Mining Distance Based Outliers in Near Linear Time with Randomization and a Simple Pruning Rule

Stephen D. Bay
Institute for the Study of Learning and Expertise
2164 Staunton Court
Palo Alto, CA 94306
sbay@apres.stanford.edu

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

ABSTRACT

Defining outliers by their distance to neighboring examples is a popular approach to finding unusual examples in a data set. Recently, much work has been conducted with the goal of finding fast algorithms for this task. We show that a simple nested loop algorithm that in the worst case is quadratic can give near linear time performance when the data is in random order and a simple pruning rule is used. We test our algorithm on real high-dimensional data sets with millions of examples and show that the near linear scaling holds over several orders of magnitude. Our average case analysis suggests that much of the efficiency is because the time to process non-outliers, which are the majority of examples, does not depend on the size of the data set.

1. INTRODUCTION

Detecting outliers, examples in a database with unusual properties, is an important data mining task. Recently researchers have begun focusing on this problem and have attempted to apply algorithms for finding outliers to tasks such as fraud detection [7], identifying computer network intrusions [18, 10], data cleaning [21], and detecting employers with poor injury histories[17].

Outlier detection has a long history in statistics [3, 13], 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 [16] have taken a non-parametric approach and proposed using an example's distance to its nearest neighbors as a measure of unusualness [2, 19, 17, 10].

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 problem is a major one for many real databases where there are often millions of records.

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 [19]. The common 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. 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 [16, 17, 19].
- 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) [16, 17] or have only tested the scaling properties on simple synthetic domains.
- We analyze why our algorithm performs so well. The result of an average case analysis suggests that under certain conditions, the time to process non-outliers, which are the large majority of points, does not depend on the size of the data set.

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 [19, 17, 16, 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 [16, 17].
2. Outliers are the top n examples whose distance to the kth nearest neighbor is greatest [19].
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. [19] 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 [11] 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 [16, 17, 19]. 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 [12], or X-tree [5] to find the nearest neighbors of each candidate point [16, 17, 19]. 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. [8] 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 [16] partition the space into cells that are hyperrectangles. 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 less dimensions. Others use a linear time clustering algorithm to partition the data set [19, 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 al-

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
```

gorithm 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 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 bins corresponding to eight levels of hue and four levels of saturation.
- *Covertype.* This data set represents the type of forest coverings for 30 × 30 meter cells in the Rocky Mountain region. For each cell, the data contains the cover type, which is the dominant tree species, and additional attributes such as elevation, slope, and soil type.
- *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 we 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 [20]. 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, Covertype, and

KDDCup 1999 from the UCI KDD Archive [14] and the census data from the IPUMS repository [20].

Table 2: Description of Data Sets

Data Set	Features	Continuous	Examples
Corel Histogram	32	32	68,040
Covertype	55	10	581,012
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 we converted 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 follow nearly straight lines on the log-log graphs 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
Covertype	1.25
KDDCup 1999	1.13
Household 1990	1.32
Person 1990	1.16
Normal 30D	1.15

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 [16] and Ramaswamy et al. [19] implemented versions of the nested loop algorithm and reported quadratic performance. However, Knorr and Ng did not use pruning or randomization in their algorithm, and Ramaswamy et al. only incorporated pruning.

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.

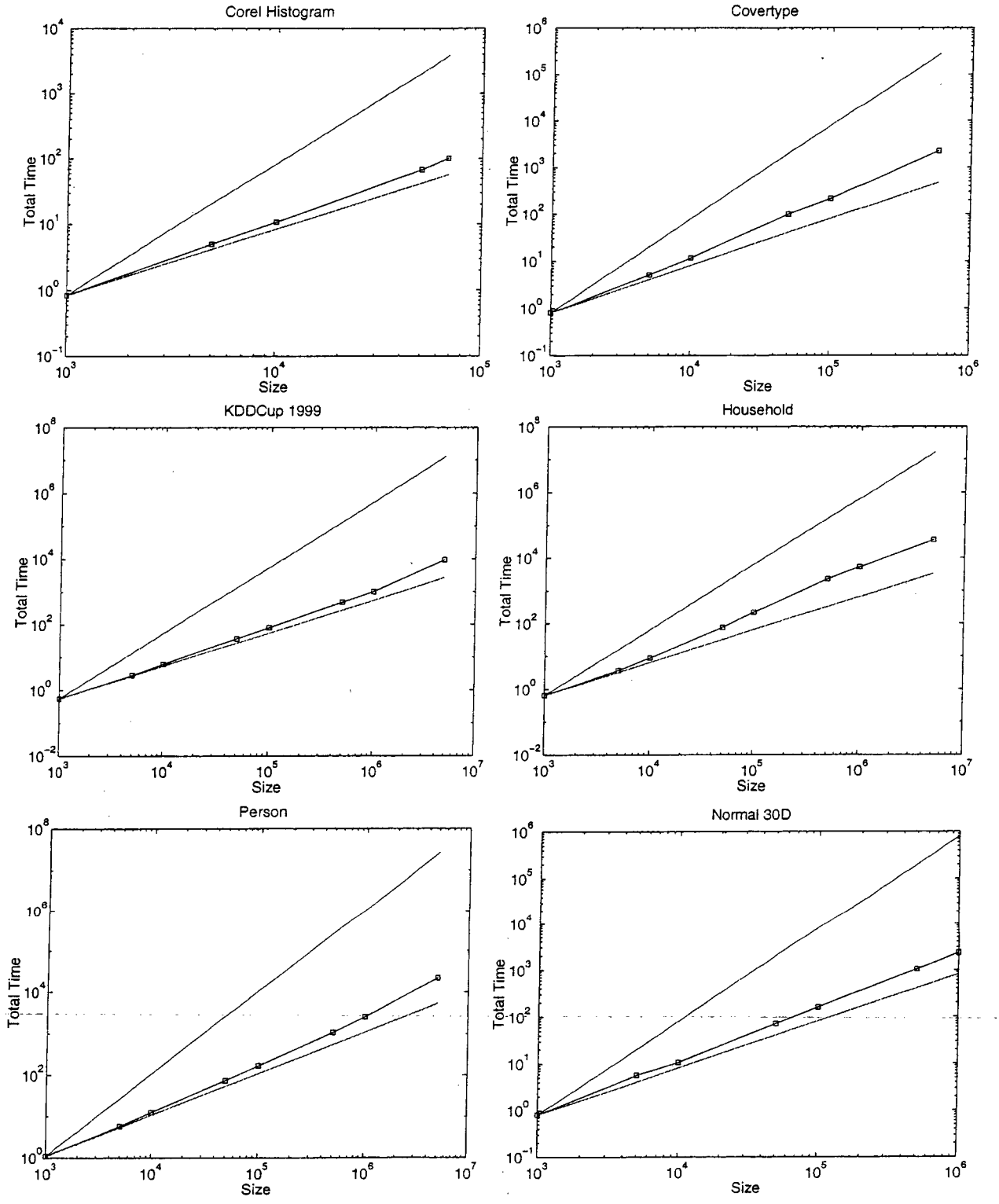

Figure 1: Total time (CPU and I/O) taken to mine outliers as N, the number of points, increases. The top and bottom lines represent the theoretical time taken by a quadratic and linear algorithm based on scaling the observed time at $N = 1000$.

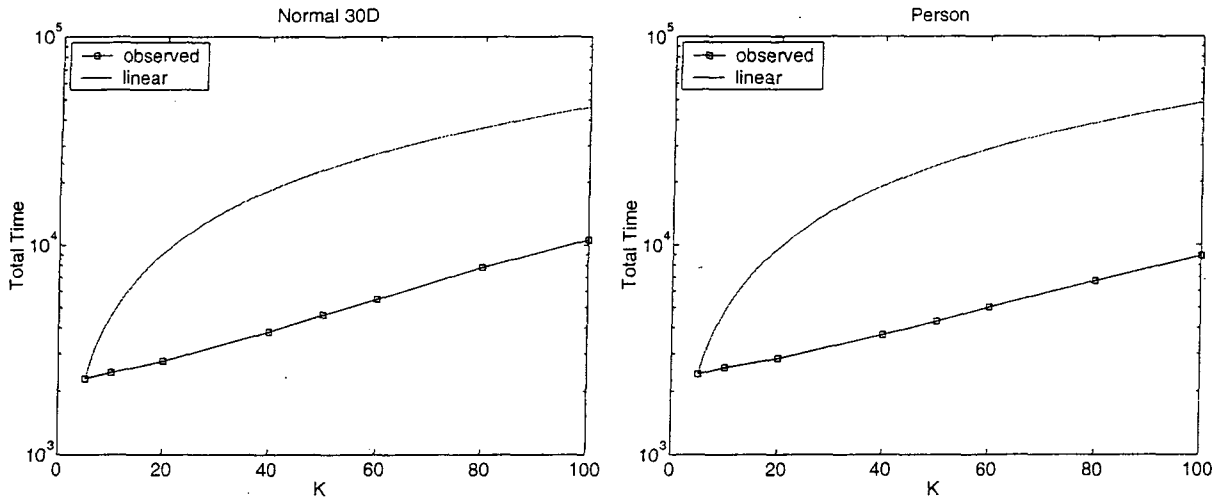

Figure 2: Total time (CPU and I/O) taken to mine outliers as k increases. The top curved line represents the theoretical time taken by an algorithm linear in k based on scaling the observed time for $k = 5$.

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 distribution:

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

The number of expected samples we need to draw to process one example $\mathbf{x}$ is:

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

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,

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

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

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

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 varies during program execution, and the final cutoff required to return the top n outliers changes with N. However, the relationship between cutoff value and percentage of the data set processed often stays the same for different values of N. For example, Figure 3 shows the plot of cutoff value against the percentage of the data set processed for different values of N.

In general, we expect that if the final 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 4 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

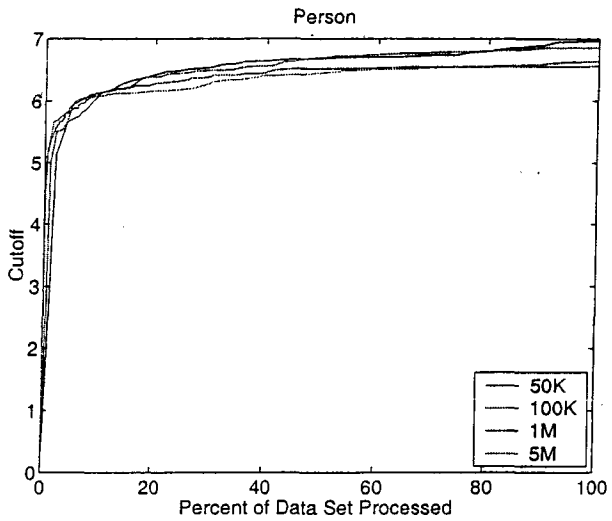

Figure 3: Value of the cutoff versus the percentage of the data set processed for N = 50K, 100K, 1M, and 5M.

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.

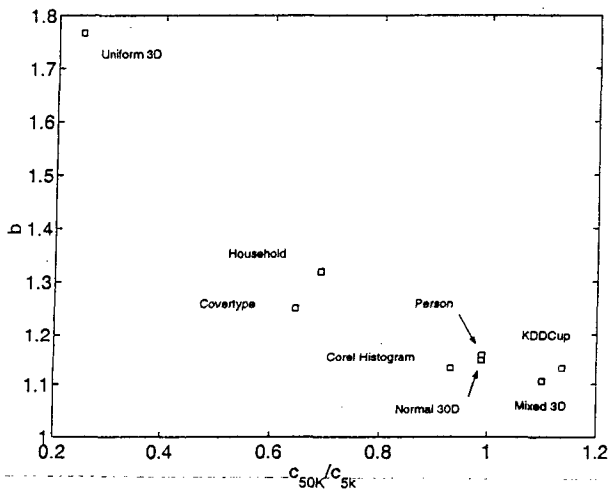

Figure 4: Empirical scaling factor b versus c_{50K}/c_{5K}, the ratio of cutoff scores for $N = 50,000$ and $N = 5,000$.

Figure 5 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

Although the use of distance based outliers is well established, in this section, we show results from the census data to give the readers a qualitative idea of the types of outliers found when large data sets are mined. We also compare the discovered outliers with examples flagged as unusual by GritBot, a commercial program from RuleQuest Research that was designed to find anomalies in data [21].

As we have limited space in this paper, we present only selected results. The full list of outliers on the Household and Person data sets for both our algorithm and GritBot are available online[1] and we encourage the readers to view this list directly.

We emphasize that we are not claiming that one set of results is better than another, but rather we feel these results show that distance based outlier detection finds unusual examples of a qualitatively different nature than GritBot.

5.1 Distance Based Outliers

We report selected results from running our outlier detection algorithm on the full set of 5 million examples to return the top 30 outliers with $k = 5$.

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[2], 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, 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.

[1] http://www.isle.org/~sbay/papers/kdd03/

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

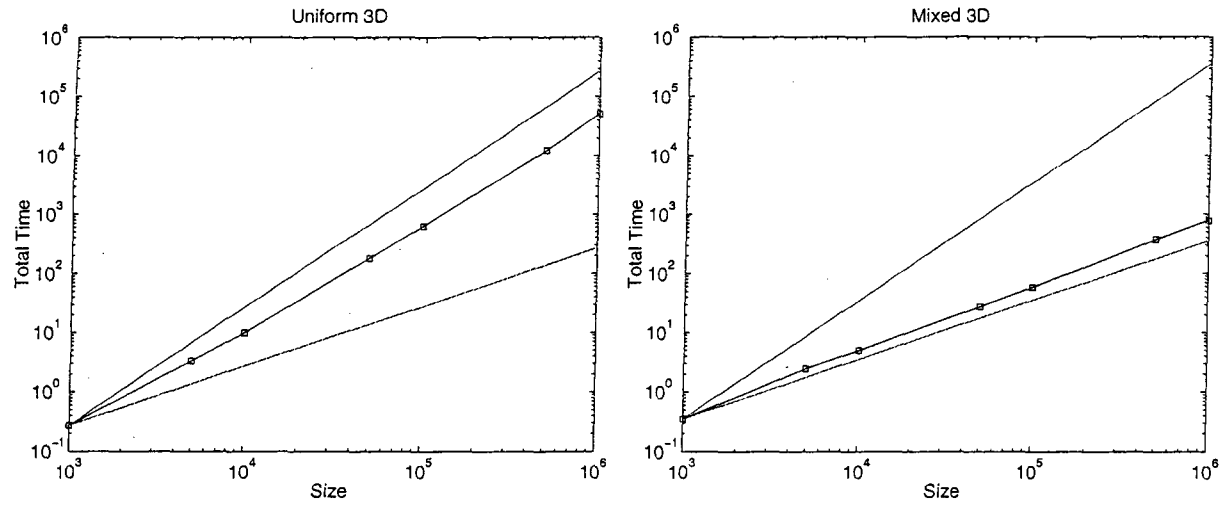

Figure 5: Total time (CPU and I/O) taken to mine outliers on the data sets Uniform 3D ($b = 1.76$) and Mixed 3D ($b = 1.11$).

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.

5.2 GritBot

GritBot finds records that have a surprising value on one attribute given the values of other attributes. For example, an outlier GritBot found on the Person data set was

```
case 481942:
    raced = White (31831 cases, 98.94% 'Black')
        ancestld = African American
        languagd = English
```

This means that 98.94% of people who have African American ancestry and who speak English, listed their race as Black. Case 481942 is unusual because the race listed was White.

We were not able to run GritBot on the household and person data sets with five million examples because of memory limitations. GritBot's requirements exceeded the available main memory as it loaded the entire data set and then allocated additional memory during program execution. However, we were able to run GritBot on smaller data sets, and specifically, we ran GritBot using the default settings on approximately one million household records and one half million person records.

Since GritBot and our algorithm compute two different sets of outliers, precise comparisons of their running times is not very meaningful. However, to give the reader a rough idea of their performance, GritBot took approximately 70 minutes to process one million household records and 170 minutes to process one half million person records on a 550 MHz SGI Origin 3000 with 4 GB of memory. In comparison, our algorithm took 87 and 18 minutes respectively to process similar amounts of data on a 1.5 GHz Pentium 4 with 1 GB of memory.[3]

In contrast to the results from distance based outliers, GritBot found qualitatively different outliers. For example, on the household data GritBot found a total of 266 anomalies. These anomalies could be divided into roughly three groups:

- 228 records for which the household was listed as "Rural" although another field indicated that the household was urban (e.g., metro = In metro area – Central city or citypop > 100000)
- 28 records for which the household was listed as "Urban" although another field indicated that the household was rural.
- 10 records with a total family income (ftotinc) greater than the household income (hhincome). By definition the household income should be greater than or equal to the family income.

On the person data set, GritBot found a total of 1407 anomalies. Unlike the household data, we could not place the examples into neat categories, but as before GritBot found records with unusual combinations of attributes which included

[3]The datasets were not exactly identical as they contained different samples of the Census records.

- people with unusual combinations of ethnicity, Hispanic origin, and race. For example, GritBot found records for people who are White and African-American, Black and Italian, Black and Swedish, Black and German, Black and Polish, Hispanic and Scotch-Irish.
- people who live in the same house where they lived 5 years ago, but also claimed to live in a different country five years ago.
- people who don't work, but have a place of work.
- a person whose ancestry is Mexican, but the language spoken at home is Chinese.
- a 16 year old person who last worked more than 10 years ago.
- a 75 year old female veteran.

In general, GritBot tended to find examples in which a small number of attributes made the example unusual. This is not surprising as by default GritBot is set to examine four or less conditions. However, GritBot often did not use all four conditions and many outliers had only one or two terms.

6. LIMITATIONS AND FUTURE WORK

The main goal of our experimental study was to show that our algorithm could scale to very large data sets. We showed that on large, real, high-dimensional data sets the algorithm had near linear scaling performance. However, the algorithm depends on a number of assumptions, violations of which can lead to poor performance.

First, our algorithm assumes that the data is in random order. If the data is not in random order and is sorted then the performance can be poor. For example, the Census data as retrieved from the IPUMS repository [20] came with the examples sorted by state. This can cause problems when our algorithm considers a person from Wyoming. It will try to eliminate it by finding the k nearest neighbors who are also likely to be from Wyoming. To find these neighbors, the algorithm will first scan all examples from states Alabama to Wisconsin given the sequential manner it accesses the data.

Second, our algorithm depends on the independence of examples. If examples are dependent in such a way that they have similar values (and will likely be in the set of k nearest neighbors) this can cause performance to be poor as the algorithm may have to scan the entire data set to find the dependent examples.

An extreme version of this problem can occur when the data set originates from a flattened 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.

However, our algorithm may still perform acceptably on data sets with less severe violations. For example, the examples in the Person data set are not completely independent as they are tied together by a common household.[4] However, the performance on this data set ($b = 1.16$) was still very good.

The third situation when our algorithm can perform poorly occurs when the data does not contain outliers. For example, our experiment with the examples drawn from a uniform distribution had very poor scaling. However, we believe data sets of this type are likely to be rare as most physical quantities one can measure have distributions with tails.

We are interested in extending our work in this paper in several ways. First, we are interested in speeding up the algorithm even further. In Section 4 we showed that the scaling performance depended on how the cutoff changes as we process increasingly larger data sets. The algorithm starts with a cutoff threshold of zero which increases as better outliers are found. One modification is to start the algorithm with a pre-defined cutoff threshold below which we would consider any example to be uninteresting. In preliminary experiments, a good initial guess could cut time to a third. There may also be automatic ways to get a good cutoff early. For example, we could first process the examples with a small data set to get an idea of the examples that are most unusual. We then place these examples at the beginning of the data file.

Another 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. Many real data sources will be in the form of relational databases with multiple tables that relate different types of information to 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 households that have a variable number of individuals. We also found that flattening a database could create dependencies between examples and, as we explained above, 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 for relational data [6, 9, 15]. 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 it's 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.

[4]The Census microdata is based on cluster samples, i.e., the samples are made of households or dwellings from which there may be multiple individuals. Individuals from the same household are not independent.

Finally, there are many practical issues with algorithms for mining distance based outliers that we did not investigate such as determining how to set algorithm parameters such as k, the block size, the distance measure, and the score function. Each of these parameters can have a large effect on the discovered outliers (or running time for the block size). 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.

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 addressed the scaling problem with 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.

8. ACKNOWLEDGMENTS

We thank Thomas Hinke and David Roland of NASA Ames 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.

9. REFERENCES

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

[2] F. Angiulli and C. Pizzuti. Fast outlier detection in high dimensional spaces. In *Proceedings of the Sixth European Conference 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 *Proceedings of the 22nd International Conference on Very Large Databases*, pages 28–39, 1996.

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

[7] R. J. Bolton and D. J. Hand. Statistical fraud detection: A review (with discussion). *Statistical Science*, 17(3):235–255.

[8] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying density-based local outliers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2000.

[9] W. Emde and D. Wettschereck. Relational instance-based learning. In *Proceedings of the thirteenth International Conference 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] 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.

[12] R. Guttmann. A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

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

[14] 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.

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

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

[17] 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.

[18] 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.

[19] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the ACM SIGMOD Conference*, pages 427–438, 2000.

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

[21] Rulequest Research. Gritbot. [http://www.rulequest.com/].