



The University of Sydney

**Disk-Based Successive Sampling for Outlier
Detection in High Dimensional Data**

Technical Report Number 544

January 2004

Pei Sun and Sanjay Chawla

ISBN 1 86487 622 0

**School of Information Technologies
University of Sydney NSW 2006**

Disk-Based Successive Sampling for Outlier Detection in High Dimensional Data

Pei Sun
The University of Sydney
Madsen Building F09
NSW 2006, Australia
psun2712@it.usyd.edu.au

Sanjay Chawla
The University of Sydney
Madsen Building F09
NSW 2006, Australia
chawla@it.usyd.edu.au

ABSTRACT

We propose a sampling based outlier detection method for large high-dimensional data. Our method consists of two phases. In the first phase, we combine a “successive sampling” strategy with a simple randomized partitioning technique to generate a candidate set of outliers. This phase requires one full data scan and the running time has linear complexity with respect to the size and dimensionality of the data set. An additional data scan, which constitutes the second phase, extracts the actual outliers from the candidate set. The running time for this phase has complexity $O(CN)$ where C and N are the size of the candidate set and the data set respectively. A major strength of the proposed approach is that no partitioning of the dimensions is required thus making it particularly suitable for high dimension data. Furthermore our method can handle both continuous and categorical attributes. We also present a detailed experimental evaluation of our proposed method on real and synthetic data sets.

General Terms

Outlier Detection, Sampling, High Dimension, Randomization

1. INTRODUCTION

Historically, two schools of thought have existed regarding the utility of detecting outliers in data. The first school argues that outliers are instances of “noise”, generated because of instrumentation errors (machine or human) and must be identified and discarded before the data is seriously analyzed. This school also advocates the use of estimators which are less sensitive to noise even at the cost of accuracy. For example, the median is such a robust estimator: insensitive to outliers but less accurate than the mean.

The second school of thought, to which probably most of the data mining researchers belong, has relatively more faith in the underlying fidelity of the data, and argues that outliers are indicative of events which were not anticipated, or which lie outside the norm. For example, a relatively large body of research has focused on applications of outlier detection for network intrusion [11]. Here

the argument is that malicious events, like hacking, are outside the norm, and the underlying signature of such events could be indicative of the intent. Since the underlying signature is typically multivariate the emphasis has always been on the development of techniques for outlier detection in high-dimensional data.

Knorr and Ng [6] were the first to propose a definition of outliers which was free of any distributional assumptions and was readily generalizable to multi-dimensional data. *An object O in a dataset T is a $DB(p,D)$ -outlier if at least fraction p of the objects in T lie at a greater distance than D from O .* The authors then go on to prove that this definition of outliers generalizes the folk definition of outliers “three standard deviations away from the mean”. For example, if the dataset T is generated from a normal distribution with mean μ and standard deviation σ , and $t \in T$ is such that $\frac{|t-\mu|}{\sigma} > 3$ then t is a $DB(p,D)$ outlier with $p = 0.9988$ and $D = .13\sigma$. Similar extensions were shown for other well known distributions including the Poisson.

While the notion of a $DB(p,D)$ outliers set the stage for a distribution free research in outlier detection there were some practical deficiencies with this definition. First this definition did not offer a mechanism to rank the outliers and more importantly it is difficult to estimate apriori what a suitable value of D in $DB(p, D)$ should be, especially in high-dimensional data. To address this Ramawamay et. al [10] have proposed the following definition which will remain our working definition for the rest of the paper: *Outliers are the top n data elements whose distance to the k th nearest neighbor is greatest.* Such outliers will be called DB_n^k outliers.

Based on the above definitions several algorithms have appeared in the literature based on either partitioning, indexing or clustering of the underlying multidimensional data space [3, 2, 10]. We will survey some of the important results in Section 2.

In this paper we show that by strategically sampling in the data space, DB_n^k outliers can be identified with high accuracy. Our sampling method is based on the successive sampling technique proposed by Mettu and Plaxton [9] who use it design a time optimal constant-factor randomized approximation algorithm for the K-median clustering problem. The basic idea of our sampling strategy is to sample in stages and at each stage purge regions of high density. What is left are the candidate outliers and a another single-pass through the data can be used to extract all legitimate DB_n^k outliers from the candidate set.

Specifically, our main contributions are:

- We propose a sampling strategy based on successive sampling [9] to discover DB_n^k outliers with high accuracy. Our sampling strategy does not require the partitioning of the underlying data space and thus gracefully handles one aspect of the “curse of high dimensionality”. As far as we know this is the only sampling-based outlier detection algorithm which can scale to large high-dimensional data.
- The algorithm is disk-based and, after randomization, requires only two data scans. Furthermore the algorithm can handle both continuous and categorical data.
- We have carried out on an extensive experimental evaluation, on real and synthetic data sets, to test the scalability, accuracy and dependence on parameters, of our proposed algorithm.

We should note that Aggarwal and Wu [1] have questioned the validity of using distance as a metric to determine outliers in high-dimensional space. The argument is that in high-dimensional space data is sparse so every point is an equally good candidate for being an outlier. However they show that by projecting data into a lower dimension space, meaningful outliers can still be discovered. While we do not address this problem directly, we should point out that our sampling strategy will remain effective in a lower-dimension projected space.

The rest of the paper is as follows. In Section 2 we survey related work in outlier detection algorithms. In Section 3 we collect all the important definitions and notations used throughout the paper. Section 4 is the core of paper and is devoted to development of the algorithm and its analysis. In Section 5 comprehensive experiment results are reported and discussed. We conclude in Section 6 with a summary and directions for future work.

2. RELATED WORK

As we have noted in the introduction the study of distribution-free outlier detection was initiated by Knorr and Ng [6] with their definition of a distance based $DB(p,D)$ outlier.

Till then statisticians had extensively studied the problem in the context of a given distributional model. The textbook approach [4] works as follows. Let M be a given model of the underlying data and let t be a realization of M , then if $p(t|M)$ is smaller than a threshold then t is deemed an outlier. The shortcoming of this approach is ofcourse that for high-dimensional data it may be very hard to pin down a suitable candidate for M .

The simplest algorithm to detect all $DB(p,D)$ (even DB_n^k) outliers is a nested loop approach. Each point in the database is examined with all other points until it no longer can possibly be an outlier, i.e., when more than “ p percent of the points lie within a distance D of the point being examined.” However this simple approach has a complexity of $O(dN^2)$ making it impractical for large data sets.

If the nested loop approach is carried out after the data has been indexed, using spatial index structures like R^* trees then in theory the overall complexity can be reduced to $O(dN \log N)$. However as several authors have noted that in high-dimensional space these data structures are inefficient and a search (for the k -nearest neighbor) reduces to a sequential scan.

In order to amortize the cost of examining every point, some researchers have proposed a partitioning approach to search for the

k nearest neighbors. By partitioning the underlying feature space into hypercubes, all the points in each partition can be examined in bulk. However a partitioning approach is not conducive in high-dimension space. In order to reduce the large number of partitions Ramaswamy et. al [10] have proposed an approach which combines the partitions into clusters using a clustering algorithm. They report substantial improvements (over partitioning based approaches) on data sets whose dimensions vary from four to ten. However in high dimension space every cluster could contain potential outliers if the number of clusters is not large enough.

Aggarwal and wu [1] have questioned the validity of distance-based outliers in high dimensional space. The argument is that in high dimension data is sparse and every point is a potential outlier. They also propose a set of properties that outlier detection algorithms for high dimensional data should satisfy. The authors then introduce an algorithm based on a metaheuristic (evolutionary algorithm) which searches for low density cells in projected subspaces of the original space. One drawback of their approach that they still have to partition each dimension and calculate the density of each k -dimensional sub-cube. Another drawback is that it is likely to miss some outliers that are not emergent in the lower dimensional projection, for example a point surrounded by a spherical band of dense points at a certain distance.

Fabrizio et. al [2], proposed a new algorithm based on space-filling curves The method is likely to do $d+1$ sorts and scans of the whole data set with cost of $O(d^2 kN)$, where d is the number of dimensions and N is the number of points in the dataset, so it can not scale well to the large and high-dimensional data set. Another limitation of [1], [7], [2] and [10] is that they can only deal with numerical attributes.

Bay and Schwabacher [3] have shown that randomization in conjunction with the nested loop algorithm results in near-linear average case complexity. In each loop, the algorithm uses part of the data to compare with the whole data set and keeps track of the closest neighbors currently found for each point. When the score of a point’s closest neighbors is lower than a threshold, the point is removed from the data set because it no longer can be an outlier. As more points are being processed, more extreme outliers will be found and the threshold will be increased, so more points will be pruned quickly. An important features of this algorithm is that it can be applied to categorical attributes, for example, by using the Hamming distance metric. However, when the whole data set cannot be held in memory, this method will not work well, because it is required to do $N/\text{blocksize}$ times expensive data scan, where N is the number of records in the data set. Furthermore the running time is extremely sensitive to the number of outliers that are required to be reported.

The only sampling based approach, that we are aware of, is the biased sampling technique reported by Kollios et. al [7]. The authors design a sampling strategy such that the probability that a point will be drawn into the sample depends on the local density of the data set. The points will be sampled only if they are located in the region with density smaller than a predefined threshold. The core of this technique is to build a density estimator for the data set using a kernel-density method. The authors only report results for two-dimensional data. Building a kernel-density estimator in high-dimension will be a non-trivial exercise as it will involve computing the integral using either numeric or monte-carlo techniques. Furthermore the method requires the partitioning of the

Notation	Description
U	the whole dataset
U'	candidate set for outlier detection
U_p	a partition of U
d	the dimensionality of U
Point	each row in the dataset is a point in the d dimensional space
N	the number of points in the whole dataset
N_p	the number of points in a partition (chunk)
$D(s,p)$	the distance between the point s and p
$D^k(p)$	the distance between the point p and its k^{th} nearest neighbor
α	sampling ratio ($0 < \alpha < 1$)
β	a threshold ($0 < \beta < 1$). When $ U_p \leq \beta N_p$, our algorithm will stop removing points from U_p
n	number of outliers reported
M	the size of a container, which hold M nearest neighbors of a sampled point

Table 1: Definition and Notation

feature space into cells, something which is not practical in high-dimension space.

3. DEFINITIONS AND NOTATION

In this section we collect all the important definitions and notation used throughout the paper.

- **DB(p,D)-outlier:** An object O in a dataset T is a DB(p,D) outlier if at least fraction p of the objects in T lie at a greater distance than D from O .
- **DB_n^k -outlier:** Top n data elements whose distance to the k^{th} nearest neighbor is greatest.
- **Successive Sampling:** A sampling strategy introduced by Mettu and Plaxton [9] which consists of sampling points from a database in stages.
- **Euclidean Distance:** The L^2 distance metric for continuous metric. If x and y are two points in a d -dimensional space, then $d(x, y) = (\sum_{i=1}^d (x_i - y_i)^2)^{\frac{1}{2}}$.
- **Hamming Distance:** A distance measure for categorical attributes. The distance between two bit patterns of equal length is the number of slots where the bits disagree.
Note that we have two sets of parameters. The parameters k and n are associated with the problem definition. Parameters α, β and M are associated with the algorithm.

All the notations that will be used throughout the paper is listed in table 1.

4. ALGORITHM AND ANALYSIS

The proposed algorithm for outlier detection consists of two phases. In phase one a set of candidate outliers is generated. In phase two a full scan through the data is used to extract DB_n^k outliers from the candidate set.

Phase one is the core of the algorithm and is adapted from the successive sampling technique introduced by Mettu and Plaxton [9,

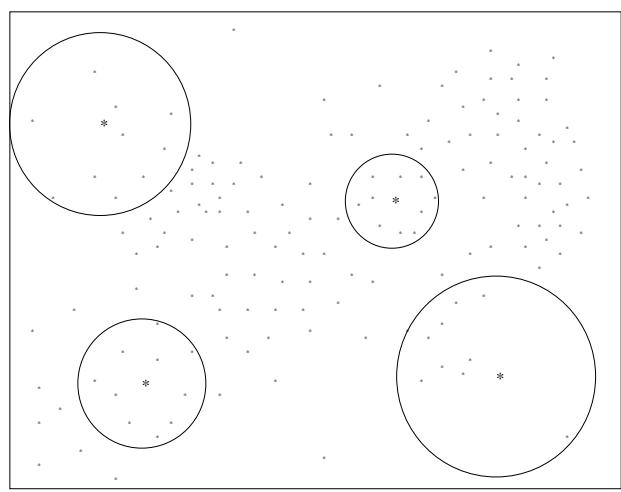


Figure 1: Original dataset with four sampled points and their top ten nearest neighbors

8]. The authors used the sampling technique to develop an approximation algorithm for the k -median clustering problem.

Our adaptation proceeds as follows. We randomly sample a pre-defined fraction (α) of points from the dataset, and then, for each sampled point, construct a hyper-ball which can exactly hold its M nearest neighbors. These hyper-balls are then sorted based on their radii and the smaller hyper-balls, those whose radii are less than the median, are purged. This procedure is carried on recursively on the remaining points until a threshold (β) is reached. The points that remain are the candidate outliers. The main idea behind this method is that since all hyper-balls contain the same number of points the smaller balls are relatively more densely packed and therefore are less likely to contain outliers.

Figure 1 shows an example in a two dimensional space. Four points are sampled and each point is a center of a circle which exactly contains its ten nearest neighbors. All the points in two smaller circles, including the two sampled points, will be eliminated, but the points in two bigger ones will be retained as possible sample candidates in the next round.

By applying sampling in stages, we save a huge amount of running time for outlier detection compared with, say the nested-loop approach. However, this is not enough, because the complexity of this method is still quadratic in the size of dataset. We will prove this in the next section.

In order to address this problem, we first randomize the data set and break it into equal or near equal size partitions (chunks) according to a pre-defined parameter, and then we apply the sampling technique to each partition. Although the running time for sampling in each partition is quadratic with respect to the size of the partition, we can set it to a small value so that it will make a trivial contribution to the overall complexity. Now the complexity for candidate generation becomes linear with the number of partitions, i.e. the size of whole data set.

The partitioning is carried out after randomization so that all the partitions have the same distribution as the whole data set. If a point in a partition is a candidate outlier, it is also a candidate for the

whole data set. Another advantage of the partitioning approach is that the data for each partition is small and can be held in memory—so expensive disk access operations are minimized.

If the data set has been randomized, only two full data scans are required to enumerate all the outliers: one scan for candidate generation and one for outlier verification. In the remainder of the paper, we assume that the data sets have been pre-processed and randomized.

The value of container size, M , plays a crucial role in phase one and may appear to be hard to set. However, in practice, we can change the value of M and make it adapt to the size of the dataset that is left after each sampling round. In the beginning when the dataset has pockets of high density, we can use a large value for M to gain efficiency. As we keep removing points from the dataset, it becomes sparser, and we can gradually change the value of M to a smaller one to gain accuracy. By doing so, the accuracy of our algorithm can be improved significantly with minor increase in the running time.

For example, if in the first step, the size of the data set is 5000 and the value for M is 100, after removing some points, the size of dataset becomes 4000, the parameters M adapts to size 80. The general formula we use is

$$M_{k+1} = \min\left\{10, \frac{U_{k+1}}{U_k} M_k\right\}$$

where U_k is the size of the partition in step k .

We will demonstrate the effect of adapting M to the size of remaining data set in Section 5.

4.1 Algorithm

The algorithm for **Phase One** is shown in Table 2. The algorithm begins by loading a portion of U into U_p , and choosing a random sample S of size $\alpha|U_p|$. For each point in $s \in S$ a container C_s of size M is constructed to hold the M current nearest neighbors. Each point in U_p is associated with its nearest sample point s and added to the container C_s . Associated with each C_s is a hyper-ball containing exactly M nearest neighbors of s . The hyper-balls are sorted based on the radii and those balls whose radii is smaller than the median are purged of all their points including the sampled points. This process (of sampling and purging) is repeated within each partition U_p till the number of points in U_p falls below the threshold $\beta|U_p|$. The whole process is carried out till all the partitions, $\lceil N/N_p \rceil$ of them, are examined.

The algorithm for **Phase Two** is shown in Table 3. The input to this algorithm is U , the full data set and U' , the candidate set of outliers. Each element of U' is examined with respect to U to verify if it is a DB_n^k -outlier. The result is a set of top n outliers.

4.2 Complexity Analysis

For each step of successive sampling of a partition, we list the size of the population, number of points sampled, and the computational cost in table 4. We have N/N_p partitions (ignoring ceiling function), so the total cost of phase one is:

$$\left(\frac{2dN_p^2}{M} \frac{\alpha M}{2} (1 + (1 - \frac{\alpha M}{2})^2 + \dots + (1 - \frac{\alpha M}{2})^{2(k-1)})\right) \frac{N}{N_p} \quad (1)$$

Phase One

Input: U, M, α, β

Output: U' (candidate set of outliers)

$U' \leftarrow \emptyset$

For ($i=0; i \leq \lceil N/N_p \rceil; i++$)

 read a partition of data from disk into U_p

While $|U_p| > \beta N_p$

 construct a set of points S by sampling $\alpha|U_p|$ points from U_p ;

 for each point s in S , construct a container C_s to hold its top M nearest neighbors;

 for each point $p(p \neq s)$ in U_p , assign it to a C_s such that $D(p, s)$ is the smallest for each $s \in S$;

 for each C_s , define a hyper ball, which can exactly contain its top M nearest neighbors and the center is s ;

 remove all the points, include the sampled points themselves, in the smaller hyper balls from U_p ;

 compute a new value for M according to the size of U_p ;

End while

$U' \leftarrow U' \cup U_p$

End for

Table 2: The Algorithm for Phase One. A sampling strategy to generate the candidate set of outliers.

In step $k+1$, sampling will stop when

$$\left(1 - \frac{\alpha M}{2}\right)^k N_p = \beta N_p$$

Thus:

$$\frac{\alpha M}{2} = 1 - \beta^{\frac{1}{k}} \quad (2)$$

Combining (1) and (2), the total cost of phase one becomes:

$$\frac{2dN_p^2 N}{M} (1 + (\beta^{\frac{2}{k}})^1 + (\beta^{\frac{2}{k}})^2 + \dots + (\beta^{\frac{2}{k}})^{k-1}) (1 - \beta^{\frac{1}{k}}) \quad (3)$$

Now, let

$$S = 1 + (\beta^{\frac{2}{k}})^1 + (\beta^{\frac{2}{k}})^2 + \dots + (\beta^{\frac{2}{k}})^{k-1}$$

Then,

$$S \beta^{\frac{2}{k}} = (\beta^{\frac{2}{k}})^1 + (\beta^{\frac{2}{k}})^2 + \dots + (\beta^{\frac{2}{k}})^{k-1} + (\beta^{\frac{2}{k}})^k$$

$$S - S \beta^{\frac{2}{k}} = 1 - (\beta^{\frac{2}{k}})^k = 1 - \beta^2$$

$$S = \frac{1 - \beta^2}{1 - \beta^{\frac{2}{k}}}$$

So,

$$(1 + (\beta^{\frac{2}{k}})^1 + (\beta^{\frac{2}{k}})^2 + \dots + (\beta^{\frac{2}{k}})^{k-1}) (1 - \beta^{\frac{1}{k}}) = \frac{1 - \beta^2}{1 + \beta^{\frac{1}{k}}}$$

When β is small and k large we get,

$$\lim_{\beta \rightarrow 0} \lim_{k \rightarrow \infty} \frac{1 - \beta^2}{1 + \beta^{\frac{1}{k}}} = \lim_{\beta \rightarrow 0} \frac{1 - \beta^2}{2} = \frac{1}{2}$$

Step	Population Size	Number of Sampled Points	Computational Cost
1	N_p	αN_p	$\alpha d N_p^2$
2	$(1 - \frac{\alpha M}{2}) N_p$	$\alpha (1 - \frac{\alpha M}{2}) N_p$	$\alpha (1 - \frac{\alpha M}{2})^2 d N_p^2$
3	$(1 - \frac{\alpha M}{2})^2 N_p$	$\alpha (1 - \frac{\alpha M}{2})^2 N_p$	$\alpha (1 - \frac{\alpha M}{2})^4 d N_p^2$
...
k	$(1 - \frac{\alpha M}{2})^{k-1} N_p$	$\alpha (1 - \frac{\alpha M}{2})^{k-1} N_p$	$\alpha (1 - \frac{\alpha M}{2})^{2(k-1)} d N_p^2$
k+1	$(1 - \frac{\alpha M}{2})^k N_p$		

Table 4: The Cost of Each Step in Phase One

Phase Two

Input: U, U' (candidate set of outliers)

Output: top n outliers

For ($i=0; i \leq \lceil N/N_p \rceil; i++$)

read a partition of data from disk into U_p ;

for each point in U' , compute the distance to each point in U_p , and keep tracking the distance to its k^{th} nearest neighbor;

End for

Sort all the candidates of outliers according to the distances calculated;

Output top n outliers.

Table 3: Algorithm for Phase Two. The candidate set of outliers are examined and only the DB_n^k outliers are retained.

Thus, the total cost of Phase One (formula (3)), becomes $\frac{dN_p N}{M}$, leading to the following lemma.

Lemma 1: The cost of phase one is $O(\frac{dN_p N}{M})$ when k is large (this means sampling ratio α is small) and β is small.

When the sampling ratio α is small, we remove fewer points in each step, so we need more iterations to finish the sampling. Since the data of a partition can be held in memory and all the sampling operations can be done within the memory, we can use a very small α , i.e. large k, without degrading the efficiency of our algorithm.

In phase two, we have to compute the distance between each point in U and each point in U' , so the complexity is $O(\beta d N^2)$. Typically, β takes on a very small value, e.g. 0.005, causing the cost for phase two to be only 1/200 of the comparisons with the nest-loop algorithm, which has the complexity of $O(d N^2)$. Experimental results on real and synthetic data sets in section 5 show that 0.005 is a suitable value for β . If the size of dataset is very large, β can take on even smaller values.

5. EXPERIMENTAL RESULTS

In this section we carry out extensive experiments on real and synthetic data sets to verify the accuracy and efficiency of our proposed approach. Since phase one of the algorithm involves several parameters we also carry out experiments to show how they affect the results. For example, the experiments will confirm the result of Lemma 1 that the running time of phase one is independent of the sampling ratio (α) and the sampling threshold (β) when both these parameters are small.

The real data set we used is the KDDCUP 1999 (KDD) data which we downloaded from the UCI data mining repository[5]. This data

set is a raw TCP dump for a local-area network (LAN) and includes a wide variety of intrusions simulated in a military network environment. The data has already been processed and each record consists of 34 continuous and 7 categorical attributes. Several data sets, of varying size and dimensionality, were created by sampling different number of attributes and rows. For continuous attributes we used the Euclidean distance and for categorical attributes, the Hamming distance was used as the metric.

For synthetic data we used a Gaussian data generator to produce a series of synthetic datasets (SYN) with different size and dimensionality. Each synthetic dataset consists of 20 hyper-spherical clusters, which have radius 0.03 and are composed of points with a normal distribution along each dimension. Outliers are peppered around each cluster between distance 0.04 and 0.06 from the center (of each cluster).

We implemented the algorithm in Java and all our tests were run on a Pentium 4 machine equipped with a 2.53GHz CPU and 1 G main memory. The size of the memory is not crucial as our algorithm is disk-based. We were able to process very large data sets with only 64M memory.

In all our tests, the running time reported is the CPU time plus I/O time. The running time of phase one includes the time of one full data scan, and the total running time includes the time of two full data scans.

5.1 Scalability

We tested the scalability of our algorithm with respect to the size and dimensionality of the data set, and the number of outliers reported. The size of the data set was varied from 100K to 1 million, the dimensionality was varied from 5 to 60 for SYN and 5 to 40 for the KDD data set respectively, and the number of outliers reported was varied from 100 to 500. For these experiments the sampling ratio (α), sampling threshold (β) and the container size (M) were kept fixed.

5.1.1 Size of the Data Set

Figure 2 shows the result of how the running time varies with the size of the data set. Phase one, which is the core of our proposed approach, scales linearly with respect to the data size (KDD_ph1 and SYN_ph1). As the size of the data set increases the ratio of running time of phase one and the total running time (KDD_total and SYN_total) decreases by a factor proportional to the size of the data set. This is because we are using a brute-force approach to test whether the candidate outliers are actual outliers. In fact the running time of phase two is $O(\beta d N^2)$, where β is the sampling threshold, and normally takes a value smaller than 0.01.

In order to decrease the running time of phase two, we must make the value of β as small as possible. This requires a small set of

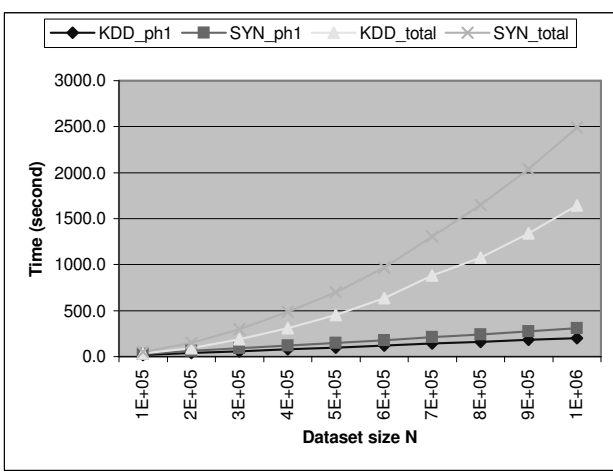


Figure 2: The running time of phase one scales linearly with the size of the data set. The total running time includes the time for outlier verification for candidates generated in phase one.

candidate outliers and later we will discuss how the parameters in phase one can be tuned to achieve this goal and retain high accuracy.

5.1.2 Dimensionality of the Data Set

Figure 3 shows how the running time scales with the dimensionality of the data set. Clearly the running time scales linearly with both phase one (KDD_ph1, SYN_ph1) and the total running time (KDD_total, SYN_total). This is one of the key strengths of our approach. We were able to achieve linear scalability with respect to the dimension of the data set without resorting to a partitioning of the data (feature) space. The only other sampling approach for outlier detection [7], that we are aware of, relies on partitioning the data space into non-overlapping hypercubes after computing a global density estimator function- making it infeasible for high dimension data.

5.1.3 The Number of Outliers

In the last step of **Phase Two**, all the candidates outliers are sorted according to distance and the top n outliers are reported. So the running time is not affected by the number of outliers.

However, Figure 4 shows that the number of outliers does affect the overall accuracy. As the number increases from 100 to 500, the accuracy decreases. We can tune the parameters in **Phase One** to achieve high accuracy when a large number of outliers need to be reported. This will be discussed in a later section.

5.2 Parameter Performance

In this section we evaluate the effect of three parameters: sampling ratio α , sampling threshold β and container size M on the accuracy and running time of the algorithm.

We will exclusively focus on phase one of the algorithm and all running times shown in the remainder of the paper are the times of phase one. In the following experiments, the number of outliers is set to 100, the size of data set is 250K and the dimensionalities are 34 and 60 for KDD and SYN respectively.

5.2.1 Varying Container Size M

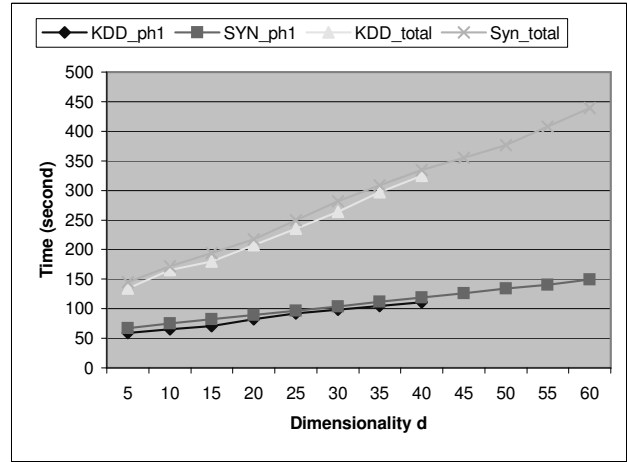


Figure 3: Both the running times of phase one and total scale linearly with the dimensionality

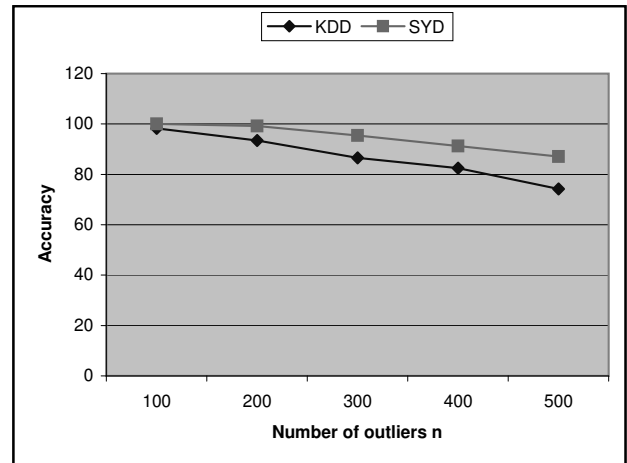


Figure 4: How accuracy scales with the number of outliers output

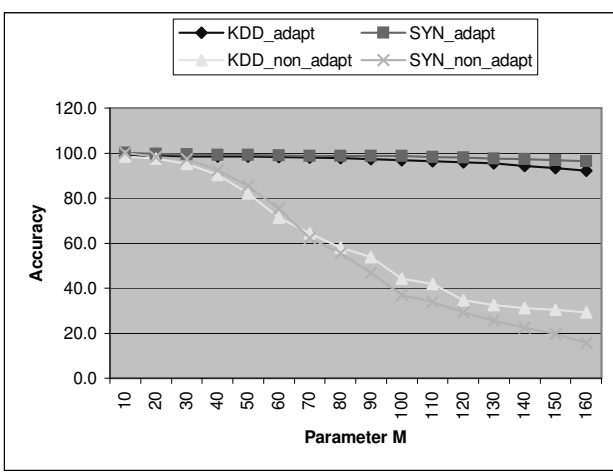


Figure 5: The accuracy decreases very slowly and stays at a high level when M is adaptive. However, the accuracy for non-adaptive M decreases rapidly

In each iteration of successive sampling (phase one), we extract a random sample of size determined by the sampling ratio α , find the M nearest neighbors of each sampled point, sort these containers based on their radii and purge all containers whose radius is less than the median.

Parameter M is very important because it affects both the efficiency and accuracy of the algorithm. We have tested two situations. One is using a constant value for M in all the iterations of phase one (KDD_non_adapt and SYN_non_adapt in Figure 5 and 6). The other is making the value of M adaptive to the size of dataset left after each sampling iteration (KDD_adapt and SYN_adapt in Figure 5 and 6). For example, if in the first step, the size of the data set is 5000 and the value for M is 100, after removing some points, the size of dataset becomes 4000, the parameters M adapts to size 80. The general formula we use is

$$M_{k+1} = \min\left\{10, \frac{U_{k+1}}{U_k} M_k\right\}$$

where U_k is the size of the partition in step k .

Figure 5 shows incredible improvement in accuracy by making the value of M adaptive. As the value of M increases, the accuracy of non-adaptive method decreases very fast, while the accuracy of adaptive method decreases marginally and stays at a high level. This improvement gives us the flexibility of using even a more smaller value for β . This in turn provides precious savings in the running time of phase two, which dominates the total running time, without loss of accuracy.

5.2.2 Varying Parameter α

From Lemma 1, we already known that the running time is independent of small values of the sampling ratio α . The experimental results in Figure 7 confirm this.

Interestingly, the size of α also has little effect on the accuracy of our algorithm. In Figure 8, as the value of α changes, the accuracy of dataset KDD and SYN oscillates around a specific value with a very narrow range. This characteristic of α makes it simple for us to tune our algorithm, as we can set a small value for α .

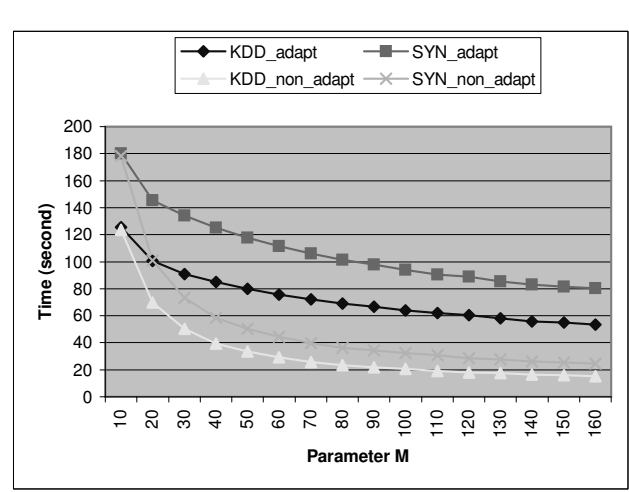


Figure 6: The running time decreases as parameter M increases

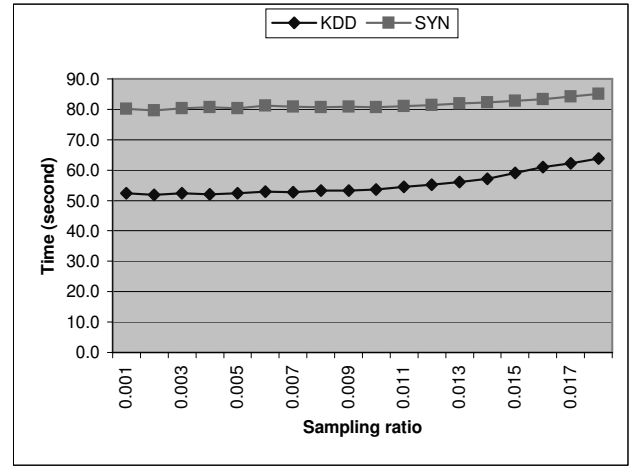


Figure 7: The running time is independent of small values of sampling ratio α . This result validates Lemma 1

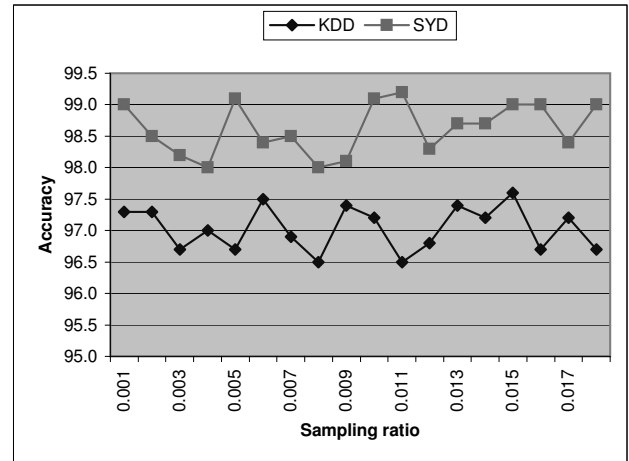


Figure 8: The sampling ratio α has little effect on the accuracy

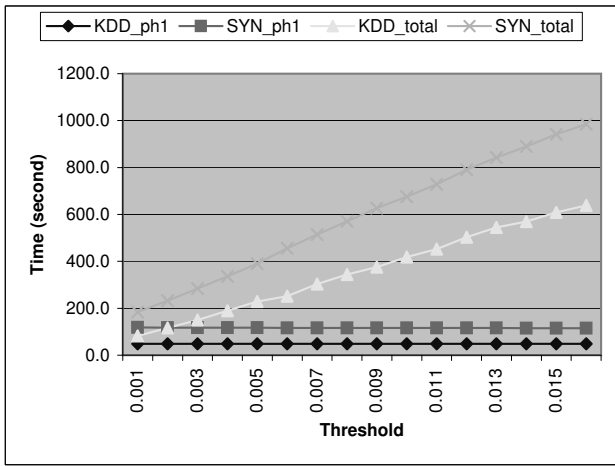


Figure 9: The threshold β has no effect on the running time of phase one when it takes a small value, the total running time scales linearly with β

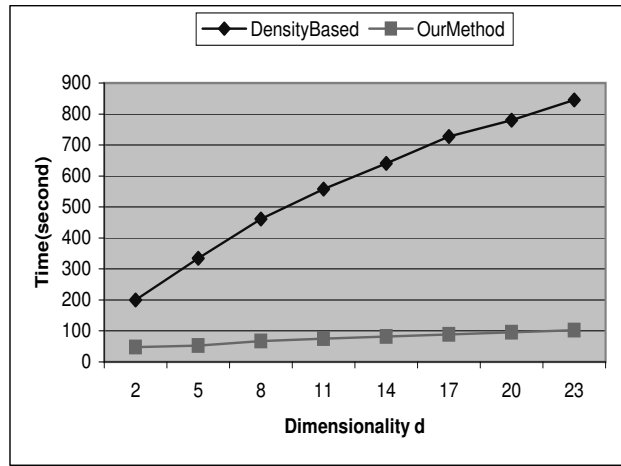


Figure 11: The accuracy increases as the threshold β increases, but when β reaches to a specific value, the improvement of accuracy is not obvious

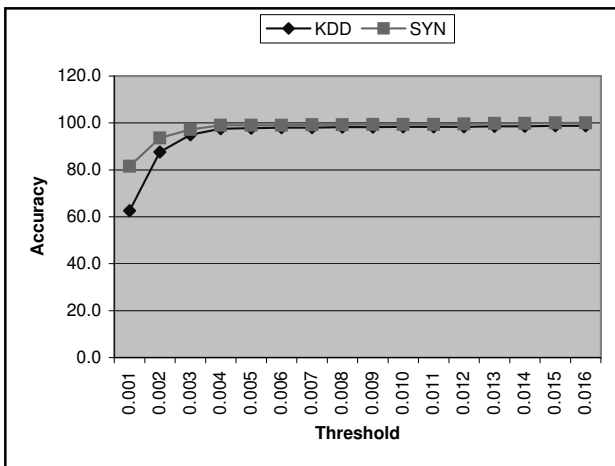


Figure 10: The accuracy increases as the threshold β increases, but when β reaches to a specific value, the improvement of accuracy is not obvious

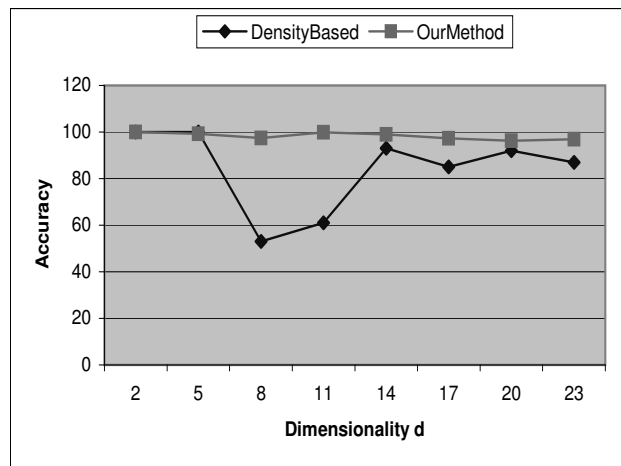


Figure 12: The accuracy increases as the threshold β increases, but when β reaches to a specific value, the improvement of accuracy is not obvious

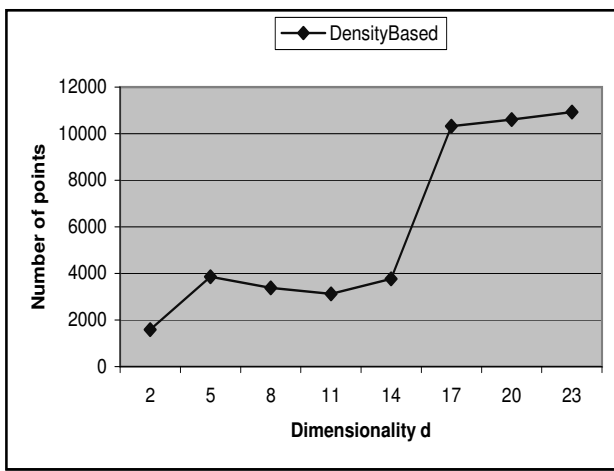


Figure 13: The accuracy increases as the threshold β increases, but when β reaches to a specific value, the improvement of accuracy is not obvious

5.2.3 Varying Parameter β

From Lemma 1, we know that β has no effect on the running time of phase one when it takes on a small value. The experimental results shown in Figure 9 also show this. However, β has strong influence on the total running time, because when we double the value of β , the size of the candidate set is doubled, and as a result, the running time of phase two is also doubled.

β also has some influence on the accuracy. As Figure 13 displays, when β takes on a small value, the accuracy is low for both KDD and SYN datasets. As its value increases, the accuracy also increases. But after it reaches a certain value, e.g. 0.004, the improvement of accuracy is minor, even we double its value. Because when β takes a value of 0.004, the candidate sets are large enough for KDD and SYN to contain near all the outliers.

When more outliers need to be reported, we have to use a larger value for β . Extensive experiments show that when the size of candidate set is 6 to 10 times larger than the number of outliers reported, our algorithm can retain high accuracy. We may refer to this to select a value for β .

When selecting a value for β , an important factor that we may take into account is the distribution of the datasets used. When points in a dataset cluster together and outliers are outstanding, we may use a smaller value for β . On the other hand, if the points scatter the whole space, we have to use a larger value for β in order to gain certain accuracy. However, in most cases, we don't know the distribution of datasets unless we do some pre-analysis. In such situation, the opinion of domain experts is valuable.

6. CONCLUSIONS AND FUTURE WORK

We have presented a sampling-based outlier detection method based on the concept of top n distance-based outliers. This method is especially designed for large (disk-based) high-dimensional datasets. Only two full data scans are required after randomizing the dataset. No partitioning on dimensions is involved, so the problem of sparseness in high-dimensional space is solved intrinsically. Both the complexity analysis and experimental results showed that this method scales well with respect to the size and dimensionality of the datasets.

For future work we would like to provide theoretical guarantees about the accuracy of our proposed method under reasonable conditions. Furthermore we would like to test the accuracy of the sampling approach under proximity-preserving random projections into lower dimensional spaces.

7. REFERENCES

- [1] C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 21-24, 2001, Santa Barbara, California, USA, 2001*.
- [2] F. Angiulli and C. Pizzuti. Fast outlier detection in high dimensional spaces. In *Proceedings of the 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), 2002*.
- [3] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003*.
- [4] D. Hawkins. *Identification of Outliers*. Chapman and Hall, London, 1980.
- [5] 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.
- [6] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB'98, Proceedings of 24th International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA, pages 392-403. Morgan Kaufmann, 1998*.
- [7] G. Kollios, D. Gunopulos, N. Koudas, and S. Berchtold. Efficient biased sampling for approximate clustering and outlier detection in large datasets. *IEEE Transactions on Knowledge and Data Engineering, 2003*.
- [8] R. R. Mettu. *Approximation Algorithm for NP-hard Clustering Problem*. PhD thesis, The University of Texas, 2002.
- [9] R. R. Mettu and C. G. Plaxton. Optimal time bounds for approximate clustering. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence, pages 344-351, August 2002*.
- [10] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA, pages 427-438. ACM, 2000*.
- [11] S. Stolfo, W. Lee, P. Chan, F. Fan, and E. Eskin. Data mining-based intrusion detectors: An overview of columbia ids projects. *SIGMOD Record*, 30(4):5-14, 2001.