Near Linear Time Detection of Distance-Based Outliers and Applications to Security

Stephen D. Bay[1] **and Mark Schwabacher**[2]

[1]Institute for the Study of Learning and Expertise
2164 Staunton Court
Palo Alto, CA 94306
sbay@apres.stanford.edu

[2]NASA Ames Research Center
Computational Sciences Division
MS 269-3, Moffet Field, CA 94035
Mark.A.Schwabacher@nasa.gov

Abstract

Many automated systems for detecting threats are based on matching a new database record to known attack types. However, this approach can only spot known threats and thus researchers have also begun to use unsupervised approaches based on detecting outliers or anomalous examples. A popular method of finding these outliers is to use the distance to an example's k nearest neighbors as a measure of unusualness. However, existing algorithms for finding distance-based outliers have poor scaling properties, making it difficult to apply them to large datasets typically available in security domains. In this paper, we propose modifications to a simple, but quadratic, algorithm for finding distance-based outliers, and show that it achieves near linear time scaling allowing it to be applied to real data sets with millions of examples and many features.

Keywords: outlier detection, anomaly detection, nearest neighbors, aviation security

1 Introduction

Detecting threats by analyzing the examples in a database is an important problem in security domains. For example, researchers in data mining are developing algorithms to detect computer intrusions from audit records [21, 10, 19]. The U.S. Federal Aviation Administration developed the Computer Assisted Passenger Pre-screening System (CAPPS) [8, 11, 25], which screens airline passengers on the basis of their flight records and flags individuals for additional checked baggage screening.

One approach to addressing this task is developing sophisticated models of a threat and how they manifest themselves in a data set record. For example, although the exact details of CAPPS are not published, it is thought to assign higher threat scores to cash payments [25]. However, explicit threat models can only capture known attack types. Another approach is based on outlier or anomaly detection where one looks for unusual examples that appear suspicious and may require additional screening [20, 10, 22].

Outlier detection has a long history in statistics [3, 14], but has largely focussed on univariate data with a known distribution. These two limitations have restricted the ability to apply these types of methods to large real-world databases which typically have many different fields and have no easy way of characterizing the multivariate distribution of examples. Other researchers, beginning with the work by Knorr and Ng [17], have taken a non-parametric approach and proposed using an example's distance to its nearest neighbors as a measure of unusualness [2, 23, 18, 10]. Eskin et al. [10], and Lane and Brodley [20] applied distance-based outliers to detecting computer intrusions from audit data.

Although distance is an effective non-parametric approach to detecting outliers, the drawback is the amount of computation time required. Straightforward algorithms, such as those based on nested loops, typically require $O(N^2)$ distance computations. This quadratic scaling means that it will be very difficult

to mine outliers as we tackle increasingly larger data sets. This is a major problem for security screening databases where there are often millions of records. For example, U.S. airlines carry over 600 million passengers per year[8].

Recently, researchers have presented many different algorithms for efficiently finding distance-based outliers. These approaches vary from spatial indexing trees to partitioning of the feature space with clustering algorithms [23]. The main goal is developing algorithms that scale to large real data sets.

In this paper, we show that one can modify a simple algorithm based on nested loops, which would normally have quadratic scaling behavior, to yield near linear time mining on real, large, and high-dimensional data sets. Our goal is to develop algorithms for mining distance-based outliers that can feasibly be applied to help detect threats in large security data sets. Specifically, our contributions are:

- We show that an algorithm based on nested loops in conjunction with randomization and a simple pruning rule has near linear time performance on many large real data sets. Previous work reported quadratic performance for algorithms based on nested loops [17, 18, 23].
- We demonstrate that our algorithm scales to real data sets with millions of examples and many features, both continuous and discrete. To our knowledge we have run our algorithm on the largest reported data sets to date and obtained among the best scaling results for real data sets. Other work has reported algorithms with linear time mining but only for low-dimensional problems (less than 5) [17, 18] or have only tested the scaling properties on simple synthetic domains.
- We analyze why our algorithm performs so well. Under certain conditions, the results suggest that the time to process non-outliers, which are the large majority of points, does not depend on the size of the data set.
- We apply our algorithm to an airline passenger database and we discuss some limitations of our approach on this database.

The remainder of this paper is organized as follows. In the next section, we review the notion of distance-based outliers and present a simple nested loop algorithm that will be the focus of this paper. In Section 3, we show that although our simple algorithm has poor worst case scaling properties, for many large, high-dimensional, real data sets the actual performance is extremely good and is close to linear. In Section 4, we analyze our algorithm and attempt to explain the performance with an average case analysis. In Section 5, we present examples of discovered outliers to give the readers a qualitative feel for how the algorithm works on real data. Finally, we conclude this paper by discussing limitations and directions for future work.

2 Distance-Based Outliers

A popular method of identifying outliers is by examining the distance to an example's nearest neighbors [23, 18, 17, 2]. In this approach, one looks at the local neighborhood of points for an example typically defined by the k nearest examples (also known as neighbors). If the neighboring points are relatively close, then the example is considered normal; if the neighboring points are far away, then the example is considered unusual. The advantages of distance-based outliers are that no explicit distribution needs to be defined to determine unusualness, and that it can be applied to any feature space for which we can define a distance measure.

Given a distance measure on a feature space, there are many different definitions of distance-based outliers. Three popular definitions are

1. Outliers are the examples for which there are less than p other examples within distance d [17, 18].
2. Outliers are the top n examples whose distance to the kth nearest neighbor is greatest [23].
3. Outliers are the top n examples whose average distance to the k nearest neighbors is greatest [2, 10].

There are several minor differences between these definitions. The first definition does not provide a ranking and requires specifying a distance parameter d. Ramaswamy et al. [23] argue that this parameter could be difficult to determine and may involve trial and error to guess an appropriate value. The second definition only considers the distance to the kth neighbor and ignores information about closer points. Finally, the last definition accounts for the distance to each neighbor but is slower to calculate than definition 1 or 2. However, all of these definitions are based on a nearest neighbor density estimate [12] to determine the points in low probability regions which are considered outliers.

Researchers have tried a variety of approaches to find these outliers efficiently. The simplest are those using nested loops [17, 18, 23]. In the basic version one compares each example with every other example to determine its k nearest neighbors. Given the

neighbors for each example in the data set, simply select the top n candidates according to the outlier definition. This approach has quadratic complexity as we must make all pairwise distance computations between examples.

Another method for finding outliers is to use a spatial indexing structure such as a KD-tree [4], R-tree [13], or X-tree [5] to find the nearest neighbors of each candidate point [17, 18, 23]. One queries the index structure for the closest k points to each example, and as before one simply selects the top candidates according to the outlier definition. For low-dimensional data sets this approach can work extremely well and potentially scales as $N \log N$ if the index tree can find an example's nearest neighbors in $\log N$ time. However, index structures break down as the dimensionality increases. For example, Breunig et al. [7] used a variant of the X-tree to do nearest neighbor search and found that the index only worked well for low-dimensions, less than 5, and performance dramatically worsened for just 10 or 20 dimensions. In fact, for high-dimensional data they recommended sequential scanning over the index tree.

A few researchers have proposed partitioning the space into regions and thus allowing faster determination of the nearest neighbors. For each region, one stores summary statistics such as the minimum bounding rectangle. During nearest neighbor search, one compares the example to the bounding rectangle to determine if it is possible for a nearest neighbor to come from that region. If it is not possible, all points in the region are eliminated as possible neighbors. Knorr and Ng [17] partition the space into cells that are hyper-rectangles. This yields a complexity linear in N but exponential in the number of dimensions. They found that this cell based approach outperformed the nested loop algorithm, which is quadratic in N, only for four or fewer dimensions. Others use a linear time clustering algorithm to partition the data set [23, 10]. With this approach, Ramaswamy et al. demonstrated much better performance compared with the nested loop and indexing approaches on a low-dimensional synthetic data set. However, their experiments did not test how it would scale on larger and higher-dimensional data.

Finally, a few researchers have advocated projections to find outliers. Aggrawal and Yu [1] suggest that because of the curse of dimensionality one should focus on finding outliers in low-dimensional projections. Angiulli and Pizzuti [2] project the data in the full feature space multiple times onto the interval [0,1] with Hilbert space filling curves. Each successive projection improves the estimate of an example's outlier score in the full-dimensional space. Their initial scaling results are promising, and appear to be close to linear, however they have reported results on only two synthetic domains.

In this paper, we show that the simplest type of algorithm based on nested loops in conjunction with randomization and a pruning rule gives state-of-the-art performance. Table 1 shows our variation of the nested loop algorithm in more detail. The function distance computes the distance between any two examples using, for example, Euclidean distance for continuous features and Hamming distance for discrete features. The score function can be any monotonically decreasing function of the nearest neighbor distances such as the distance to the kth nearest neighbor, or the average distance to the k neighbors.

The main idea in our nested loop algorithm is that for each example in D we keep track of the closest neighbors found so far. When an example's closest neighbors achieve a score lower than the cutoff we remove the example because it can no longer be an outlier. As we process more examples, the algorithm finds more extreme outliers and the cutoff increases along with pruning efficiency.

Note that we assume that the examples in the data set are in random order. The examples can be put into random order in linear time and constant main memory with a disk-based randomization algorithm. One repeatedly shuffles the data set into random piles and then concatenates them in random order.

In the worst case, the performance of the algorithm is very poor. Because of the nested loops, it could require $O(N^2)$ distance computations and $O(N/blocksize * N)$ data accesses.

3 Experiments on Scaling Performance

In this section, we examine the empirical performance of the simple algorithm on several large real data sets. The primary question we are interested in answering is "How does the running time scale with the number of data points for large data sets?" In addition, we are also interested in understanding how the running time scales with k, the number of nearest neighbors.

To test our algorithm we selected the five real and one synthetic data set summarized in Table 2. These data sets span a range of problems and have very different types of features. We describe each in more detail.

- *Corel Histogram.* Each example in this data set encodes the color histogram of an image in a collection of photographs. The histogram has 32

Table 1: A simple algorithm for finding distance-based outliers. Lowercase variables represent scalar values and uppercase variables represents sets.

```
Procedure: Find Outliers
Input: k, the number of nearest neighbors; n, the number of outliers to return; D, a set of examples
in random order.
Output: O, a set of outliers.
Let maxdist(x, Y) return the maximum distance between x and an example in Y.
Let Closest(x, Y, k) return the k closest examples in Y to x.
begin
1.      c ← 0       // set the cutoff for pruning to 0
2.      O ← ∅       // initialize to the empty set
3.      while B ← get-next-block(D) {       // load a block of examples from D
4.          Neighbors(b) ← ∅ for all b in B
5.          for each d in D {
6.              for each b in B, b ≠ d {
7.                  if |Neighbors(b)| < k or distance(b,d) < maxdist(Neighbors(b),b) {
8.                      Neighbors(b) ← Closest(b,Neighbors(b) ∪ d, k)
9.                      if score(Neighbors(b),b) < c {
10.                         remove b from B
11.                     } } } }
12.         O ← Top(B ∪ O,n)      // keep only the top n outliers
13.         c ← min(score(o)) for all o in O       // the cutoff is the score of the weakest outlier
14.     }
15.     return O
end
```

bins corresponding to eight levels of hue and four levels of saturation.

- *Airline Passenger.* We obtained 90 days' worth of passenger data from a major U.S. airline. This database includes the information that each passenger provided to the airline (or to a travel agent) when buying an airline ticket. The passenger data is best seen as a relational database. For example, each record can contain multiple passengers traveling together, and each group of passengers can have multiple flight segments. For the experiments described in this paper, we flattened one day's worth of data to create a table in which each row represents one passenger flight segment. We selected 15 fields from this flattened table which we believe could be useful for passenger screening. We intend to scale up to 90 days' worth of data and to develop anomaly detection algorithms that operate directly on relational databases without first flattening them (see Section 6).

- *KDDCUP 1999.* The KDDCUP data contains a set of records that represent connections to a military computer network where there have been multiple intrusions by unauthorized users. The raw binary TCP data from the network has been processed into features such as the connection duration, protocol type, number of failed logins, and so forth.

- *Census.* This data set contains the responses from the 1990 decennial Census in the United States. The data has information on both households and individuals. We divided the responses into two tables, one that stores household records and another that stores person records, and treated each table as its own data set. Both the Household and Person data sets have a variety of geographic, economic, and demographic variables. Our data comes from the 5% State public use microdata samples and we used the short variable list [24]. In total, the 5% State sample contains about 5.5 million household and 12.5 million person records. For our experiments we used a maximum of 5 million records for each data set.

- *Normal 30D.* This is a synthetic data set generated from a 30-dimensional normal distribution centered on the origin with a covariance matrix equal to the identity matrix.

We obtained the data sets Corel Histogram and KDDCup 1999 from the UCI KDD Archive [15] and the census data from the IPUMS repository [24].

Table 2: Description of Data Sets

Data Set	Features	Cont.	Examples
Corel Histogram	32	32	68,040
Airline Passenger	15	3	439,381
KDDCup 1999	42	34	4,898,430
Household 1990	23	9	5,000,000
Person 1990	55	20	5,000,000
Normal 30D	30	30	1,000,000

We preprocessed the data by normalizing all continuous variables to the range [0,1] and converting all categorical variables to an integer representation. We then randomized the order of examples in the data sets. Randomizing a file can be done in $O(N)$ time and constant main memory with a disk-based shuffling algorithm as follows: Sequentially process each example in the data set by randomly placing it into one of n different piles. Recombine the piles in random order and repeat this process a fixed number of times.

We ran our experiments on a lightly loaded Pentium 4 computer with a 1.5 GHz processor and 1GB RAM running Linux. We report the wall clock time, the time a user would have to wait for the output, in order to measure both CPU and I/O time. The reported times do not include the time needed for the initial randomization of the data set and represent one trial. Preliminary experiments indicated that alternate randomizations did not have a major effect on the running time. To measure scaling, we generated smaller data sets by taking the first n samples of the randomized set. Unless otherwise noted, we ran experiments to return the top 30 anomalies with $k = 5$, a block size ($|B|$) of 1000 examples, and we used the average distance to the nearest k neighbors as the score function.

Our implementation of the algorithm was written in C++ and compiled with gcc version 2.96 with the -O3 optimization flag. We accessed examples in the data set sequentially using standard **iostream** functions and we did not write any special routines to perform caching. The total memory footprint of the executing program was typically less than 3 MB.

Figure 1 shows the total time taken to mine outliers on the six data sets as the number of examples varied. Note that both the x and y axes are in a logarithmic scale. Each graph shows three lines. The bottom line represents the theoretical time necessary to mine the data set given a linear algorithm based on the running time for $N = 1000$. The middle line shows the actual running times of our system. Finally, the top line shows the theoretical time needed assuming a quadratic algorithm based on scaling the running time for $N = 1000$.

These results show that our simple algorithm gives extremely good scaling performance that is near linear time. The scaling properties hold for data sets with both continuous and discrete features and the properties hold over several orders of magnitude of increasing data set size. The plotted points typically follow a straight line on the log-log graph which means that the relationship between the y and x axis variables is of the form $y = ax^b$ or $\log y = \log a + b \log x$, where a and b are constants. Thus, the algorithm scales with a polynomial complexity with an exponent equal to the slope of the line. Table 3 presents for each data set the slope of a regression line fit to the points in Figure 1. The algorithm obtained a polynomial scaling complexity with exponent varying from 1.13 to 1.32.

Table 3: Slope b of the regression fit relating $\log t = \log a + b \log N$ (or $t = aN^b$) where t is the total time (CPU + I/O), N is the number of data points, and a is constant factor.

Data Set	slope
Corel Histogram	1.13
Airline Passenger	1.20
KDDCup 1999	1.13
Household 1990	1.32
Person 1990	1.16
Normal 30D	1.15

One exception to the straight line behavior is the last point plotted for the Airline Passenger data set. Up to 100,000 examples the running time follows a straight line very closely with $b = 1.08$, however, the last point at $N = 439,000$ represents a large increase in time. We believe this is caused by the way we flattened the original relational database. For example, if there are two tables X and Y, with each example in X pointing to several different objects in Y, our flattened database will have examples with form (X_1, Y_1), (X_1, Y_2), (X_1, Y_3), (X_2, Y_4), ... and so forth. As it is likely that the closest neighbors of (X_1, Y_1) will be the examples (X_1, Y_2) and (X_1, Y_3) our algorithm may have to scan the entire data set until it finds them to obtain a low score. For small random subsets, it is unlikely that the related examples would be present and so this does not affect scaling performance. Flattening destroys the independence between examples which we will see in Section 4 is necessary for good performance.[1]

[1]We are also considering alternative problem formulations

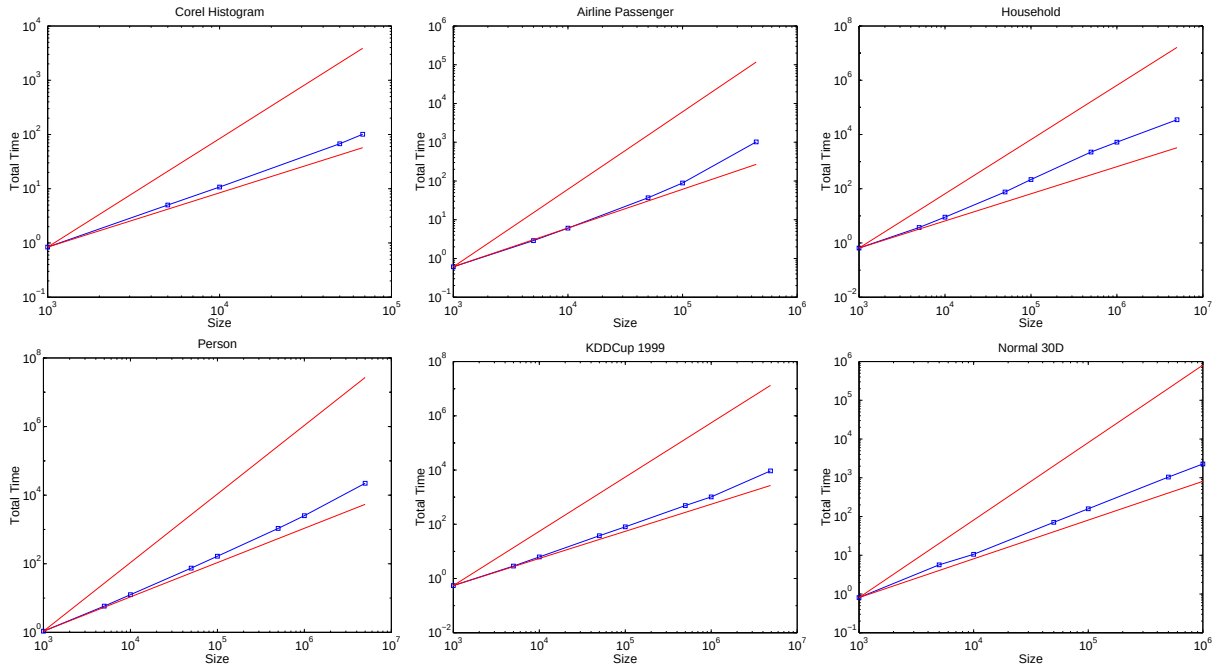

Figure 1: Total time (CPU and I/O) taken to mine outliers as N, the number of points, increases. The top and bottom lines represent the theoretical time taken by a quadratic and linear algorithm based on scaling the observed time at $N = 1000$.

We also examined how the total running time scales with k, the number of neighbors and the results for Normal 30D and Person ($N = 1,000,000$) are shown in Figure 2. The bottom line represents the observed time; the curved top line represents the time assuming linear scaling based on the timing results for $k = 5$. In these graphs, the y axis is logarithmic and the x axis is linear which means that a straight line indicates that the relationship between the y and x axis variables is of the form $y = ae^{bx}$ or $\log y = \log a + bx$ where a and b are constants. This relationship suggests that the running time scales exponentially with k. However, the empirical value of b as determined by a regression fit is very small. For Normal 30D $b = 0.0163$ and for Person $b = 0.0135$. Practically, the observed scaling performance is much better than linear for $k \leq 100$, mainly because of the large fixed computation costs unrelated to k.

4 Analysis of Scaling Time

In this section, we explain with an average case analysis why randomization in conjunction with pruning performs well, especially when much of the past literature reported that nested loop designs were extremely slow because of the $O(N^2)$ distance computations. In particular, both Knorr and Ng [17] and Ramaswamy et al. [23] implemented versions of the nested loop algorithm and reported quadratic performance.

Consider the number of distance computations needed to process an example $\mathbf{x}$. For now we assume that we are using outlier definition 2, rather than definition 3 which we used in our experiment, for ease of analysis. With this definition an outlier is determined by the distance to its kth nearest neighbor. In order to process $\mathbf{x}$ we compare it with examples in the data set until we have either (1) found k neighbors within the cutoff distance d, in which case we eliminate it as it cannot be an outlier, or (2) we have compared it with all N examples in the data set and failed to find k neighbors within distance d, in which case it is classified as an outlier.

We can think of this problem as a set of independent Bernoulli trials where we keep drawing instances until we have found k successes (k examples within distance d) or we have exhausted the data set. Let $\pi(\mathbf{x})$ be the probability that a randomly drawn example lies within distance d of point x, let Y be a random variable representing the number of trials until we have k successes, and let $P(Y = y)$ be the probability of obtaining the kth success on trial y. The probability $P(Y = y)$ follows a negative binomial dis-

that preserve independence of examples.

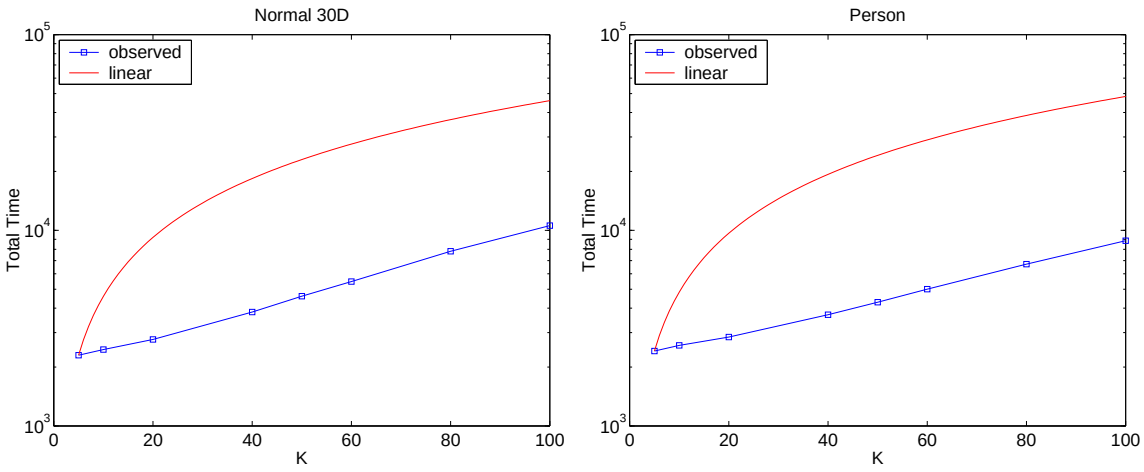

Figure 2: Total time (CPU and I/O) taken to mine outliers as k increases. The top curved line represents the theoretical time taken by an algorithm linear in k based on scaling the observed time for $k = 5$.

tribution:

$$P(Y = y) = \binom{y-1}{k-1} \pi(\mathbf{x})^k (1 - \pi(\mathbf{x}))^{y-k} \quad (1)$$

The number of expected samples we need to draw to process one example x is:

$$E[Y] = \sum_{y=k}^{N} P(Y = y)\ y + \left(1 - \sum_{y=k}^{N} P(Y = y)\right) N \quad (2)$$

The first term is the expectation of concluding a negative binomial series within N trials. That is, as we are processing an example, we keep drawing more examples until we have seen k that are within distance d, at which point we eliminate it because it cannot be an outlier. The second term is the expected cost of failing to conclude the negative binomial series within N trials, in which case we have examined all N data points because the example is an outlier (less than k successes in N trials).

The expectation of a negative binomial series with an infinite number of trials is,

$$\sum_{y=k}^{\infty} \binom{y-1}{k-1} \pi(\mathbf{x})^k (1 - \pi(\mathbf{x}))^{y-k}\ y = \frac{k}{\pi(\mathbf{x})} \quad (3)$$

This is greater than the first term in Equation 2. Combining Equations 2 and 3 yields,

$$E[Y] \leq \frac{k}{\pi(\mathbf{x})} + \left(1 - \sum_{y=k}^{N} P(Y = y)\right) N \quad (4)$$

Surprisingly, the first term which represents the number of distance computations to eliminate non-outliers does not depend on N. The second term, which represents the expected cost of outliers (i.e, we must compare with everything in the database and then conclude that nothing is close) does depend on N, yielding an overall quadratic dependency to process N examples in total. However, note that we typically set the program parameters to return a small and possibly fixed number of outliers. Thus the first term dominates and we obtain near linear performance.

One assumption of this analysis is that the cutoff distance is fixed. In practice, the cutoff distance changes with different values of N. However, we should expect that if the cutoff distance increases with larger N, then scaling will be better as $\pi(\mathbf{x})$ is larger and any randomly selected example is more likely to be a success (neighbor). Conversely, if the cutoff distance decreases, the scaling will be worse. In Figure 3 we plotted the relationship between b, the empirical scaling factor, and c_{50K}/c_{5K}, the ratio of the final cutoffs for $N = 50000$ and $N = 5000$ for the six data sets used in the previous section. We also plotted results for two additional data sets, Uniform 3D and Mixed 3D, which we believed would be respectively extremely difficult and easy. Uniform 3D is a three-dimensional data set generated from a uniform distribution between [-0.5,0.5] on each dimension. Mixed 3D is a mixture of the uniform data set (99%) combined with a Gaussian (1%) centered on the origin with covariance matrix equal to the identity matrix.

The results indicate that for many data sets the cutoff ratio is near or greater than 1. The only data set with an extremely low cutoff ratio was Uniform3D. The graph also indicates that higher values

of the cutoff ratio are associated with better scaling scores (lower b). This supports our theory that the primary factor determining the scaling is how the cutoff changes as N increases.

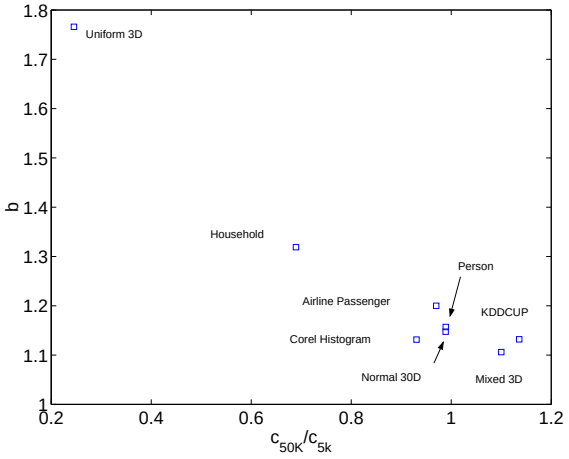

Figure 3: Empirical scaling factor b versus c_{50K}/c_{5K}, the ratio of cutoff scores for $N = 50,000$ and $N = 5,000$.

Figure 4 shows the running time plot for Uniform 3D and Mixed 3D. We expected Uniform 3D to have extremely bad scaling performance because it has no true outliers as the probability density is constant across the entire space. Increasing N simply increases the density of points and drops the cutoff score but does not reveal rare outliers. In contrast, the results for Mixed3D were extremely good ($b = 1.11$). In this data set, as we increase N we find more extreme outliers from the Gaussian distribution and the cutoff distance increases, thus improving pruning efficiency. Finally, we note that data sets with a true uniform distribution are probably rare in real domains.

5 Outliers in Census Data

We have applied our algorithm to mining outliers in the Airline Passenger database. However, for security reasons, we cannot describe the data set nor the discovered outliers in detail. Instead, we present results from the Household and Person data sets to give the readers a qualitative feel for outliers that can be mined from large data sets. We report selected results from running our outlier detection algorithm to return the top 30 outliers with $k = 5$. The full list of top 30 outliers for both household and person are available online[2] and we encourage readers to view this list directly.

[2]http://www.isle.org/∼sbay/papers/siam03/

The top outlier in the household database is a single family living in San Diego with 5 married couples, 5 mothers, and 6 fathers. In the census data, a family is defined as a group of persons related by blood, adoption, or marriage. To be considered a mother or father, the person's child or children must be present in the household. The house had a reported value of $85K and was mortgaged. The total reported income of the household was approximately $86K for the previous year.

Another outlier is a single-family rural farm household in Florence, South Carolina. The house is owned free and clear by a married couple with no children. This example is unusual because the value of the house is greater than $400K (not including the land), and they reported a household income of over $550K.

In the person data set one of the most extreme outliers was a 90+ year old Black Male with Italian ancestry who does not speak English, was enrolled in school[3], has a Doctorate degree, is employed as a baker, reported $110K income of which $40K was from wages, $20K from business, $10K from farm, $15K from welfare, and $20K from investments, has a disability which limits but does not prevent work, was a veteran of the U.S. armed forces, takes public transportation (ferry boat) to work, and immigrated to the U.S. 11-15 years ago but moved into his current dwelling 21-30 years ago. Clearly, there are inconsistencies in this record and we believe that this record represents an improperly completed form.

A second outlier was a 46 year old, White, widowed female living with 9 family members, two of which are her own children. She has a disability that limits but does not prevent her work as a bookkeeper or accounting clerk in the theater and motion picture industry. She takes public transportation to work (bus or trolley) and it takes her longer than 99 minutes to go from home to work.

A third outlier was a 19 year old, White, female with Asian ancestry and Mexican Hispanic origin with a disability that limits but does not prevent work. She earned $123K in business income, and $38K in retirement income (which may include payments for disabilities), and is also enrolled in school.

Finally, we ask readers to keep in mind that these outliers were discovered using a base set of features which were not collected for a specific task (e.g., security screening). In our own work on the airline passenger database, we are spending much effort on feature engineering to get the most relevant information for our algorithm.

[3]Taking a course that a high school or college would accept for credit would count under Census definitions.

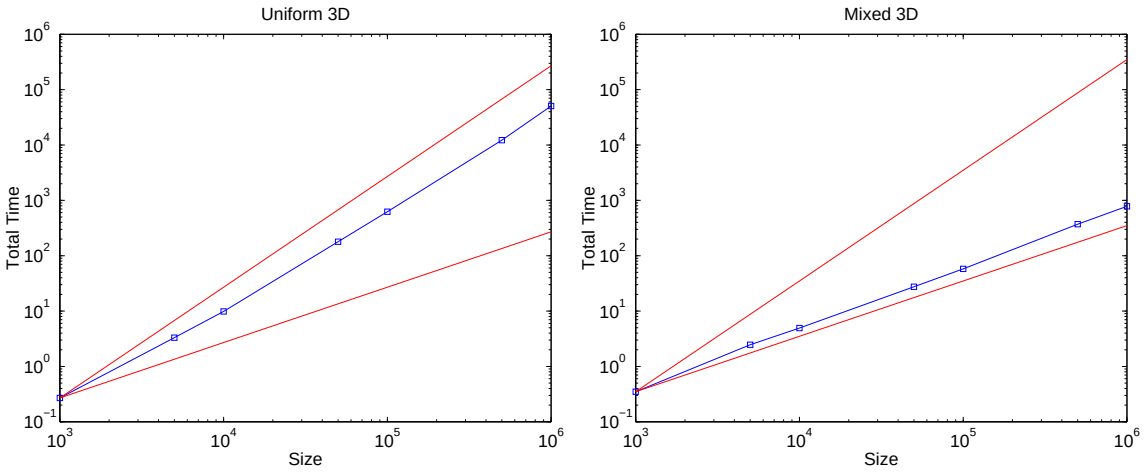

Figure 4: Total time (CPU and I/O) taken to mine outliers on data sets. For Uniform 3D $b = 1.76$, and for Mixed 3D $b = 1.11$.

6 Limitations and Future Work

In this paper, we addressed one roadblock toward applying distance-based outlier detection algorithms to security screening: dealing with the computation time required for large data sets. However, we feel that there are several limitations in applying outlier detection to security databases and that further research needs to be conducted to address these.

The largest and most pressing limitation is that our work has only addressed finding outliers in the data sets that can be represented with a vector space or equivalently a single table in a database. Clearly, almost all real data sources will be in the form of relational databases with multiple tables that relate different types of information about each other.

To address relational data, the simplest solution is to flatten the database with join operators to form a single table. While this is a convenient solution it loses much of the information available. For instance, a flattened database cannot easily represent passengers that have a variable number of flight trips. We also found that flattening a database could create dependencies between examples and this can reduce the effectiveness of randomization and pruning.

We are currently investigating how we can extend our algorithm to handle relational data natively. There are two research questions that arise. First, how does one define a distance metric to compare objects which may have a variable number of linked objects? There has been some work on defining metrics to work on relational data [6, 9, 16]. The central idea is to apply a recursive distance measure. That is, to compare two objects one starts by comparing their features directly, and then moves on to compare linked objects and so on. Second, how does one efficiently retrieve an object and its related objects to compare them in the context of searching for outliers? Retrieving related objects may involve extracting records in a non-sequential ordering and this can greatly slow database access.

A second area we did not address in this paper is determining how to set algorithm parameters such as k, the distance measure, and the score function. Each of these parameters can have a large effect on the discovered outliers. In supervised classification tasks one can set these parameters to maximize predictive performance by using a hold out set or cross-validation to estimate out-of-sample performance. However, outlier detection is unsupervised and no such training signal exists.

Finally, our approach combines randomization and pruning to speed the computation of the top outliers in the data set. Our method gives exactly the same results as computing all N^2 pairwise distances between examples and from these distances selecting the most extreme outliers. Alternatively, one could use a small random subset of examples to determine the outliers in the entire data set. This would require $O(NN_s)$ distance computations where N_s is the size of the subset. This subset approach is not guaranteed to return the same outliers as performing all N^2 pairwise comparisons but it may perform adequately. Our initial experiments indicate that the correspondence with the full N^2 comparison strongly depends on the data set. For example, on Mixed3D a subset of 1000 examples was sufficient to give 90% correspondence whereas on the Person data set a subset of 1000 points gave less than 30% correspondence and 1,000,000 examples resulted only in 70% correspon-

dence. We plan to investigate this issue further.

7 Conclusions

In our work applying outlier detection algorithms to large, real databases a major limitation has been scaling the algorithms to handle the volume of data. In this paper, we presented an algorithm based on randomization and pruning which finds outliers on many real data sets in near linear time. This efficient scaling allowed us to mine data sets with millions of examples and many features.

Acknowledgments

We thank Thomas Hinke and David Roland of NASA Ames for their help with the airline passenger data and for reviewing a draft of this paper. This work was supported by the CICT Program at NASA Ames Research Center under grant NCC 2-5496.

References

[1] C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 2001.

[2] F. Angiulli and C. Pizzuti. Fast outlier detection in high dimensional spaces. In *Proc. of the Sixth European Conf. on the Principles of Data Mining and Knowledge Discovery*, pages 15–26, 2002.

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

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[5] S. Berchtold, D. Keim, and H-P Kreigel. The X-tree: an index structure for high-dimensional data. In *Proc. of the 22nd Int. Conf. on Very Large Databases*, pages 28–39, 1996.

[6] G. Bisson. Learning in FOL with a similarity measure. In *Proc. of the Tenth National Conf. on Artificial Intelligence*, pages 82–87, 1992.

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

[8] Commission on Engineering and Technical Systems. *Assessment of Technologies Deployed to Improve Aviation Security: First Report*. Number NMAB-482-5. National Academy Press, 1999.

[9] W. Emde and D. Wettschereck. Relational instance-based learning. In *Proc. of the thirteenth Int. Conf. on Machine Learning*, 1996.

[10] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In *Data Mining for Security Applications*, 2002.

[11] Federal Aviation Administration. Security of checked baggage on flights within the united states – proposed rule. *Federal Register*, 64(74):19219–19240, April 19, 1999.

[12] E. Fix and J. L. Hodges. Discriminatory analysis: Nonparametric discrimination: Small sample performance. Technical Report Project 21-49-004, Report Number 11, USAF School of Aviation Medicine, Randolf Field, Texas, 1952.

[13] R. Guttmann. A dynamic index structure for spatial searching. In *Proc. of the 1984 ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[14] D. Hawkins. *Identification of outliers*. Chapman and Hall, 1980.

[15] S. Hettich and S. D. Bay. The UCI KDD archive. [http://kdd.ics.uci.edu/]. Irvine, CA: University of California, Department of Information and Computer Science, 1999.

[16] T. Horvath, S. Wrobel, and U. Bohnebeck. Relational instance-based learning with lists and terms. *Machine Learning*, 43:53–80, 2001.

[17] E. M. Knorr and R. T. Ng. Finding intensional knowledge of distance-based outliers. In *Proc. of the 25th VLDB Conf.*, 1999.

[18] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: algorithms and applications. *VLDB Journal: Very Large Databases*, 8(3-4):237–253, 2000.

[19] T. Lane and C. Brodley. An application of machine learning to anomaly detection. In *Twentieth Annual National Information Systems Security Conf.*, volume 1, pages 366–380, 1997.

[20] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2(3):295–331, 1999.

[21] W. Lee, S. Stolfo, and K. Mok. Adaptive intrusion detection: a data mining approach. *Artificial Intelligence Review*, 14(6):533–567, 2000.

[22] L. Portnoy, E. Eskin, and S. Stolfo. Intrusion detection with unlabeled data using clustering. In *Proc. of ACM CSS Workshop on Data Mining Applied to Security*, 2001.

[23] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Proc. of the ACM SIGMOD Conf.*, 2000.

[24] S. Ruggles and M. Sobek. Integrated public use microdata series: Version 2.0. [http://www.ipums.umn.edu/], 1997.

[25] U.S. House of Representatives, Subcommittee on Aviation. Hearing on aviation security with a focus on passenger profiling. http://www.house.gov/transportation/aviation/02-27-02/02-27-02memo.html, February 27, 2002.