A Fast Randomized Method for Local Density-based Outlier Detection in High Dimensional Data

Minh Quoc Nguyen, Edward Omiecinski, and Leo Mark

College of Computing,
Georgia Institute of Technology,
Atlanta, GA 30332, USA
{quocminh,edwardo,leomark}@cc.gatech.edu

Abstract. Local density-based outlier (LOF) is a useful method to detect outliers because of its model free and locally based property. However, the method is very slow for high dimensional datasets. In this paper, we introduce a randomization method that can computer LOF very efficiently for high dimensional datasets. Based on a consistency property of outliers, random points are selected to partition a data set to compute outlier candidates locally. Since the probability of a point to be isolated from its neighbors is small, we apply multiple iterations with random partitions to prune false outliers. The experiments on a variety of real and synthetic datasets show that the randomization is effective in computing LOF. The experiments also show that our method can compute LOF very efficiently with very high dimensional data.

1 Motivation

Recently, several different methods for outlier detection [6] have been presented. We can roughly categorize the methods into parametric and nonparametric. The nonparametric methods have a great advantage over the parametric methods is that they do not require prior knowledge of the processes that produce the events (e.g. data distribution). These methods can further be categorized into globally based and locally based. The globally based methods [10] identify the observations that are considered to be the top outliers with respect to distance for the entire dataset. Breunig et al [5] introduced a local density-based method (LOF) to detect local outliers with respect to its neighbor. The concept of local outliers show to be very useful [6] [18] [15] [16] for two reasons. First, it is because, in practice, an observation is an outlier due to its deviation from its locally similar observations rather than the entire dataset. Second, it can detect outlier without requiring any statistical model assumption.

However, the k-nearest neighbors need to be computed for the LOF method. The time complexity is $O(N^2)$ for a data set of size N. This is expensive. In very low dimensional data, one may use indexing methods to speedup the nearest neighbor searches, namely R*-tree [3], X-tree [4], Kd-tree [11], etc. The main

idea of the indexes is to create a hierarchical tree of boundaries of the points in it. The index trees scale well with n but the number of boundaries exponentially increases with the number of dimensions. In fact, Bay and Schwabacher [2] showed that the performance of index trees is worse than the brute force search for more than 20 dimensions. Therefore, even though LOF is very useful method, it remains a challenge to be used in practice for large and high dimensional datasets [6] [18].

In this paper, we present a randomization method to compute LOF efficiently for datasets with very high dimensionality. To the best of our knowledge, we are the first who evaluate a method that can compute LOF very efficiently in very high dimensions up to 500 dimensions.

The method is made possible by our observation of the outlier consistency property of local outliers (which will be discussed in the formalism section). As a result, we can employ a randomization method to compute LOF. From now on, we will refer to the original version of LOF with full k-nearest neighbor search as the **nonrandomized version** of LOF. In the following sections, we will formally define the randomized method. In the experiment section, we will evaluate the effectiveness and efficiency of the **randomized version** against the nonrandomized version of LOF.

2 Related Work

Outliers have been studied extensively in the field of statistics [12] by computing the probability of an event against its underlying distribution. However, this method requires prior knowledge about the process that produces the events, which is usually unknown. Knorr et al [10] introduce a distance-based method to identify the outliers. The outliers are those points whose distance to other observations are the largest. Their method can detect global outliers. The advantage of this method is that no prior knowledge about the underlying distribution is required. Breunig et al [5] introduce a local density based method for outlier detection. An outlier is a point that deviates from its neighbors. The outlier local factor is measured by the ratio of its distance to its neighbors and the local density. Spiros et al [22] introduce the method to detect outliers by using the multi-granularity deviation factor (MDEF). The authors then propose an approximate version to speed up the method. The method is based on the modification of an approximate nearest neighbor search algorithm (quad-tree) in order to avoid the cost of computing the MDEF scores for all the points in the dataset. Thus, the method depends on the performance of the index tree. Recently, Kriegel et al [15] propose the angle-based method that computes outlier scores based on the angles of the points with respect to other points. The method aims to provide more accurate rankings of the outliers in high dimensions. However, the method can not detect outliers surrounded by other points. The naive implementation of the algorithm runs in $O(n^3)$. Bay and Schwabacher [2] introduce a randomize method to detect distance-based outlier. However, their method can not be used for density-based outlier.

3 Local Outlier Factor

We revisit the concept of local density-based outlier introduced by Breunig et al [5]. The local density-based outlier is based on the k-nearest neighbor distance, the local reachability and local outlier factor which are formally defined as follows:

Definition 1 (k-distance of an object p). *For any positive integer k, the k-distance of object p, denoted as k-distance(p), is defined as the distance $d(p, o)$ between p and an object $o \in D$ such that:*

- *for at least k objects $o' \in D \setminus \{p\}$ it holds that $d(p, o) \leq d(p, o)$, and*
- *for at most k-1 objects $o' \in D \setminus \{p\}$ it holds that $d(p, o) < d(p, o)$.*

Definition 2 (k-distance neighborhood of an object p). *Given the k-distance of p, the k-distance neighborhood of p contains every object whose distance from p is not greater than the k-distance, i.e. $N_{k-distance(p)}(p) = \{q \in D \setminus \{p\} | d(p, q) \leq kdistance(p)\}$.*

Definition 3 (reachability distance of an object p w.r.t. object o). *Let k be a natural number. The reachability distance of object p with respect to object o is defined as $reach - dist_k(p, o) = \max\{k - distance(o), d(p, o)\}$. These objects q are called the k-nearest neighbors of p.*

Definition 4 (local reachability density of an object p). *The local reachability density of p is defined as*

$$lrd_{MinPts(p)} = 1/\frac{\sum_{o \in N_{MinPts}(p)} reach - dist_{MinPts}(p, o)}{|N_{MinPts(p)}|}$$

Definition 5 (local outlier factor of an object p). *The (local) outlier factor of p is defined as*

$$LOF_{MinPts}(p) = \frac{\sum_{o \in N_{MinPts(p)}} \frac{lrd_{MinPts(o)}}{lrd_{MinPts(p)}}}{|MinPts(p)|}$$

Intuitively, the local outlier factor (LOF) of p is the ratio between the average local reachability of its neighbors and its local reachability. If p is in a deep cluster, the local outlier factor is close to 1. If p is outside the clusters, it is greater than 1. The local outlier factor measures the degree of local deviation of p with respect to its neighbors.

4 Generalized Local Density-based Outlier

We observe that the main idea of local outlier factor is in fact similar to the computation of the ratio between the distance from p to its nearest points with the density of its local subspace in order to identify local outliers. Breunig et al

measure the local density by using the average k-distance of the nearest neighbors of p. This metric, however, can be generalized to other local density function without affecting the meaning of local density-based outlier. A reasonable choice can be kernel density function. We say that S is approximately uniform if the following two conditions hold. The variance of the k-nearest distances is less than a small ϵ and there is no k-nearest distance is larger than the average k-nearest distance with ϵ unit for some k. In this study and in the following theorems, we measure the local density by the average closest distance between the points in S ($density(S)$).

We also observe that if the distance of p to its nearest points is much greater than the density of any subset in D ($dist(d, S)$) that is approximately uniform, p is not in any cluster. Clearly, p is an outlier in D. On contrary, if there is a subset S' such that the difference is small, p is likely to be generated from the same distribution of S'. We can not conclude that p is an outlier, so p is considered to be normal. These two observations lead to the conclusion that the ratio between the distance and the density must be high for all the subsets in the dataset for a point to be an outlier. Thus, we can define a local density-based outlier as follows:

Definition 6. *Given a point p, a dataset D, and for any subset S of D such that S_i is approximately uniform, p is an outlier with respect to D iff* $\frac{dist(p,S)}{density(S)} \gg 1$.

Figure 1 illustrates two outliers p_1 and p_2 based on the definition. In the figure, p_1 is not only a local outlier for the cluster containing S_1, but p_1 is also an outlier with respect to S_2, S_3, and S_4. Similarly, p_2 is also an outlier with respect to S_1. By this definition, we observe that if we take a random hyperplane to partition

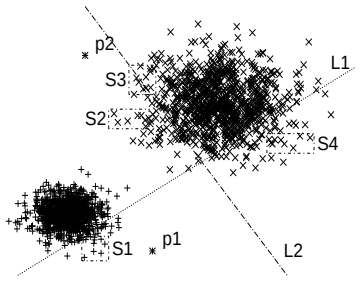


Fig. 1: Outliers with respect to their local subspaces.

a data set into two subset. In most of the cases, the local outlier factors will not change dramatically. Then, we can recursively partition the subsets into smaller subsets. We can partition the data set until the subsets are small enough for us to compute the local outlier factors efficiently. As we see, we do not need to perform the nearest neighbor computation for the entire dataset in order to detect the local outliers.

Figure 1 illustrates an example of the partition. L_1 and L_2 partition the dataset. $S_1 \dots S_4$ are unchanged after the partitions. L_2 cuts S_3 into two subspaces S_3' and S_3''. We see that S_3' and S_3'' are still approximately uniform after the partition. The points p_1 and p_2 remain to be outliers in the new subsets partitioned by L_1 and L_2.

The procedure to detect the local outliers assumes that the partition does not affect the density for the partitioned sets. There are two cases where it can go wrong. The first case is when a point q is on a cluster boundary and the partition isolates it from the cluster it belongs to. If the distance between local subspace of q and the new cluster in the subset it belongs to is large, q is incorrectly identified as an outlier with respect to the new clusters. The second case is when there are many points like q that are separated from their clusters. It may make an outlier p to be normal in the new subset contains only these points.

These problem in fact can be avoided if during the separation, the new subsets contain enough neighbors of these points. Fortunately, it can be shown that the probability of partitions that separate an normal point from all of their neighbors is small. It is because that if a set C which contains q (on the cluster boundary) is large, then the probability of drawing a hyperplane cutting C such that it only contains q is small.

Theorem 1. *Given a point p in a set of size N, the probability of selecting k nearest neighbors of p or less is k/N.*

Proof. The probability to choose a value k is $1/N$. Thus, the probability to choose up to k nearest neighbors is $\sum_{i=1}^{k} \frac{1}{N} = \frac{k}{N}$.

If p is not an outlier, it should belong to a cluster. It implies that $k \ll N$. The theorem shows that the probability of a point p on the boundary to be separated its cluster is small. This is an important observation because we can detect the local outliers effectively using randomization. If we partition the dataset randomly multiple times, in most partitions, q will appear to be normal. Thus, if a point appears to be normal in most partitions, we can flag it as normal with high confidence.

We can illustrate this using figure 1. St will be rare for the small group of points S_1 to be always separated from its cluster using random partitioning.

The observations above are the principles of the randomized method for computing outliers by randomly partitioning a dataset and running the algorithm multiple times so that the false outliers can be ruled out.

The discussions above are based on the assumption that the local subsets are approximately uniform. Practically, data sets do not usually contain uniform subsets. However, this does not affect the randomization method. In the section above, we discuss the definition based the density of the local set but there is no requirement about the size of the set. The subsets do not have to be large for the definition to be correct. In fact, S can be any size of at least two and the definition is still applied. Therefore, we can consider any data set as a set of approximately uniform subsets.

5 Algorithm

The randomized algorithm is described in Algorithm 1. In this algorithm, PARTITION (Algorithm 2) takes a dataset D as an input. Then, it will call SPLIT to split the dataset into two subsets S_1 and S_2 in the following way. SPLIT randomly selects two points p_1 and p_2 in D. For every point in D, SPLIT computes the distance from it to p_1 and p_2. D will be split into S_1 and S_2 where S_1, S_2 contain all the points that are closer to p_1, p_2 respectively. This SPLIT is equivalent to choosing a hyperplane P to partition the dataset. Then, for $S \in \{S_1, S_2\}$, if the size of S is still greater than a threshold M_θ, PARTITION will be applied to S. This recursive PARTITION will be performed until the size of the result sets are smaller than a chosen size of M_θ. At this point, the LOF for all the points in S will be computed with respect to S. M_θ should be greater the parameter $MinPts$ of LOF. Other than that, it can be any value that allows the outlier detection can be computed efficiently. In the end, we will have all the outlier

Algorithm 1 COMPUTEOUTLIER($Set\ D, N\_ITER$)

```
for all i ∈ [1, N_ITER] do
    PARTITION(D)
end for
for all i ∈ [1, N_ITER] do
    COMBINESCORES
end for
```

scores for D. As discussed in section 4, the result set of outliers may contain false outliers due to isolated points. Therefore, we run PARTITION multiple times to rule out the false outliers. The final LOF for each point will be its minimum score of all the iterations. We use the parameter M_{iter} to set the number of iterations of the algorithm. According to the experiments, the output tends to be stable with $M_{iter} = 10$. We can speed up the algorithm by filtering points with low scores that are less than a threshold δ_{out}. The points with the scores computed in the first few iteration less than δ_{out} will not be considered in the next iterations.

It is expected that there will always be some small differences in the rankings between the original method and the randomized method. In the original LOF method, the ranking depends on $MinPts$. The choice of $MinPts$ is subjective. A small change in $MinPts$ will lead to a change in the ranking by LOF. Therefore, it is acceptable for the ranking to be slightly different. In the case, that a more similar LOF ranking is desired, we can recompute the outlier scores for the top N outliers by using the original nonrandomized version. It will give the exact score for these points. The number of top outliers is small, thus the computation time is fast. We call this version the **recompute version** of the randomized method, while we call the earlier one the **naive version**.

We can also run the multiple times with the new final score being the average of all the runs. We call it the (**merge version**). We notice that even though the recompute version can produce a ranking which is nearly the same as the ranking of the nonrandomized version, it is limited to the top N outliers. On the other hand, the output of the merge version is less similar for the top outliers, but the similarity can be improved for all the points. Thus, we first produce the average outlier scores using the merge version, then we recompute the score of the top outliers (**hybrid version**). Finally, we have rankings similar to that of the nonrandomized method for all the outliers.

Algorithm 2 PARTITION(*Set D*)

```
SPLIT(D, S_1, S_2)
if |S_1| > M_θ then
    PARTITION(S_1)
else
    COMPUTECANDIDATES(S_1)
end if
if |S_2| > M_θ then
    PARTITION(S_2)
else
    COMPUTECANDIDATES(S_2)
end if
```

5.1 Query Time of New Point

The partition can actually be treated as the creation of a binary tree with two-key nodes. Each key represents a new subset. The two keys are the selected points (called split points) for the partition. Each key has a pointer to the child node. The structure is then recursively created. A leaf node is a node which represents a subset which will not be further partitioned. The leaf node contains all points of the subset. The keys of the root node are the first two randomly selected points. To traverse the tree, we start with the root node. We compare a query point p to the keys of the parent node and choose the key which is closest to p. Then, we traverse the tree to the child node referred by this key. We repeat this process until we reach a leaf node where we will compute the outlier score for p with respect to the leaf node.

As we discussed earlier, we will maintain multiple trees for ruling out false outliers. The number of trees corresponds to the number of iterations. The score of a point will be the minimum score computed from all the trees. The time complexity of a query is $O(h + f(M_\theta))$, where h is the height of the tree and $f(M_\theta)$ is the time required to compute the outlier scores for the subset. If the trees are balanced, the number of steps to reach the leaf nodes is $O(logn)$.

5.2 Time Complexity Analysis

We use the tree structure discussed in section 5.1 to analyze the time complexity of the algorithm. The algorithm consists of three main steps: partition, outlier score computation for local sets, and merge.

The *partition step* is fastest when the tree is perfectly balanced. Multiple partitions are required until the subsets are less than M_θ. For each level h, there are 2^h subsets, the size of each set is $\frac{n}{2^h}$, thus the partition cost at this level is $O(2^h \times \frac{n}{2^h}) = O(n)$. The total time for all levels is $O(H \times n)$, where H is the height of the tree. If the tree is balanced, $H \approx logn$. The total time will be $O(nlogn)$. In the *outlier score computation step*, we consider it a constant $O(c)$ because the sizes of the subsets are very small. The maximum number of subsets is n, the worst time complexity to compute the scores is $O(n)$. In the worst case, the *merging process* for different runs and iterations can be done in $O(n)$.

In total, the upper bound for the balanced tree is $0(nlogn)$. In practice, we may not have a balanced tree, however, if we assume that most of the time the ratio of the sizes of subsets after a split is a reasonable value, the time complexity can be roughly approximated as in the balanced tree. It is possible that a partition may result in two completely unbalanced subsets where one set contains most of the points. Therefore, the key is to ensure that the probability of completely unbalanced subsets is rare. If a tree is completely unbalanced, the data set is always divided into two groups such that one of them contains most of the data. However, theorem 1 shows that the probability of isolating a point from its neighbors is small. Therefore, the probability of always isolating a point from its neighbors is small. In other words, the probability for the algorithm to approach $O(n^2)$ is small. The speed is guaranteed under this assumption; however, in practice, it is showed that the algorithm can yield fast performance consistently.

6 Experiments

6.1 2D Example

We use a two dimensional dataset to show that the randomized method can detect local outliers correctly. We generate two Gaussians with different means and standard deviations. We then generate two local outliers p_1 and p_2 for clusters C_1 and C_2. The dataset is illustrated in figure 2. First, we compute the outlier scores using the nonrandomized version. The LOF method detects two outliers p_2 (2.75) and p_1 (2.4) as two top outliers. In addition, it returns two other outliers q_2 (2.2) and q_1 (2.1). These outliers are synthetically generated by the Gaussian. Then, we compute the scores using the merge version. We set $M_\theta = 100$ and $N_{run} = 6$. The points p_2 and p_1 are consistently detected as the top two outliers for all the different runs. Their final scores are 2.65 and 2.35 respectively, which are very close to the original scores. In contrast with p_1 and p_2, the rankings for q_2 and q_1 are not consistent. However, when using merge version, they were also ranked correctly. The scores are 2.1 and 1.9 respectively.

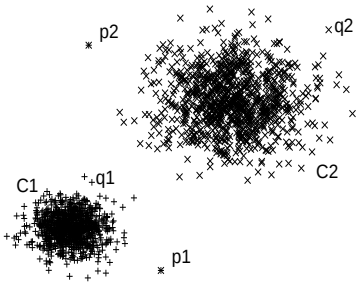


Fig. 2: 2D test data for local outlier

Table 1: Dataset description.

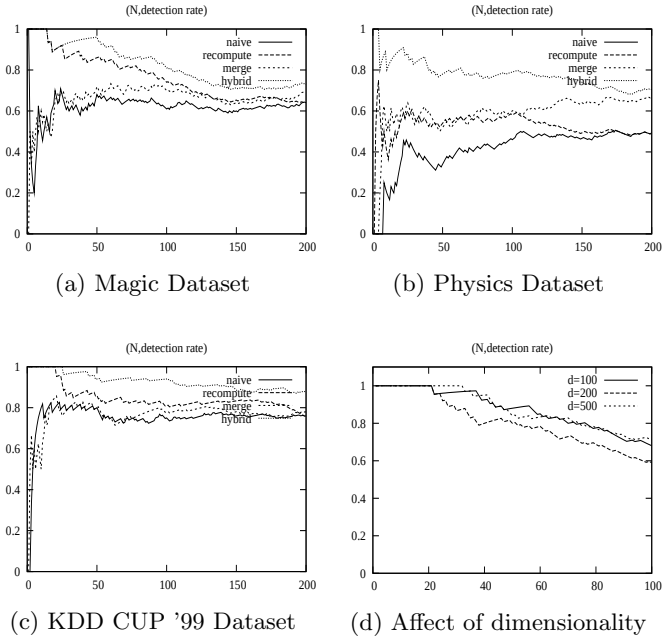
Dataset	D	N
Magic	10	19K
Physics	69	50K
KDD Cup '99	34	80K

The experiment shows that the randomized method is as good as the original method using full nearest neighbor computation. In some cases, some outliers may be ranked differently but on the average the output of the randomized method converges to the original method.

6.2 Real Datasets

Dataset Description We will evaluate the performance of our method against the original LOF method with three different datasets: MAGIC Gamma Telescope [20], Physics [7], and KDD Cup '99 Network Intrusion [20]. The detail of the data sets after normalization and removing nonnumerical attributes are shown in Table 1. We will refer the outliers computed by the nonrandomized LOF as nonrandomized outliers.

Evaluation Metrics Before proceeding with the experiments, we will discuss about the metrics for evaluating the effectiveness of the randomized method. The LOF gives two results which are the local outlier factor (which we called score in our method) and the ranking of the points according to the local outlier factor. We observe that the LOF is sensitive to the parameter $MinPts$. With the same LOF method, a small in change in $MinPts$ can lead to changes in the ranking and the scores. Except for very strong outliers where the scores are distinct, the ranking is sensitive to the scores. It is even more sensitive for points with low outlier scores. For an example, there is no much difference between the rankings of 200 and 203 for the outliers with the scores of 1.90 and 1.85 due to the statistical variation. Therefore, the objective is not to have the exact same scores and rankings between the original and randomized versions. Instead,



(a) Magic Dataset (b) Physics Dataset

(c) KDD CUP '99 Dataset (d) Affect of dimensionality

Fig. 3: Similarity in ranking for randomized outlier detection

the first objective is to have the similar scores with some acceptable statistical variation. Since we are interested in the top outliers, we will try to preserve the ranking for these outliers. This preservation is important if there are strong and distinct outliers in the dataset. Therefore, we will evaluate the method using the following metrics:

We will use the "detection rate" $\frac{N-N_{lof}}{N}$ to evaluate the top outliers, where N_{lof} are the number of outliers in the top N outliers in our method that also appear in the top N nonrandomized outliers. According to the experiments, the scores drop quickly when the points are outside the top 100 outliers (which makes the ranking sensitive to small changes of the scores). Thus, we will vary N up to 200 outliers. For the weak outliers, we compute mean and standard deviation of the ratios of the absolute differences between the methods for every point. If they are small, the two methods produce similar scores.

Effectiveness of the Randomized Method We will then evaluate the effectiveness of the randomized method as follows:

First, we run the nonrandomized LOF on the datasets to compute the outlier scores ($minpts = 20$). Then, we run the randomized method on the datasets ($M_\theta = 500$, $N_{iter} = 20$, and $N_{run} = 6$). The results are shown in Figure 3a, 3b, and 3c. In all the figures, the naive version performs worst in comparison with

the others. Nonetheless, in all the experiments, it still guarantees the detection rate of 40% for $N = 25$. It means that at least the top ten outliers are detected. The method performs best for the KDD dataset where the top 20 outliers are identified. The merge version produces slightly better results for the Magic and KDD datasets. At least 50% of the top 50 outliers are detected. The performance of the merge version is more stable compared with the naive versions when N increases. As expected, the recompute version boosts the performance for all the datasets. In the figures, all the top five outliers are correctly detected. At least 80% of the top 50 outliers are detected in the Magic and KDD datasets. However, the differences in the rankings start to increase when N increases. By using the hybrid approach, the performance of the randomized version becomes stable with high accuracy. As we can see, this approach is the best in all the experiments.

By manually examining the results, we found that the KDD dataset contained many strong outliers. The outlier scores for the KDD dataset are high while those in the Magic and Physics datasets are low. It can be explained by the fact that the KDD dataset contains many intrusion attack connections. This makes the distinction between the outlier scores in the KDD dataset more obvious. Therefore, the results in the KDD dataset are more stable than those in the other two datasets.

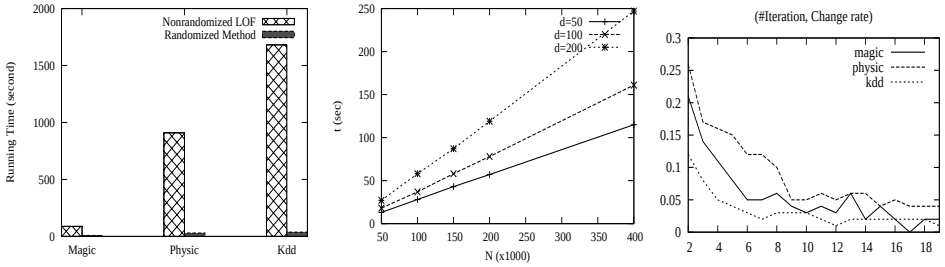
For the weak outliers, we compute the mean and standard deviation as mentioned earlier. We found that the top 180, 167, and 151 outliers had the exact same scores with the outliers computed by the original LOF in the Magic, Physics, and KDD datasets respectively. The statistics imply that our method and the original LOF method produce similar results .

6.3 Dimensionality

We want to answer the question whether the effectiveness of the randomized method will also be affected by the "curse of dimensionality" as index trees. We generate synthetic datasets with the dimensionality up to 500. We run the experiments with d = 100, 200, and 500. The datasets consist of the Gaussians with randomly generated means and standard deviations. We also inject ten randomly generated outliers into the datasets. According to figure 3d, the ten injected outliers are correctly identified and the top 20 outliers are correctly identified in all the experiments. We notice that there is a slight decrease in the detection rate when d increases. When we examine the outliers manually, we find that it is because the scores of the outliers become closer when d increases which makes the ranking fluctuate. This experiment shows that the randomized method is still viable in very high dimensions.

6.4 Speed Comparison

We evaluate the running time of the randomized method against the nonrandomized version of LOF using the Magic, Physics, and Kdd Cup '99 datasets. In these datasets, Magic is the smallest (19K points) while Kdd is the largest



(a) Speed Comparison (b) Running Time (c) Convergence rate

Fig. 4: Performance

Table 2: Running Time.

Dataset	**Nonrandomized LOF (second)**	**Randomized LOF (second)**
Magic	89	8
Physics	909	30
Kdd Cup '99	1681	37
Synthetic	4848	106

(80K points). In Figure 4a, the running time of the nonrandomized version grows quickly when the size of the datasets increase from 19K to 80K. In the mean while, the speed of the randomized method grows slower. In addition to Magic, Physics, and KDD Cup '99 datasets, we use a synthetic dataset with 200 dimensions and 100K points. The synthetic dataset contains five randomly generated Gaussians and ten random outliers. According to the experiment, the randomized method is consistently faster than the original LOF method. The running time of the methods in seconds are shown in Table 2.

6.5 Performance

We randomly generate the Gaussian clusters with different means and standard deviations for the sizes from 50K to 400K. We randomly injects top 10 outliers in the datasets. We generate the datasets for d = 50, 100, and 200. According to the results, all the generated outliers are detected as the top outliers. Figure 4b shows the running time for different datasets. The vertical axis shows the running time in seconds. In the figure, the running time is linear with the size of the dataset for different dimensions. The experiments show that the algorithm can scale well with high dimensionality.

6.6 Convergence Rate

The method relies on multiple iterations in order to rule out false outliers. We will evaluate how the iterations affect the effectiveness of the method. We observe that in the first few iterations there will be many false outliers. However, when the number of iterations (N_{iter}) increases, these outliers will be ruled out in the next iterations. The quality of detected outliers will be stable at some iterations. We will evaluate it based on the changes in the scores. This experiment aims to identify a good value of N_{iter} in practice.

Figure 4c shows the rate of change in the size of outliers for the Magic, Physics, and KDD dataset (after filtering out the low score outliers). As expected, the figure shows that the number of outliers changes rapidly in the first few iterations and the rate of change becomes stable when N_{iter} approaches 10. The rate of change is insignificant when $N_{iter} > 10$. We also evaluated with the datasets in multiple runs and we found that in general, $N_{iter} = 10$ is a reasonable choice for the randomized algorithm.

Parameter M_θ The parameter M_θ is the stop condition for the partition step. The partition will stop if there is less than M_θ points in the partition. As discussed earlier, the M_θ should not affect the quality of the algorithm as long as it is large enough. In our experiments, the scores are not affected when we try to increase M_θ.

7 Conclusion

We have shown that it is unnecessary to perform the KNN computation for the entire dataset in order to identify local density-based outliers. We introduced a randomized method to compute the local outlier scores very fast with high probability without finding KNN for all data points by exploiting the outlier consistency property of local outliers. We also introduced a hybrid version for the randomized method by recomputing the scores and combining the scores using multiple runs of the algorithm to improve its accuracy and stability. The parameters can be selected intuitively. We have evaluated the performance of our method on a variety of real and synthetic datasets. The experiments have shown that the scores computed by the randomized method and the original LOF are similar. The experiments also confirm that the randomized method is fast and scalable for very high dimensional data. To the best of our knowledge, our method is the first that can compute LOF very efficiently with high accuracy in high dimensional data. In our experiment, we have evaluated the method up to 500 dimensions.

References

1. C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 37–46, New York, NY, USA, 2001. ACM.

2. S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 29–38, New York, NY, USA, 2003. ACM.
3. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, 1990.
4. S. Berchtold, D. A. Keim, and H.-P. Kriegel. The x-tree: An index structure for high-dimensional data. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 28–39, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.
5. M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, 2000.
6. V. Chandola, A. Banerjee, and V. Kumar. Outlier detection: A survey. *ACM Computing Surveys*, pages 1–72, Sept. 2009.
7. Charles Young et al. KDD Cup 2004: Quantum physics dataset, 2004.
8. N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer. *SMOTEBoost: Improving prediction of the minority class in boosting*, volume 2838/2003 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, Germany, 2004.
9. K. Das and J. Schneider. Detecting anomalous records in categorical datasets. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 220–229, New York, NY, USA, 2007. ACM.
10. Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 392–403, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
11. J. H. Freidman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.
12. D. Hawkins. *Identification of outliers*. Chapman and Hall, London, 1980.
13. R. A. Jarvis and E. A. Patrick. Clustering using a similarity measure based on shared near neighbors. *IEEE Transactions on Computers*, C-22(11):1025– 1034, 1973.
14. F. Korn, B.-U. Pagel, and C. Faloutsos. On the 'dimensionality curse' and the 'self-similarity blessing'. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):96–111, 2001.
15. H.-P. Kriegel, M. S. hubert, and A. Zimek. Angle-based outlier detection in high-dimensional data. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 444–452, New York, NY, USA, 2008. ACM.
16. A. Lazarevic and V. Kumar. Feature bagging for outlier detection. In *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166, New York, NY, USA, 2005. ACM.
17. H. Mannila, D. Pavlov, and P. Smyth. Prediction with local patterns using cross-entropy. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 357–361, New York, NY, USA, 1999. ACM.
18. Micheline Kamber and Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2 edition, Mar. 2006.

19. Minh Quoc Nguyen, Edward Omiecinski, and Leo Mark. A Fast Feature-based Method to Detect Unusual Patterns in Multidimensional Data. In *11th International Conference on Data Warehousing and Knowledge Discovery*, Aug. 2009.
20. C. B. D. Newman and C. Merz. UCI repository of machine learning databases, 1998.
21. U. Shaft and R. Ramakrishnan. Theory of nearest neighbors indexability. *ACM Trans. Database Syst.*, 31(3):814–838, 2006.
22. Spiros Papadimitriou, Hiroyuki Kitagawa, Philip B. Gibbons, and Christos Faloutsos. LOCI: Fast outlier detection using the local correlation integral. In *Proceedings of the 19th International Conference on Data Engineering: 2003*, pages 315– 326. IEEE Computer Society Press, Mar. 2003.
23. I. Steinwart, D. Hush, and C. Scovel. A classification framework for anomaly detection. *J. Mach. Learn. Res.*, 6:211–232, 2005.