI Machine Learning Repository [3].

When analyzing LOADED's performance in a centralized setting, we ran our experiments on a Pentium III processor (1GHz and 512MB RAM) running Linux kernel 2.4. For evaluating the distributed version of LOADED, in a grid-like setting, we used machines from four different sites, two housed in the Computer Science Department but on separate subnets, one housed at the Ohio Supercomputing Center and one housed in the Department of Biomedical Informatics. The processor specifications are similar to the

[1]For all experiments unless otherwise noted, we ran LOADED with the following parameter settings: *FrequencyThreshold = 10, CorrelationThreshold = 0.3, ΔScore = 10, ScoreWindowSize = 40.*

Attack Type	Detection Rate (Training)	Detection Rate (Testing)
Apache 2	n/a	100%
Buffer Overflow	91%	87%
Back	97%	n/a
FTP Write	33%	n/a
Guess Password	100%	100%
Imap	100%	n/a
IP Sweep	37%	33%
Land	100%	n/a
Load Module	100%	n/a
Multihop	94%	100%
Named	n/a	100%
Neptune	98%	n/a
Nmap	91%	n/a
Perl	100%	n/a
Phf	0%	20%
Pod	53%	100%
Port Sweep	100%	100%
Root Kit	33%	n/a
Satan	72%	n/a
Saint	n/a	100%
Sendmail	n/a	50%
Smurf	22%	21%
Snmpgetattack	n/a	52%
Spy	100%	n/a
Teardrop	49%	n/a
Udpstorm	n/a	0%
Warez Client	43%	n/a
Warez Master	0%	n/a
Xlock	n/a	50%
Xsnoop	n/a	100%

Table 2: Evaluation of Single Pass LOADED on KDDCup Data

above. The implementation of all algorithms was in C++ and we used sockets for communication purposes in the distributed experiments.

4.2 Nature of the Discovered Outliers

KDDCup 1999: As explained previously, LOADED flags data points as outliers by taking into account frequency counts for various combinations of $(attribute, value)$ pairs. A fair evaluation for our approach would require outliers to constitute between (1-3%) of the dataset. However, the KDDCup dataset has a larger fraction of outliers (intrusions). We filtered out all the anomalous records from the dataset and reinserted them into the dataset in a semi-random fashion so that the total number of anomalous records was equal to 2% of the entire dataset. The semi-randomness stems from the fact that we maintained the same proportion for different types of intrusions in the original dataset and we also insured that attacks that happen in blocks (e.g. DOS attacks) will occur in blocks in the new dataset.

Our results on both the training and test datasets are reported in Table 2. Overall the results are quite good for a majority of the attacks. The attacks in the training dataset for which we did poorly are *Phf, Warez Master, FTP Write, IP Sweep, Smurf and Rootkit. Phf* had exactly one instantiation in this dataset. *Warez Master and FTP Write* are quite similar to normal ftp traffic so were not flagged by our algorithm. The remaining intrusions for which we do poorly are detected in a second pass of the algorithm. The rationale for this behavior can be explained by the distribution of their arrivals. For instance, *smurf* attacks, which tend to be grouped attacks (DOS), arrive in the first part of the dataset. In our one pass algorithm the first few *smurf* attacks were flagged but the later ones were not since at that point in time only a small part of the dataset had been read and the percentage contribution of smurfs were quite high. A similar rationale extends to the other intrusions with low detection rates. On the testing dataset the good news is that our approach was able to detect most of the unknown attacks (a problem for almost all the KDDCUP 1999 participants). Similar results were observed here when running the two pass algorithm.

Note that our approach is a general anomaly detection scheme and has not been trained to catch specific intrusions. Once an anomaly has been detected, a network administrator can perform a more detailed investigation and develop a signature for this attack, thus facilitating signature-based intrusion detection in conjunction with our approach. All the intrusions reported in the above table were caught in a single pass, since making two passes of the data is infeasible in the domain of network monitoring.

Our results are even more impressive when we examine the false positive rate (which we compute as number of normal records flagged as outliers divided by the total number of records in the dataset). *Our approach gives us a false positive rate of 0.35%, which is extremely good for anomaly detection schemes especially considering our high detection rates.* This can be attributed to a better representation for anomalous behavior obtained by capturing the dependence between categorical attributes and their associated continuous attributes.

Adult Database: This dataset was processed in two passes by our algorithm. In the first pass, LOADED learns the model and in the second pass, it flags the outliers. We did not process this dataset in a single pass since this is a static dataset that does not warrant on-the-fly outlier detection. Learning the model in the first pass gives us better global information for the second pass. The following data records were marked as the top outliers:

- A 39 year old self-employed male with a doctorate, working in a Clerical position making less than 50,000 dollars per year
- A 42 year old self-employed male from Iran, with a bachelors degree, working in an Executive position making less than 50,000 dollars per year
- A 38 year old Canadian female with an Asian Pacific Islander origin working for the US Federal Government for 57 hrs per week and making more than 50,000 dollars per year

Apart from these, the other outliers we found tended to come from persons from foreign countries that are not well represented in this dataset.

Congressional Votes: As in the previous case, the limited size of the dataset warranted a two pass approach. Many of the top outliers contained large numbers of missing values, though there are examples in which this is not the case. One such example is a Republican congressman, who has four missing values, but also voted significantly differently from his party on four other bills. According to the conditional probability tables supplied with the dataset, the likelihood of a Republican voting in a such a manner is very low.

4.3 Benefits of Approximation Scheme

To demonstrate the benefits of our approximation scheme, we again considered the KDDCup, Adult, and Congressional Votes datasets. In our experiments, we measured the execution time, detection rate and the false positive rate as a function of the number of itemset lattice levels maintained. For execution time, in the case of one pass algorithms (for KDD Cup) we measured the execution time per 1000 tuples processed, and in the case of two pass algorithms we report the overall execution time. For the Adult and Congressional Votes datasets, we define the detection rate as the number of outliers found using only the sub-lattice of itemsets compared to those found using the complete lattice. The primary benefit of our approximation scheme is that far better execution times can be achieved if we maintain fewer lattice levels, as can be seen in Figure 5. However, our false positive rates increase and detection rates decrease as the number of lattice levels decrease, as can be seen in Figure 6.

From the figures, one can see that number of lattice levels has a greater affect on the detection rate in the case of the KDDCup dataset than in the other datasets. This can be attributed to larger categorical attribute dependencies being used in the detection process for the KDDCup dataset. The other datasets can work reasonably well with smaller lattices. The number of lattice levels maintained do not affect the false positive rate as much as they affect the detection rate. For both the false positive rate and the detection rate, the more information that is used (i.e. the more lattice levels that are maintained), the less likely the system is to make a mistake. We would also like to note that the processing rate per 1000 network transactions is within reasonable response time rates even for a lattice level equal to four.

Ideally, one would like to tune the application based on the learning curves in an application specific manner, as each application will exhibit a specific dependence behavior. For example, in network intrusion detection, execution time must be minimized to keep pace with the network traffic, and since only one packet of an attack needs to be flagged in order to alert a monitor that an attack is in progress, using fewer lattice levels may be advisable. However, in other domains this might not be the case. As is evident from the figures, execution time grows near-linearly with the number of lattice levels.

4.4 Speedup in a Distributed Setting

First, we explored the speedups obtained when running the distributed version of LOADED on two, three and four sites. The KDDCup and Adult datasets were split evenly between the nodes. Furthermore, for the KDDCup data, we considered the following cases: a) Intrusions that are uniformly distributed across all sites; and b) Intrusions that are distributed across sites according to the attack class. The Congressional Votes dataset is too small to consider for these experiments. Figure 7 shows speedups obtained on the KDDCup and Adult datasets. As there are relatively few outliers in the datasets and we have a very low false positive rate, there is very little communication overhead overall. Moreover, there is less synchronization between the nodes and each can work independently. As the number of nodes increase, the communication overhead increases, since there are more nodes exchanging messages. As a result, there is a slight reduction in speedup and efficiency as the number of nodes increases. We see an efficiency in the range of (95 - 99%) for this reason. When intrusions are distributed by attack type, we see a slightly reduced efficiency. When intrusions are uniformly distributed by attack type, outliers detected at other computing nodes aid in faster model convergence. On the other hand, when intrusions are distributed by attack type, more messages need to be exchanged to achieve the same model convergence, resulting in a slightly reduced speedup. The Adult dataset has a slightly lower efficiency because it was processed in two passes. During the first pass, a relatively large number of outliers are generated, which requires more communication between nodes and hence results in more overhead.

Furthermore, we varied the average link latencies between the nodes in this controlled environment in order to simulate a wide area network spanning larger distances. As shown in Figure 7, efficiency falls to only 90% in a wide area setting with an average link latency of 25ms. This is representative of networks spanning across several states (for instance, we have a 25ms average latency between a machine located at the Ohio State University and a machine located at the University of Rochester). Figure 8 shows the per 1000 transaction response time delivered in a wide area network setting as we varied latency. Even for a high link latency of 25ms, the response time increases by only a small fraction. This is indicative of excellent scalability.

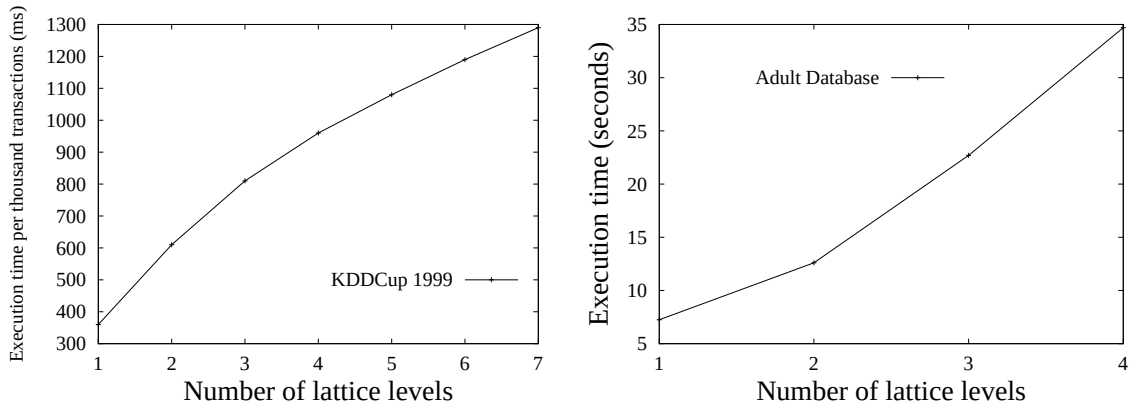

Figure 5: Execution time variation with increase in lattice levels

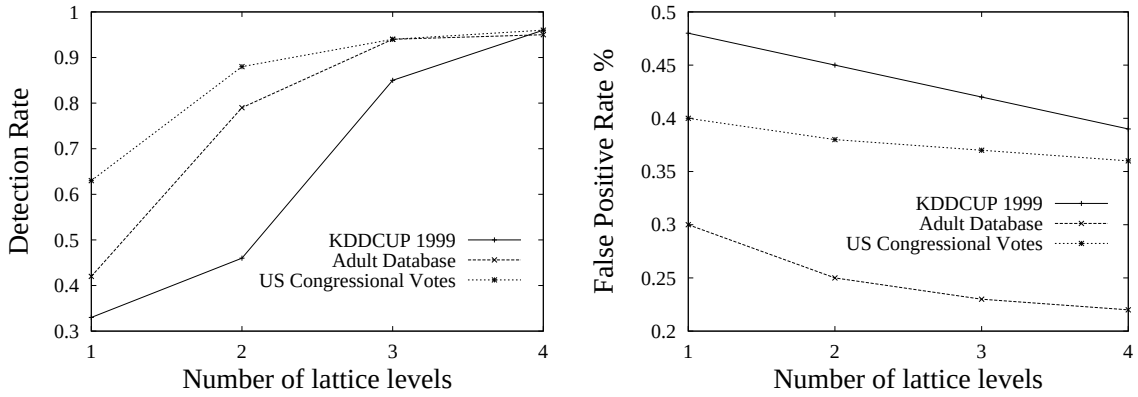

Figure 6: Detection and False positive rates

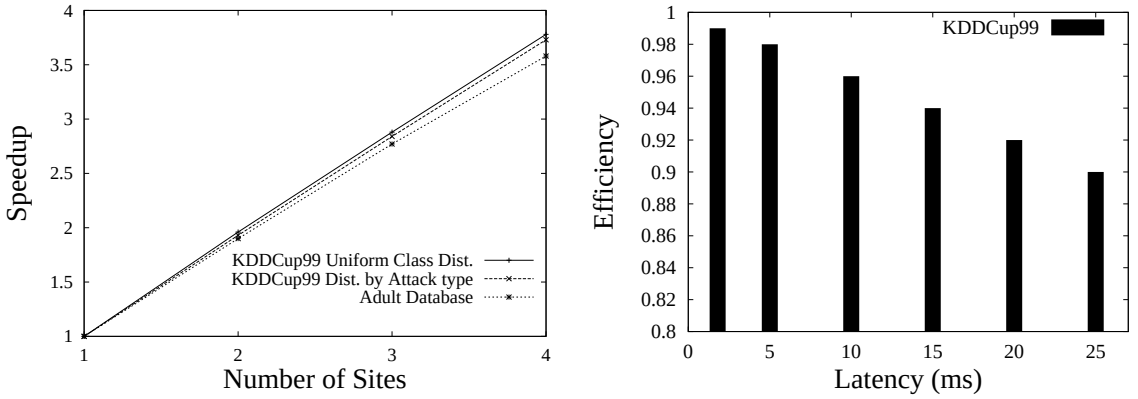

Figure 7: (a) Speedup within a cluster and (b) Efficiency in a wide area network

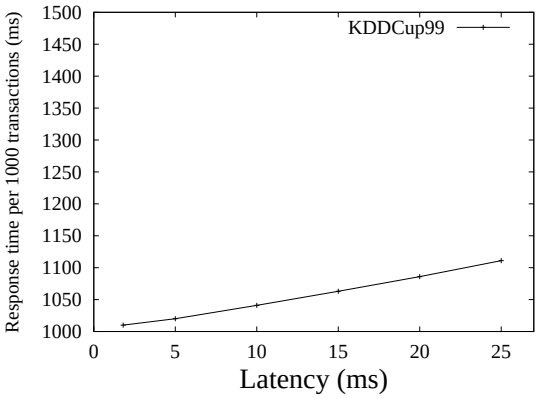

Figure 8: Response time per 1000 network transactions

5 Conclusion

Applying existing distance-based outlier detection algorithms to mixed attribute data has a major limitation in that distance is ill-defined in the categorical data space. In this paper, we presented LOADED, an algorithm for effective outlier and anomaly detection in such mixed attribute datasets. Our work combines the following key components. First, we define a robust unifying similarity metric that combines a link-based approach (for categorical attributes) with correlation statistics (for continuous attributes). Data points that violate these dependencies are flagged as outliers. It is a one pass algorithm that can operate on evolving datasets. Second, since the memory requirements imposed by LOADED can be quite high in large feature spaces, we describe a tunable approximation mechanism that reduces its memory requirements at a small cost to accuracy. Third, we extend the algorithm to operate in a distributed setting, which is an important practical concern for applications like distributed network intrusion detection. Finally, we evaluated LOADED on several real mixed attribute datasets with good results. As part of future work, we are trying to extend the proposed work to reduce the memory requirements of our algorithm and are trying to evaluate our work on CISCO router Netflow data from the OSU Office of Information Technology routers.

References

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

[2] S. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *ACM SIGKDD*, 2003.

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

[4] R. Bolton and D. Hand. Statistical fraud detection: A review. *Statistical Science*, 2002.

[5] M. Breunig *et al.* LOF: Identifying density-based local outliers. In *ACM SIGMOD*, 2000.

[6] E. Eskin *et al.* A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In *Data Mining for Security Applications*, 2002.

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

[8] A. Ghoting and S. Parthasarathy. DASPA: A disk-aware stream processing architecture. In *SIAM High Performance Data Mining Workshop*, 2003.

[9] S. Guha *et al.* ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 2000.

[10] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2001.

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

[12] Z. Huang. Clustering large data sets with mixed numeric and categorical values. In *PAKDD*, 1997.

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

[14] M. Jiang *et al.* Feature mining paradigms for scientific data. In *SIAM Data Mining*, 2003.

[15] T. Johnson *et al.* Fast computation of 2d depth contours. In *ACM SIGKDD*, 1998.

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

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

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

[19] A. Lazarevic *et al.* A comparative study of outlier detection schemes for network intrusion detection. In *SIAM Data Mining*, 2003.

[20] M. Mahoney and P. Chan. Learning non-stationary models of normal network traffic for detecting novel attacks. In *ACM SIGKDD*, 2002.

[21] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, 2002.

[22] M. Otey *et al.* NIC-based intrusion detection: A feasibility study. In *ICDM Workshop on Data Mining for Cyber Threat Analysis*, 2002.

[23] S. Papadimitriou *et al.* LOCI: Fast outlier detection using the local correlation integral. In *ICDE*, 2003.

[24] K. Penny *et al.* A comparison of multivariate outlier detection methods for clinical laboratory safety data. *The Statistician, Journal of the Royal Statistical Society*, 2001.

[25] Rulequest Research. www.rulequest.com.

[26] J. Rice. *Mathematical Statistics and Data Analysis*, 1995.

[27] K. Sequeira and M. Zaki. ADMIT: Anomaly-based data mining for intrusions. In *ACM SIGKDD*, 2002.