Disk-Based Sampling for Outlier Detection in High Dimensional Data

Pei Sun Sanjay Chawla Timothy de Vries Gia Vinh Anh Pham

School of Information Technologies, The University of Sydney, Australia
psun2712@usyd.edu.au, chawla@it.usyd.edu.au, tide2817@usyd.edu.au, vinhanh@gmail.com

Abstract

We propose an efficient sampling based outlier detection method for large high-dimensional data. Our method consists of two phases. In the first phase, we combine a "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. The major strengths of the proposed approach are that (1) no partitioning of the dimensions is required thus making it particularly suitable for high dimensional data and (2) a small sampling set (0.5% of the original data set) can discover more than 99% of all the outliers identified by a full brute-force approach. We present a detailed experimental evaluation of our proposed method on real and synthetic data sets and compare our method with another sampling approach.

1 Introduction and Related Work

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 analysed.

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 [9]. 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.

Statisticians had extensively studied the problem in the context of a given distributional model. The textbook approach [6] 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 of course that for high-dimensional data it may be very hard to pin down a suitable candidate for M.

Knorr and Ng [4] were the first to propose a distance-based 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.

Ramaswamy et. al [5] have proposed the following extension 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 $DB(p, D)$ and DB_n^k definitions several algorithms [1, 3, 4, 5, 12] have appeared in the literature. 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

International Conference on Management of Data
COMAD 2008, Mumbai, India, December 17–19, 2008
©*Computer Society of India, 2008*

outlier, i.e., when more than "p percent of the points lie within a distance D of the point being examined". For the low dimensional data sets, spatial index structures like R^* trees can be used in the nested loop approach. 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. Similarly, partition-based methods suffer from the "curse of high dimensionality". Of particular interest is the paper by Ghoting et. al. [12], which builds on the paper by Bay et. al. [1]. This paper uses divisive hierarchical clustering to effectively partition the data sets into clusters using distance similarity.

One of very few existing sampling based approaches that we are aware of is the biased sampling technique reported by Kollios et. al [2]. 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. We will introduce this method in detail in a later section and compare it with our proposed sampling method.

Another sampling based approach is the work of Wu and Jermaine [11]. The idea is that for each point in the dataset, a random sample set of size α is drawn and the kth-nearest neighbor distance from the point to its sample is calculated. The outliers are reported as the top λ points whose such distance is the greatest. The authors also provide a strategy to speed up distance computation, which is to uniformly replicate the distances computed by their sampling algorithm to create a full-size estimated distance matrix for all data points. Although this algorithm has a linear running time ($\Omega(\alpha n)$), it is often the case that α needs to be large enough to produce an accurate result. The approach described works well for cases where distance computations are prohibitively large.

The rest of the paper is presented as follows. In Section 2 we briefly introduce our contributions. Section 3 is the core of the paper and is devoted to development of the algorithm and its analysis. In Section 4 comprehensive experiment results are reported and discussed. In Section 5, a comparison with another sampling based approach is presented. We conclude in Section 6 with a summary and directions for future work.

2 Our Contributions

In this paper we show that by strategically sampling in the data space, DB_n^k outliers can be identified with high accuracy. The basic idea of our sampling strategy is to sample in stages and at each stage purge regions of high density since they are not likely to contain outliers. What is left are the candidate outliers and another single-pass through the data can be used to extract all legitimate DB_n^k outliers from the candidate set.

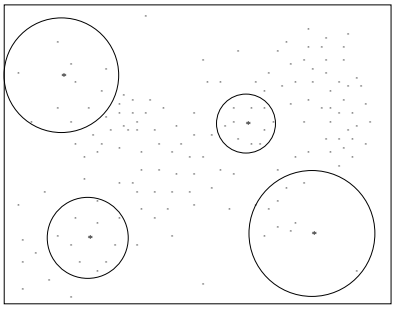


Figure 1: Four sampled points and their top ten nearest neighbors. the sampling and purging process is repeated until the number of points remaining go below a specified threshold. The points left are the candidate outliers.

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. The main idea behind this method is that since all circles contain the same number of points the smaller ones are relatively more densely packed and therefore are less likely to contain outliers.

Specifically, our main contributions are:

1. We propose a sampling strategy based on successive sampling [8] 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.

2. The algorithm is disk-based and, after randomization, requires only two data scans.

3. We have carried out an extensive experimental evaluation, on real and synthetic data sets, to test the scalability, accuracy and dependence on parameters, of our proposed algorithm.

4. We have compared our method with another sampling method proposed in [2] by using real data sets. The results show that our method is better from the points of efficiency and accuracy.

3 Algorithm and Complexity 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 similar to the successive sampling technique introduced by Mettu and Plaxton [8] who use it to design an approximation algorithm for the k-median problem. Phase one 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 in the current data. These hyper-balls are then sorted based on their radii and the smaller hyper-balls, those with radii smaller than the median of the radii in this round, 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.

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 N_p, 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.

Randomizing a file can be done in $O(N)$ time and constant main memory with a disk-based shuffling algorithm [1]. Normally, we can combine this procedure and data preparation together without adding any extra cost. In the complexity analysis section, we don't count the cost of randomization.

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 experiment section we will show how to judiciously set an appropriate value for M. We will demonstrate the effect of adapting M to the size of remaining data set in Section 5.

3.1 Algorithm

The algorithm for *Phase One* is shown in Table 1. 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 $s \in S$, a container C_s of size M is initialized. 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∈S and C_s is maintained such that it contains the M current nearest neighbors of s in U_p. Associated with each C_s is a hyper-ball containing exactly M nearest neighbors of s in U_p. The hyper-balls are sorted based on the radii and those balls whose radii is smaller than the median of the radii in this round 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 βN_p where N_p is the size of the partition. 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 2. 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

3.2 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 3. We have N/N_p partitions (ignoring ceiling function), so the total cost of phase one is:

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

In step k+1, sampling will stop when $(1-\frac{\alpha M}{2})^k N_p = \beta N_p$. Thus:

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

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

$$\frac{2dN_pN}{M}(1+(\beta^{\frac{2}{k}})^1+(\beta^{\frac{2}{k}})^2+...+(\beta^{\frac{2}{k}})^{k-1})(1-\beta^{\frac{1}{k}}) \quad (3)$$

Now, let $S = 1+(\beta^{\frac{2}{k}})^1+(\beta^{\frac{2}{k}})^2+...+(\beta^{\frac{2}{k}})^{k-1}$

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, initialize a container C_s of size M
for each point p(p $\neq$ s) in U_p, assign to it C_s such that $D(p, s)$ is the smallest for each s∈S and C_s is maintained such that it contains the M current nearest neighbors of s in U_p
for each C_s, define a hyper ball, which contains exactly its top M nearest neighbors in U_p 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 1: The Algorithm for Phase One. A sampling strategy to generate the candidate set of outliers.

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 in U;
End for
Sort all the candidates of outliers according to the distances calculated;
Output top n outliers.

Table 2: Algorithm for Phase Two. The candidate set of outliers are examined and only the DB_n^k outliers are retained.

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

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

When β is small (but fixed) and k large we get: $\lim_{k\to\infty} \frac{1-\beta^2}{1+\beta^{\frac{1}{k}}} = \frac{1}{2}$. Thus , the cost of Phase One (formula (3)), becomes $\frac{dN_pN}{M}$, leading to the following lemma.

Lemma 1: *The cost of phase one is* $O(\frac{dN_pN}{M})$ *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 $O(\beta dN^2)$. 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 $O(dN^2)$. Experimental results on real and synthetic data sets in section 4 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.

3.3 Discussion

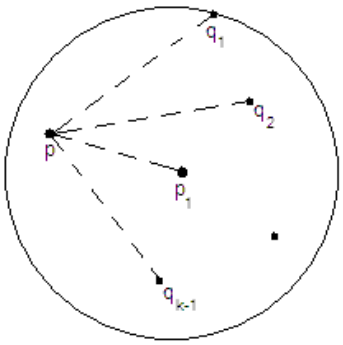
We give a short motivation for the likelihood of Phase 1 purging a true outlier being small in the average case.

Notations:
Let $D^{(t)}$ be the set of points that are left at time t. For each point $p \in D^{(t)}$, let $kNN^{(t)}(p)$ be the set of p-nearest neighbor of p in $D^{(t)}$ and $knn^{(t)}(p)$ be the distance from p to the p^{th}-nearest neighbor of p in $D^{(t)}$

Assumption:
It is natural to assume that an outlier in the original dataset $D(0)$ will have $knn^{(t)}(p)$ significantly large (compared to $knn^{(0)}$ of normal points)

Step	Population Size	Number of Sampled Points	Computational Cost
1	N_p	αN_p	$\alpha d N_p^2$
2	$(1-\frac{\alpha M}{2})N_p$	$\alpha(1-\frac{\alpha M}{2})N_p$	$\alpha(1-\frac{\alpha M}{2})^2 d N_p^2$
...	...	...	...
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 3: The Cost of Each Step in Phase One



Argument/discussion:
Consider an outlier p, then p will be purged if either

1. p is picked at time (t) and $knn^{(t)}(p)$ is small enough compared to $knn^{(t)}$ of the other $M-1$ picked points at time t, or

2. p is in $kNN^{(t)}$ of some picked point at time (t) and the ball corresponding to the picked point is small enough compared to those of other picked points at time t

• **Consider (1):**
We assume $\frac{M}{2}$ of the balls surrounding the M picked points will have their points removed in each step. Suppose the other picked points are $p_1, \ldots, p_{M-1}$, then (1) can happen if and only if $knn^{(t)}(p)$ is less than or equal to at least $\frac{M}{2} knn^{(t)}(p_i)$.

Moreover, $knn^{(t)}(p) \geq knn^{(0)}(p)$

Therefore, at least $\frac{M}{2}$ points in in $\{p_1, \ldots, p_{M-1}\}$ has $knn^{(t)}(p_i) \geq knn^{(0)}(p)$

Based on our assumption that $knn^{(0)}(p)$ is significantly large, this mean there exists at least $\frac{M}{2}$ p_i such that $knn^{(t)}(p_i)$ is significantly large.

Now, suppose p_i is in a cluster (possibly dense one). In order to have $knn^{(t)}(p_i)$ large, we must have removed the majority of p_i's *close* neighbors, and must have not removed p_i before time (t). This is unlikely to happen, and it is even more unlikely to have $\frac{M}{2}$ points for these events. So the probability that (1) can happen is very small.

• **Consider (2):**
Suppose that $p \in kNN^{(t)}(p_1)$ and $knn^{(t)}(p_1)$ and $kNN^{(t)}(p_1)$ is purged. Let $kNN^{(t)}(p_1) = \{p, q_1, \ldots, q_{k-1}\}$. We have

$$knn^{(t)}(p) \leq \max\{d(p,p_1), d(p,q_1), \ldots, d(p,q_{k-1})\} \leq 2knn^{(t)}(p_1)$$

Also

$$knn^{(t)}(p) \geq knn^{(0)}(p)$$

Therefore $knn^{(t)}(p_1) \geq \frac{knn^{(t)}(p)}{2}$

Using a similar argument as when considering (1), we also see that this event is very unlikely to happen.

In summary, the probability that Phase one purges an outlier is very small.

4 Experimental Results

In this section we carry out very relative 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 of 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 [7]. 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 attributes. Several data sets, of varying size and dimensionality, were created by sampling different number of attributes and rows.

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 with outliers peppered around 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.

4.1 Scalability

We tested the scalability of our algorithm with respect to the size and dimensionality of the data set. The size

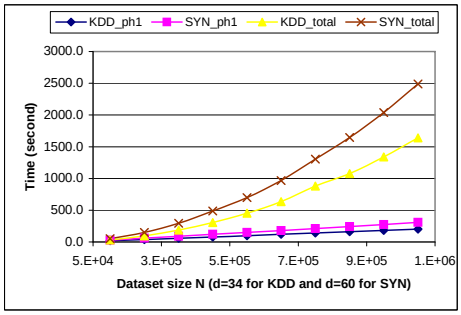


Figure 2: The running time of *phase one* scales linearly with the size of the data set. The total running time includes the time of outlier verification

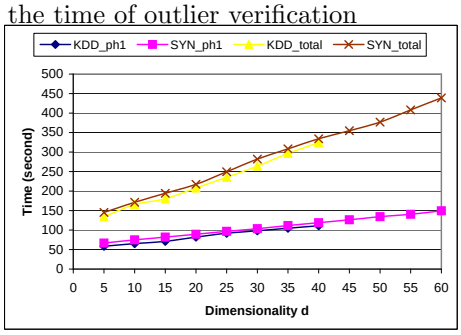


Figure 3: Both the running times of phase one and total scale linearly with the dimensionality

of the data set was varied from 100K to 1 million, the dimensionality was varied from 5 to 60 for SYN and 5 to 34 for the KDD data set respectively. For these experiments the sampling ratio (α), sampling threshold (β) and the initial container size (M) were kept fixed as 0.005, 0.005 and 20 respectively.

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 $O(\beta dN^2)$, 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 candidate outliers and later we will discuss how the parameters in phase one can be tuned to achieve this goal and retain high accuracy.

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.

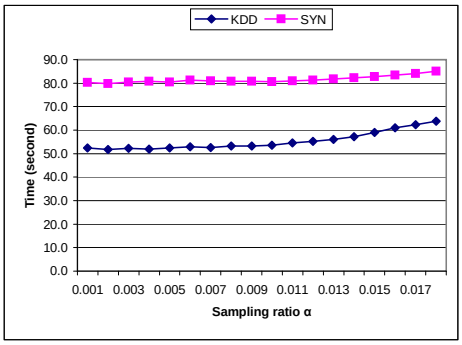


Figure 4: The running time is independent of small values of sampling ratio α. This result validates Lemma 1

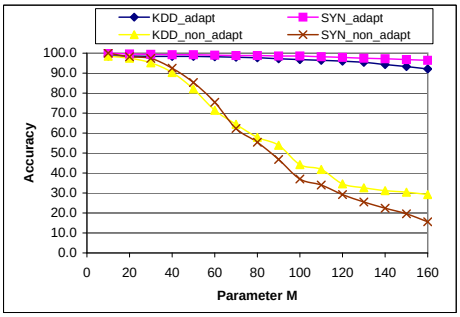


Figure 5: The accuracy decreases very slowly and stays at a high level when M is adaptive. However, the accuracy for non-adaptive M decreases rapidly

4.2 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 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.

4.2.1 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 4 confirm this. Interestingly, the size of α also has little effect on the accuracy of our algorithm. In Figure 7, 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.

4.2.2 Varying Container Size M:

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 initial value of M is set such that no more than 10 percent of points are removed in the first iteration. The general formulae we use are

$$M_0 = \frac{0.2}{\alpha}, \qquad M_{k+1} = \max\{10, \frac{U_{k+1}}{U_k} M_k\}$$

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

Figure 5 shows significant improvement in accuracy by making the value of M adaptive. As the value of M increases, the accuracy of non-adaptive method decreases rapidly, while the accuracy of adaptive method decreases marginally and stays at a high level. This improvement gives us the flexibility of using even a 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.

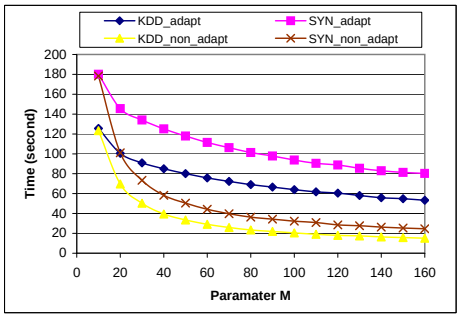


Figure 6: The running time decreases as parameter M increases

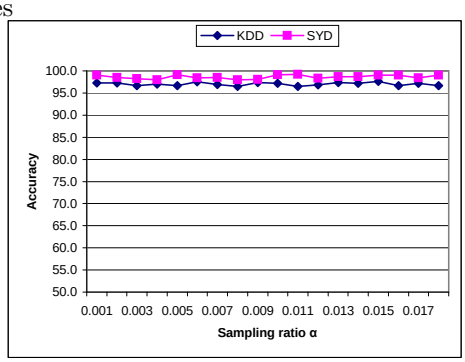


Figure 7: The sampling ratio α has little effect on the accuracy

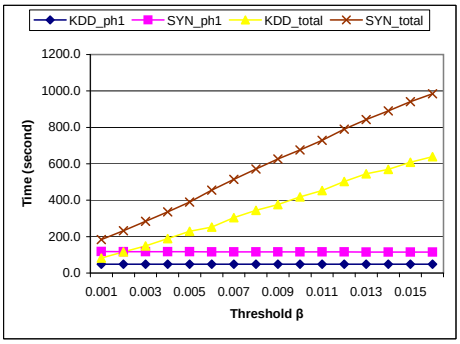


Figure 8: The threshold β has no effect on the running time of phase one when it takes a small value, the total running time scales linearly with β

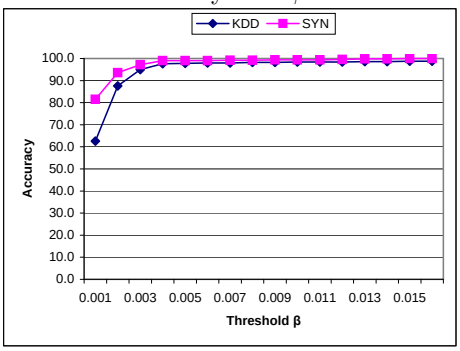


Figure 9: The accuracy increases as the threshold β increases, but when β reaches to a specific value, there is no significant improvement

4.2.3 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 8 also illustrate 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.

The parameter β also has some influence on the accuracy. As Figure 9 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 in 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 nearly all of 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.

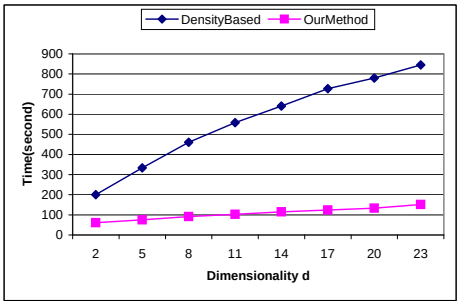


Figure 10: The running times of our method and density-based method

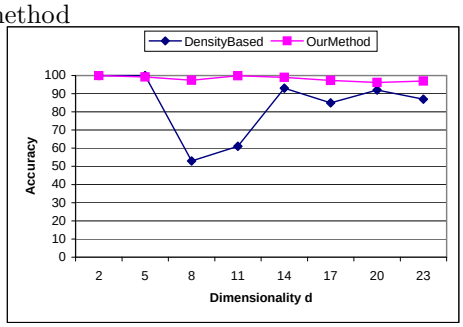


Figure 11: The accuracy of our method and the density-based method

5 Comparison with a Density-based Sampling Method

In this section, we compare our method with the one proposed in [2], which uses some sample points to build a density estimator based on the Epanechnikov kernel function, and then selects candidates for outliers according to this estimator. The functionality of this algorithm is similar to our method since it also selectively chooses low density regions to inspect with greater detail. Points with low density value are natural candidates for outliers. Their key contribution is the use of the biased sampling technique as opposed to random sampling, once the estimator is built.

In [2], the procedure of outlier detection is also divided into two phases, and the second phase is exactly the same as ours. They use the definition of DB(p,D) outlier. In order to compare easily, we modified their method slightly to solve the DB_n^k outlier problem as follows:

Phase One

• First sample some points to build the density model (This step is the same as original method);
• For each point in the dataset, calculate the density value by using the kernel density model;
• Sort all the points by density value, and use the points with a small density value as candidates.

Phase Two

• Verify the candidates.

The dataset used for comparison is the KDD CUP1999 data as described in section 4, with 494,000 points. The number of points used for building the density model is 1000 as recommended in [2]. If we use more points then phase one takes more time.

5.1 Efficiency

Figure 10 compares the running times of Phase one of the density-based and our method. Both methods produce the same number of candidates, so in phase two, both methods will need the same amount of time to prune the candidates to get the top n outliers. It is clear that our method uses less time. Both running times change linearly with the dimensionality but the density-based method has a higher rate of increase (bigger slope).

5.2 Accuracy

Figure 11 shows how accuracy varies with the dimensionality. Both methods work well when the dimensionality is low. However, when the dimensionality increases, the accuracy of our method continues to remain at a high level with little decline. However, the accuracy of the density-based method becomes very unstable.

6 A method for reducing false outliers in the candidate set

6.1 Analysis

The idea of having Phase two in the algorithm is that we would like to remove all false outliers (false positives) that remain after purging in Phase one by validating each remaining point. Often, however, the number of false outliers in this candidate set is quite large. Therefore a reduction in the size of this set will greatly speed up the polynomial-time Phase two stage, especially for larger datasets.

We observe that if a point p is in a cluster C then the probability that p is reported to be an outlier when it is not an outlier (false positive case) is very similar to any other point in the same cluster C. Therefore the false positives in the candidate set resulting from Phase one are somewhat randomly picked. We can exploit this fact if the running time of Phase two is much larger than of Phase one, as is the case in larger datasets, by running Phase one multiple times and taking the intersection of the resulting candidate outlier sets. Because the false positives in the candidate set tend to be randomly selected, the false positives in the two candidate sets resulting from the running of Phase one twice tend to have only a small amount of data points in common, leading to a large reduction in false positives if the intersection of the two candidate sets are taken. Of course, since each running of Phase one is not guaranteed to find all top outliers, it is more likely that a true outlier is dropped in one

running of Phase one and therefore dropped from the final intersected set if this method is used. However, in our experiments we found that this case was very rare.

The following lemmas show the expectation and variance of the number of false positives in the intersection set. For simplicity, we consider the case where our data has one cluster with a few distinct outliers, and Phase one is run two or three times. A similar analysis can be carried out for more complicated datasets or with more than three runs of Phase one.

Lemma 2 (For two runs of Phase one): *Given set $S = \{1, 2, \ldots, n\}$. We randomly picked $S_1, S_2 \subset S$ such that $|S_1| = |S_2| = m$. Then*

1. $E[|S_1 \cap S_2|] = \frac{m^2}{n}$
2. $Var[|S_1 \cap S_2|] = \frac{m^2(m-1)^2}{n(n-1)} + \frac{m^2}{n}$

We have

$$E[|S_1 \cap S_2|] = \sum_{i=0}^{m} iP(|S_1 \cap S_2| = i)$$

Now, for each set $T \subset S$ such that $|T| = i$, the number of pairs $(S_1, S_2) \subset S \times S$ such that $S_1 \cap S_2 = T$ is equal to the number of ways we can partition the set $S \backslash T$ into two subsets of equal size $m - i$, which is

$$\left(\begin{array}{c} n-i \\ 2(m-i) \end{array} \right) \left(\begin{array}{c} 2(m-i) \\ m-i \end{array} \right)$$

The number of such T set is $\left(\begin{array}{c} n \\ i \end{array} \right)$ and the number of possible pairs (S_1, S_2) such that $|S_1| = |S_2| = m$ is $\left(\begin{array}{c} n \\ m \end{array} \right)^2$. Therefore

$$P(|S_1 \cap S_2| = i) = \frac{\left(\begin{array}{c} n \\ i \end{array} \right) \left(\begin{array}{c} 2(m-i) \\ m-i \end{array} \right) \left(\begin{array}{c} n-i \\ 2(m-i) \end{array} \right)}{\left(\begin{array}{c} n \\ m \end{array} \right)^2}$$

Now,

$$E[|S_1 \cap S_2|] = \sum_{i=0}^{m} i. \frac{\left(\begin{array}{c} n \\ i \end{array} \right) \left(\begin{array}{c} 2(m-i) \\ m-i \end{array} \right) \left(\begin{array}{c} n-i \\ 2(m-i) \end{array} \right)}{\left(\begin{array}{c} n \\ m \end{array} \right)^2}$$

$$= \frac{1}{\left(\begin{array}{c} n \\ m \end{array} \right)^2} \left(\sum_{i=1}^{m} i \frac{n!}{i!(n-i)!} \frac{(n-i)!}{(n-i-2(m-i))!(2(m-i))!} \frac{(2(m-i))!}{((m-i)!)^2} \right)$$

$$= \frac{n!}{\left(\begin{array}{c} n \\ m \end{array} \right)^2} \left(\sum_{i=1}^{m} \frac{1}{(i-1)!(n-2m+i)!((m-i)!)^2} \right)$$

$$= \frac{n!}{\left(\begin{array}{c} n \\ m \end{array} \right)^2 (m-1)!(n-m)!} \left(\sum_{i=1}^{m} \left(\begin{array}{c} m-1 \\ i-1 \end{array} \right) \left(\begin{array}{c} n-m \\ m-i \end{array} \right) \right)$$

We have $\sum_{i=1}^{m} \left(\begin{array}{c} m-1 \\ i-1 \end{array} \right) \left(\begin{array}{c} n-m \\ m-i \end{array} \right)$ is the coefficient of x^{m-1} in

$$\left(\sum_{i=1}^{m} \left(\begin{array}{c} m-1 \\ i-1 \end{array} \right) x^{i-1} \right) \left(\sum_{j=0}^{n-m} \left(\begin{array}{c} n-m \\ j \end{array} \right) x^j \right)$$
$$= (1+x)^{m-1}(1+x)^{n-m} = (1+x)^{n-1}$$

Thus,

$$\sum_{i=1}^{m} \left(\begin{array}{c} m-1 \\ i-1 \end{array} \right) \left(\begin{array}{c} n-m \\ m-i \end{array} \right) = \left(\begin{array}{c} n-1 \\ m-1 \end{array} \right)$$

$$E[|S_1 \cap S_2|] = \frac{n!}{\left(\begin{array}{c} n \\ m \end{array} \right)^2 (m-1)!(n-m)!} \left(\begin{array}{c} n-1 \\ m-1 \end{array} \right) = \frac{m^2}{n}$$

Now

$$Var[|S_1 \cap S_2|] = \sum_{i=0}^{m} i^2 P(|S_1 \cap S_2| = i)$$
$$= \sum_{i=0}^{m} iP(|S_1 \cap S_2| = i) + \sum_{i=0}^{m} i(i-1)P(|S_1 \cap S_2| = i)$$
$$= E[|S_1 \cap S_2|] + \sum_{i=0}^{m} i(i-1)P(|S_1 \cap S_2| = i)$$

Using similar techniques as when calculating $E[|S_1 \cap S_2|]$, we have

$$\sum_{i=0}^{m} i(i-1)P(|S_1 \cap S_2| = i) = \frac{m^2(m-1)^2}{n(n-1)}$$

Therefore,

$$Var[|S_1 \cap S_2|] = \frac{m^2(m-1)^2}{n(n-1)} + \frac{m^2}{n}$$

Lemma 3 (For three runs of Phase one): *Given set $S = \{1, 2, \ldots, n\}$. We randomly picked $S_1, S_2, S_3 \subset S$ such that $|S_1| = |S_2| = |S_3| = m$. Then*

$$E[|S_1 \cap S_2 \cap S_3|] = O\left(\frac{m^2}{n^2} + \frac{m^4}{n^3} \right)$$

Note that the scalability for finding the intersection result is $\min\{O(n), O(m \log m)\}$ because:

1. If we map each set to a bitmap vector $S \mapsto (x_1, x_2, \ldots, x_n) : x_i = 1$ if $i \in S, 0$ otherwise then $|S_1 \cap S_2|$ is the sum $\sum_{i=1}^{n} x_i y_i$ where $(x_i), (y_i)$ is the bitmap representation of S_1, S_2 respectively (this takes $O(n)$ time).
2. If we sort each set then we can find intersection by allocating a pointer to each sorted set (thus the complexity is $O(m \log m)$.

In practice, the size of the candidate set is usually much smaller than n, so the time required to carry out the intersection is negligible. When this approach to reduce false outliers is applied between Phase one and Phase two of our approach, the number of points being taken as input to the polynomial-time Phase two is minimised. This therefore greatly reduces the overall runtime in cases where n is large.

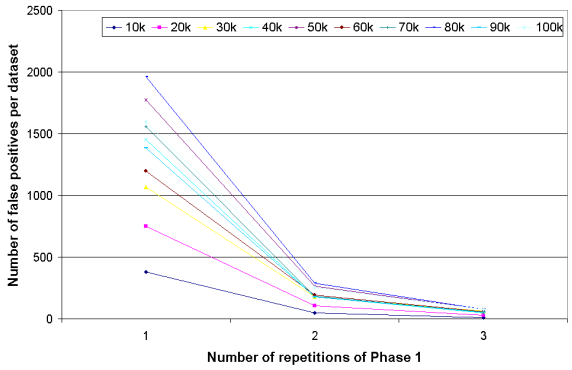


Figure 12: False positive reduction after repeating Phase one and intersecting the result. The lines shown correspond to datasets of varying size. The size of each dataset is given in the legend, where for example 10k means that the dataset contains 10,000 points. From this graph it can be clearly seen that in all cases the reduction is of exponential magnitude after each run and intersection.

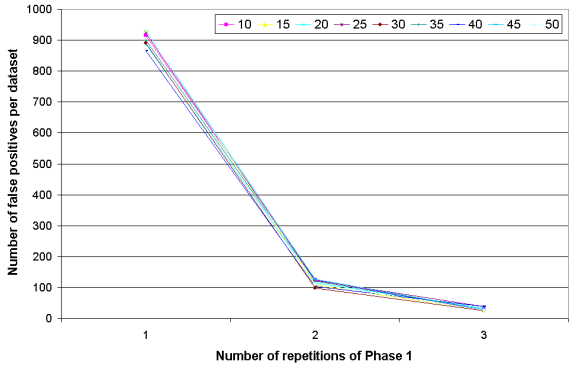


Figure 13: False positive reduction after repeating Phase one and intersecting the result. The lines shown correspond to datasets of varying dimensionality. The dimensionality of each dataset is given in the legend. This graph shows that the reduction in the number of non-outliers remains of an exponential magnitude independent of the dimensionality.

Therefore, when this additional step is applied the running time of the overall algorithm becomes near linear in complexity in most cases where the parameters can be suitably chosen. In some datasets, however, this requirement of linear running time can force a choice of parameter values which can cause a significant loss in accuracy. A tradeoff between desired runtime complexity and accuracy can be made.

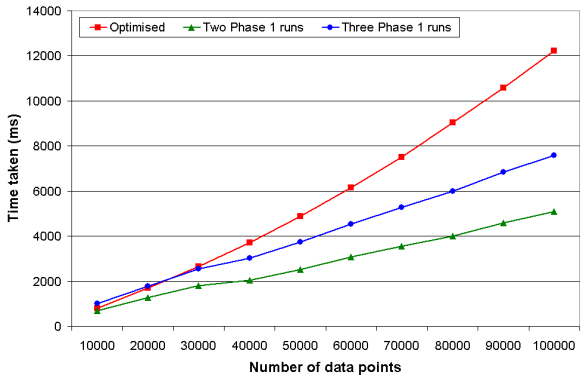


Figure 14: Runtime for varying dataset size. Phase 2 is not included in these results. Our results for two and three runs of Phase 1 plus the intersection of the candidate set is shown compared to the Bay-Schwabacher algorithm.

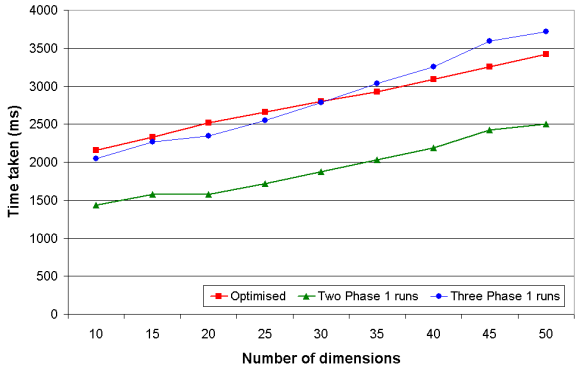


Figure 15: Runtime for varying dataset dimensionality. Phase 2 is not included in these results. Our results for two and three runs of Phase 1 plus the intersection of the candidate set is shown compared to the Bay-Schwabacher algorithm.

6.2 Experimental results

In this section, we compare our result with the Bay-Schwabacher algorithm which has near-linear runtime [1]. We ran experiments using randomly generated datasets with a single cluster containing n data points plus outliers. We varied n from 10,000 to 100,000 while keeping the dimensionality set to 10, and then varied the dimensionality from 10 to 50 while keeping n set to 25,000. The other parameters of sampling ratio (α), sampling threshold (β) initial container size (M), and partition size were kept fixed as 0.001, 0.005, 50 and 5000 respectively. Figure 12 and Figure 13 show the magnitude of the false positive reduction after carrying out the intersection of the candidate sets of multiple Phase one runs, for varying N and D respectively. In all of our experiments we experienced no cases of legit-

imate outliers being missed (false negatives), as well as no true outliers being lost in the intersection process.

Figures 14 and 15 show the runtime of our experiments over varying dataset size and dimensionality, compared to the Bay-Schwabacher algorithm. Our results show the runtime for running Phase 1 two and three times, intersecting the candidate set between each run.

While it is difficult to find when the process of repeatedly running Phase 1 and intersecting the candidate sets will converge to a solution, in practice the largest benefit occurs only after a few runs, after which Phase 2 can be run on the much smaller resulting candidate set to find the final solution. We carried out an experiment to calculate the expected accuracy loss in cases where it is difficult to distinguish between a legitimate outlier and a member of the cluster.

7 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.

References

[1] Bay, S. D., Schwabacher, M.: Mining Distance-Based Outliers in Near Linear Time with Randomization and a Simple Pruning Rule. Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, New York, NY, USA (2003) 29-38

[2] Kollios, G., Gunopulos, D., Koudas, N., Berchtold, S.: Efficient Biased Sampling for Approximate Clustering and Outlier Detection in Large Datasets. IEEE Transactions on Knowledge and Data Engineering 15(5), (2003) 1170–1187

[3] Angiulli, F., Pizzuti, C.: Fast Outlier Detection in High Dimensional Spaces. Proceedings of the 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), London, UK (2002) 15-26

[4] Knorr, E. M., Ng, R. T.: Algorithms for Mining Distance-Based Outliers in Large Datasets. Proceedings of 24rd International Conference on Very Large Data Bases, New York, USA (1998) 392-403

[5] Ramaswamy, S., Rastogi, R., Shim, K.: Efficient Algorithms for Mining Outliers from Large Data Sets. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, Texas, USA (2000) 427-438

[6] Hawkins, D.: Identification of Outliers. Chapman and Hall, London (1980)

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

[8] Mettu, R. R., Plaxton, C. G.: Optimal Time Bounds for Approximate Clustering. Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (2002) 344–351

[9] Lazarevic, A. Ertöz, L., Kumar, V., Ozgur, A., Srivastava, J.: A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection. Proceedings of SIAM International Conference on Data Mining (2003).

[10] Wang, X., Hamilton, H.: DBRS: A Density-Based Spatial Clustering Method with Random Sampling. Proceedings of the 7th PAKDD, Seoul, Korea (2003) 563-575.

[11] Wu M. and Jermaine C.: Outlier detection by sampling with accuracy guarantees. KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. Philadelphia, PA, USA (2006) 767–772

[12] Ghoting A., Parthasarathy S., Otey M.: Fast mining of distance-based outliers in high dimensional datasets. Proceedings of SIAM International Conference on Data Mining (2006).