Discovering NULL and Outliers

Guanhua Chen
New York University
New York City, New York
gc2229@nyu.edu

Wei Wang
New York University
New York City, New York
ww1306@nyu.edu

Yicong Xu
New York University
New York City, New York
yx1271@nyu.edu

ABSTRACT

The paper introduces several algorithms which can discover null and outliers in general big-scale dataset.

ACM Reference Format:
Guanhua Chen, Wei Wang, and Yicong Xu. 2018. Discovering NULL and Outliers. In *Proceedings of DS1004 (Final Project)*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Data cleaning has become a major problem in the field of Big Data. The data that is incorrect(value or format), incomplete, far beyond or below the average or null, is often trade as 'dirty data'.

Abnormal data like null and outliers make data analysis more difficult. In some case if we want to calculate the average, max or min value among a data set, null and outliers could make the result inaccurate or wrong.

In some data-sensitive fields, like bank or network company, 'dirty data' like null or outliers could be a signal of dangers. If someone's credit card is stolen, the database will appear anomalies. By detecting the fields like 'where the credit card is consumed', we could guess the probability of a credit card being stolen. If the location is far away from where the credit card is usually consumed or the address of the card's owner, we could made a decision that there might be a high probability of credit card being stolen. In industrial company, we could detect which machine is broken by detect the values it produced. According to both cases above we could conclude, detecting abnormal data can help us reduce losses as much as possible.

So, it is important and meaningful to solve the problems of detecting null and outliers. In this report, we will come up with some methods to detect null and outliers. We implement algorithm in spark to detect null. For outlier, we use both cluster-based and distance-based to calculate the outliers. The detailed method are listed in method part.

2 BACKGROUND

An outlier in a set of data is an observation or a point that is considerably dissimilar or inconsistent with the remainder of the data. There are some works related to detecting outliers before.

The work of [BL94] uses statistical technique. A distribution model is applied and the outliers are determined on how they appear in relation to the postulated model, which is not simple because user must have knowledge about the data distribution.

Knorr and Ng [KN98] come up with a new method to discover outliers by determining the distances between points in data set. And there are several algorithms to detect the distance-based outliers such as block nested-loop algorithm and cell-based algorithm. For the millstone progress, we partly reference the algorithm of this research.

3 PROBLEM FORMULATION

There are several phases during the process of the project. First of all, we should choose algorithms to detect outliers and null and find relatively good algorithms. In this report, we will use filter to detect null value. And for outliers, we apply Distance-based and Cluster-based method to detect outliers.

We implement all algorithms in Spark, which can take advantage of the high-performance of spark-rdd and the DAG execution engine.

Some issues that need to be considered/to be solved: How to deal with the null value after detecting them? By doing so, we could decide whether it should be deleted or replaced and how to do generalization among the datasets with the fields of unknown specific semantics.

High time complexity. Because for nested-loop, we need to calculate the distances between points. And we need to decide whether Hadoop and spark could simply this issue and how to apply them in algorithms like nested-loop and K-means.

How to do generalization among the dataset? In some fields, we do not know the semantics So it is difficult to find the relevances and weights between them.

4 TRANSFORMATION INNOVATION TO CALCULATE THE DISTANCE

The practical data has two major challenges before any applying of outlier detection method. The first is that the raw data may contain characters that are not in numeric format so we can not calculate the distance. Secondly, the columns may have different practical meaning. The distance generated in column A can not be compared with column B. For the first problem, we use the one hot encoding method. For the second method, we use the normalization method.

4.1 One hot encoding

In our project, we majorly use the distance between the points to detect the outliers: if a point is too far away from other points (or larger than some given values), then it may be an outlier. The key part of our innovation is the calculation of the distance.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

If the data only has integer or float features, then the distance may be easily calculated by euclidean metric:

$$distance = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_i - y_i)^2}$$

In some dataset, the data may have the categorical features (in string or character form) that need to be simplified by converting into integers. Here we offer a method to quantify the string format. For example, if we have a data frame which has the form:

Fruit
Apple
Orange
Orange
Pear

A trivial way to encode the non-numerical features is to transform those hashable features in the same column. For example, the table 1 would be transformed as:

Origin column	After encoding
Apple	1
Orange	2
Orange	2
Pear	3

The drawback of this method is that the distance between different labels is different. For instance, the distance from 'apple' to 'orange' is 1, and 'apple' to 'pear' is 2. That is not reasonable that the distance from apple to pear is large than apple to orange. To improve this, we create new features.

Algorithm 1 Encoding a non-numerate column

```
1: Mark the values of the column as $V : (v_1, v_2, \cdots, v_n)$
2: Select the unrepeatable elements in $V$ as $L$
3: $(l_1, l_2, \cdots, l_k) = set\{v_1, v_2, \cdots, v_i\}$
4: for each $l_j \in L$ do
5:     create a new empty column $C_j : (v_1^j, v_2^j, \cdots, v_n^j)$
6: end for
7: for each $v_i \in V$ do
8:     for each $j \in [1, k]$ do
9:         if $v_i = l_j$ then
10:            $v_i^j = 1$
11:        else
12:            $v_i^j = 0$
13:        end if
14:    end for
15: end for
```

The table 1 will be changed into this:

Origin column	Apple	Orange	Pear
Apple	1	0	0
Orange	0	1	0
Orange	0	1	0
Pear	0	0	1

Distance between different 'apple' and 'orange' is the same as the distance between 'apple' and 'orange'. A drawback of this method is that this method will create many new features (columns) and make the distance calculation become slower. For some instance, there are some values appears seldomly in the column, if we generate new columns for those values, than we may introduces large bias because these values may be mistakenly recorded or be outliers. To avoid this, we set a threshold. If the frequency of a value is lower than the threshold, we believe that it could be dropped out.

4.2 Normalization

There are two kinds of methods of normalization. The first normalization method treats data as a vector in a multidimensional space. The transformed data is a new vector whose norm is 1.

If the distribution of the data is Gaussian distribution, then we may normalize the data with the standard deviation. Formally, a column which obey the Gaussian Distribution contains n values $x_1, \cdots, x_n$ with the mean M and the standard deviation S, then the transformed value Z is given by $Z_i = \frac{x_i - M}{S}$ This is also called the Z-score method.

5 METHODS AND DESIGN FOR OUTLIER DETECTION

In our project, we will try two different kinds outlier detection methods. The first method is based on K-Means method. If a data point is far away from its K-centers, then the point may be an outlier. The second method is based on the distance of a point from its kth nearest neighbor.

5.1 K-means methods

The kmeans is a machine learning clustering method. Given a group of n data points, k-means partition those points into k clusters in which each point belongs to the cluster withe the nearest mean.

The method use an iterative refinement technique. Given an initial set of cluster centers $(m_1, m_2, \cdots, m_k)$, we calculate the distance from the point to each center. Each point is assigned to its nearest cluster center. Then, we update the cluster center by calculate the centroids of the points belong to that cluster. The algorithm is proved stable after several assign-update process.

When a k-means graph is built, an outlier is the point which has a larger distance to its center.

The k-means clustering algorithm is as follows:

1. initialize cluster centroids $\mu_1 \ldots \mu_k randomly$

2. Repeats:

{

For every i:

$$c^i = argmin_j ||x^i - u_j||$$

for each j:

$$\mu_j = \text{averge of points of culster j}$$

}

Algorithm 2 K-means to find outlier

```
1: L is the threshold distance
2: Apply K-means method to find centroids of each point
3: for i = 0 to N do
4:     x ← kmeans center of $v_i$
5:     if distance from $v_i$ to x >=L then
6:         Mark $v_i$ as outlier
7:     end if
8: end for
```

K-means clustering is a NP-hard problem which has a long running time. This method is especially suitable for those data who has the cluster structure.However, we need to know in advance what is the L value and how many clusters we need.

5.2 DBSCAN for outliers detection

Principle of DBSCAN: Density-based spatial clustering of applications with noise (DBSCAN) is a popular density-based clustering algorithm proposed by Martin, Ester, Hans-Peter Kriegel, Jorg Sander and Xiaowei Xu in 1996. Such density clustering algorithms generally assume that the category can be determined by the tightness of the sample distribution. In the same category of samples, they are closely linked, all the points within a category are not far from each other. After clustering, points that are closely packed together. Outliers are displayed in the regions with sparse density.

DBSCAN is based on a set of neighbors of a core points to describe tightness. DBSCAN needs a parameter pair(ε, MinPts) to describe the closeness of the sample distribution in the neighbors. ε represents a distance threshold under which two points are considered to be closed. MinPts means the minimum number of points required to form a dense region.

Here are the details of how to use DBSCAN to detect outliers [DBSCAN].

- A point p is a core point if at least MinPts points are within distance ε (represents the maximum radius of the neighbors from p).
- A point q is directly reachable from p if point q is within distance ε from point p and p must be a core point.
- A point q is reachable from p if there is a path p1, ..., pn with p1 = p and pn = q, where each pi+1 is directly reachable from pi (all the points on the path must be core points, with the possible exception of q).
- All points not reachable from any other point are outliers.

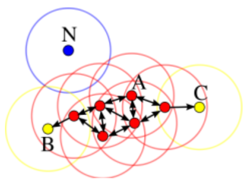

Figure 1: Process of clustering

The figure above shows the process of detect outliers. In this figure, we set MinPts = 4. All the red points are core points because they have more than 4 neighbors in the circle region with radius ε. Points B and C are not core points, but are reachable from A (via other core points) and thus belong to the cluster as well. But for the blue point N, there is no way to reach N, so N is an outlier point.

Algorithm 3 Find_Neighbors

```
Require: DB, distFunc, Q, eps
Ensure: Neighbors
1: Neighbors = new list
2: for each point P in database DB do
3:     if distFunc(Q, P) <= eps then
4:         Neighbors = Neighbors ∪ P
5:     end if
6: end for
```

Algorithm 4 DBSCAN

```
Require: DB, distFunc, eps, minPts
1: C = 0, Cluster conuter
2: for each point P in database DB do
3:     if label(P) != undefined then
4:         Continue, Previously processed in inner loop
5:     end if
6:     Neighbors N = Find_Neighbors(DB, distFunc, P, eps)
7:     if |N| < minPts then
8:         Label(P) = Noise, Label as Noise
9:         continue
10:    end if
11:    C = C + 1, next cluster label
12:    label(P) = C, label initial point
13:    Seed set S = N  P, Neighbors to expand
14:    for each point Q in S do
15:        if label(Q) = Noise then
16:            label(Q) = C, Change Noise to border point
17:        end if
18:        if label(Q) != undefined then
19:            Continue, Previously processed
20:        end if
21:        label(Q) = C, Label neighbor
22:        Neighbor N = Find_Neighbors(DB, distFunc, Q, eps),
   Find neighbor
23:        if |N|>=minPts then
24:            S = S ∪ N, Add new neighbors to seed
25:        end if
26:    end for
27: end for
```

Advantages of DBSCAN:

- It could cluster datasets of any shape.
- Abnormal points can be found at the same time as clustering and it is not sensitive to the outliers in the data set.

Disadvantages of DBSCAN:

- If the density of the data set is not uniform and the points are sparse, the clustering quality is poor. In this case, DBSCAN is not a suitable clustering algorithm.

- If the data set is large, it will take a lot of time for clustering convergence.
- Tuning the parameters are slightly more complicated than the tranditional K-Means clustering algorithms, such as the distance threshold ε and the number of neighbors threshold MinPts. The different parameter combinations have a great influence on the final clustering effect.

5.3 Kth-neighbor method

5.3.1 Nested-loop Approach. From [CHT00], we use the concept of $D^k(p)$ to calculate the outlier.

Definition: Given an input data set with N points parameters n and k, a point p is a D^k_n outlier if there are no more than n-1 other points p' such that $D^k(p') > D^k(p)$.

If we rank points according to their $D^k(p)$ distance, the top n points in this ranking are considered to be outliers.

To reduce the computational cost, we use the partition method provided by [CHT00]. We use a clustering method to partition n data into several partitions. For each partitions, we compute the bounds of D^k. If the upper bound of D^k is smaller than a certain value minDkDist, than points in that partition cannot be an outlier and we do not need to calculate that part and the computational cost is reduced.

MinDkDist is the lower bound on D^k for an outlier and is computed as follows: If a partition has more than n points (n is the number of outliers), The lower bound on D^k for an outlier can not be smaller then the lower bound on D^k in the partition. If a group of partitions has more than n points, than minDkDist is at least the minimum value of the lower bounds of those partitions.

We will implement the algorithm using python and Spark on multiple datasets and compare the results with the nested-loop and index-based algorithm.

Algorithm 5 Nested Loop K-neighbor to find outlier

```
1: for each point $p_i$ in the dataset do
2:     calculate its distance between all other points
3:     Dist[i]← distance of a point from its kth nearest neighbor.
4: end for
5: Sort Dist[i] from large to small
6: Mark $Dist_1 \ldots Dist_n$as outliers
```

5.3.2 KD-Tree based approach. But the simple nested-loop algorithm with worst-case complexity $O(\delta N^2)$ where δ is the number of dimensions and N is the number of points in the dataset. This is expensive computationally, especially if the dimensionality of points is high. It takes a lot of time when calculating large data set.

The number of distance computations can be substantially reduced by using a spatial index like an kd-tree.

A kd-tree is a tree structure where each node corresponds to a rectangle: in d-dimensional space, a rectangle is the product of d closed intervals on the coordinate axes. Each internal node has an axis-aligned hyperplane that splits the rectangle; The two subrectangles thus formed are associated with the two child nodes. Each point in a point cloud is stored in a leaf whose rectangle contains it. The set of points in a leaf is also known as a bucket, and the bucket size is normally bounded by some small constants.

After using KD-tree, the complexity for constructing the the KD-tree is $O(NlogN)$ and the average cost of 1-NN query is $O(logN)$. This cost is much less than the nest-loop approach.

Algorithm 6 Index-based K-neighbor to find outlier

```
1: for each point $p_i$ in the dataset do
2:     Insert the node to the kd-tree
3: end for
4: for each point $p_i$ in the dataset do
5:     search the node in the kd-tree
6:     store the Kth-nearest neighbor to the Dist[i]
7: end for
8: Sort Dist[i] from large to small
9: Mark $Dist_1 \ldots Dist_n$as outliers
```

5.4 Innovation on Kth-neighbor method– Local Outlier Factor

The definition of outliers from Kth-neighbor method has the advantages of being both intuitive and simple, as well as being computationally feasible for large sets of data points. However, it also has certain shortcomings:

- It requires the user to specify a K which is sometimes hard to decide. Hence for different databases, it needs people to adjust the optimal k-value manually. For small-scale dataset needs small k bit large-scale dataset needs big k.
- It cannot identify outliers in a data set that would not be outliers in another area of the data set. For example, a point at a "small" distance to a very dense cluster is an outlier, while a point within a sparse cluster might exhibit similar distances to its neighbors.

There is a new algorithm based on Kth-neighbor method: **Local Outlier Factor**. It is an algorithm proposed by Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng and JÃűrg Sander in 2000 for finding anomalous data points by measuring the local deviation of a given data point with respect to its neighbours.[BKNS00]

The local outlier factor is based on a concept of a local density, where locality is given by k nearest neighbors, whose distance is used to estimate the density. By comparing the local density of an object to the local densities of its neighbors, one can identify regions of similar density, and points that have a substantially lower density than their neighbors. These are considered to be outliers.

Formals of LOF are listed below:

5.4.1 k-distance. :Let k-distance be the distance of the object A to the k-th nearest neighbor, which is same as the definition in the Kth-neighbor method.

5.4.2 k-distance neighborhood of p. :We denote the set of k nearest neighbors as $N_k(p)$.

5.4.3 reach-distance. : $reach{-}distance_k(p, o) = max\{k{-}distance(o), d(p, o)\}$ In words, the reach distance of an object p from o is the true distance of the two objects, but at least the k-distance of o.

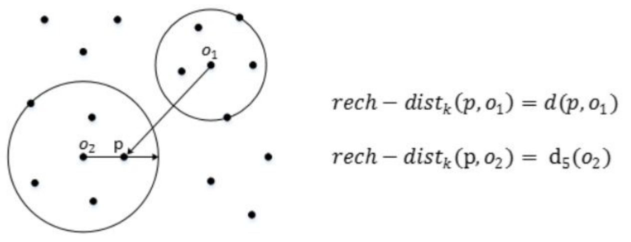
o_1
o_2
p
$rech-dist_k(p, o_1) = d(p, o_1)$
$rech-dist_k(p, o_2) = d_5(o_2)$

Figure 2: reach-distance

5.4.4 *local reachability density .* :

$$lrd_k(p) = 1/\left(\frac{\sum_{o\in N_k(p)} reach-distance_k(p,o)}{N_k(p)}\right)$$

which is the inverse of the average reachability distance of the object p from its neighbors.

5.4.5 *local outlier factor.* :

$$LOF_k(p) = \frac{\sum_{o\in N_k(p)} \frac{lrd_k(o)}{lrd_k(p)}}{N_k(p)}$$

which is the average local reachability density of the neighbors divided by the object's own local reachability density. A value of approximately 1 indicates that the object is comparable to its neighbors (and thus not an outlier). A value below 1 indicates a denser region (which would be an inlier), while values significantly larger than 1 indicate outliers.

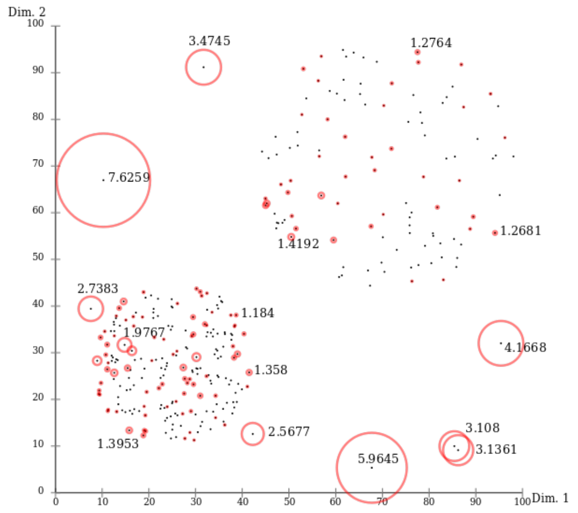

Figure 3: visualized LOF scores example

Algorithm 7 local outlier factor Algorithm to find outlier

```
1: for each point $p_i$ in the dataset do
2:     Insert the node to the kd-tree
3: end for
4: for each point $p_i$ in the dataset do
5:     search the node in the kd-tree
6:     store the nearest k- neighbor distance to the Dist[i][0] - Dist[i][k-1]
7:     store the nearest k- neighbor index to the N[i][0] - N[i][k-1]
8: end for
9: for each point $p_i$ in the dataset do
10:    Calculate the reach-distance of points in $N_k(p_i)$
11:    Calculate the local reachability density of $p_i$
12: end for
13: for each point $p_i$ in the dataset do
14:    Calculate the local outlier factor of $p_i$
15:    if LOF($p_i$) > 1.5 then
16:        Mark $p_i$ as outlier.
17:    end if
18: end for
```

5.5 Null value detection method

There are several ways to detect the null value. In python we could simply traversed the dataset. In Spark, we could simply generate spark RDD to help us analyze the dataset. By using spark, we could take advantage of fault tolerant mechanism, cache, parallel operation and so on. Spark also provides some programming interfaces for many operations like map, reduce, filter, min, max, which significantly reduced operations during the process. Also we could choose spark sql to make the process more simple and make full use of distributed data.

In this report, we try to use spark filter to deal with this issue. Firstly, we load the dataset and create sparkRDD. For each row:

Algorithm 8 Null value detection

```
1: for each row r in the Dataset do
2:     filter the row a with null value filter
3:     if the length of the row < the original length of the row then
4:         Store rows with null value in a new RDD named new_dataset
5:     end if
6: end for
7: for each row r in the new_dataset do
8:     if the dimension of the r is 0 then
9:         remove r in DataSet
10:    else
11:        replace null values in r with specific values
12:    end if
13: end for
```

6 RESULTS

6.1 Null Detection

For Null value detection, we implement the algorithm with spark. By now, we just locate the null value and remove the row from the data set. But we found that for some dataset full of null, just removing null will not be a good choice, a better strategy will be our target of next stage.

Here we apply algorithm to f42p-xqaa.tsv, it returns well the exact index and no. of column of the null value.

Table 1: Null of f42p-xqaa.tsv

index	column
72	1
72	2
72	3
...	...
284	1
284	2
...	...

6.2 K-neighbor Algorithm

For Outlier detection, The K-means method is implemented by spark and referenced from lab5, which has been shown to have a good result. So here we focus on our Kth-neighbor method result.

We use a relatively good dataset 5c5x-3qz9.tsv provided and use the algorithm to pick out the top 10 outlier. Here is part of result table.

Table 2: Outliers of 5c5x-3qz9.tsv

index	num_l_1	num_l_3	number_t	num_level_2	pct_l_3
189	3279	64874	115666	16260	83.1
187	6665	62282	118098	22034	75.7
185	11456	56553	121464	28829	66.8
183	17272	53747	124108	33493	59.1
385	1647	53613	101346	9167	89.3
193	9408	41976	113509	36667	59.4
195	8513	41469	112282	33655	62.4
383	3232	52849	100598	12770	84.1
191	10566	37463	114242	39622	56.1
391	4942	41097	106247	24973	71.8
mean	1139	5275	12245	3239	55

From the table, we can see that in many columns the outliers have really large difference from the mean value of the column, which makes sense.

6.3 LOF Algorithm

Then we run the LOF algorithm on pwva-zn2w.tsv, here is the result.

Table 3: Outliers of pwva-zn2w.tsv

INDEX	feb_2011	feb_2012	feb_2010	jan_2010	jan_2011	LOF
70	0	0	0	0	0	15.99
82	102294	73933	72739	95990	76923	14.87
89	39000	32200	41000	40600	43200	11.11
5	62800	54400	65600	61200	60000	10.68
93	24000	26400	30000	28800	26400	10.59
66	90800	90000	97200	90400	91200	10.27
17	93200	86000	98400	86800	81200	10.19
38	105600	116000	119200	132000	125600	9.91
mean	61594.94	58232.73	64944.72	63375.94	64641.74	

From the result, we can see that the rows with large LOF have great difference from the mean. For example, the most columns of row 70 are zero while other rows have values. So it is a significant outlier of the database.

6.4 DBSCAN Algorithm

Also apply DBSCAN using sklearn.cluster package. Result of DB-SCAN:

Table 4: Outliers of 5c5x-3qz9.tsv

_1	_2	_3	_4	_5	_6	_7	...	_27
85	2006	14345	32630	79993	26340	49.1	...	0
87	2007	9943	36384	78894	23570	57.5	...	0
89	2008	6335	42484	77896	19020	67.5	...	0
91	2009	3177	46031	77030	14323	77.3	...	0
183	2006	17272	53747	124108	33493	59.1	...	0
185	2007	11456	56553	121464	28829	66.8	...	0
187	2008	6665	62282	118098	22034	75.7	...	0
189	2009	3279	64874	115666	16260	83.1	...	0
191	2010	10566	37463	114242	39622	56.1	...	0
379	2011	8168	47834	101559	21841	70.5	...	0
...	...	...	...	...	...	...	...	...
391	2012	4942	41097	106247	24973	71.8	...	0
mean	2009	1139.31	5275.13	12245.73	3239.86	55.52	...	0

From the table, we can see that in many columns the outliers have really large difference from the mean value of the column, which makes sense.

6.5 Performance Result

We test four methods (nest-loop, k-means, DBSCAN, LOF) on the same dataset.

For the distance based method, the kd-tree algorithm would improve the time efficiency. We trained the kd-tree method and non kd-tree method on the same dataset. we cut the original test data set into smaller parts to show the performance result.

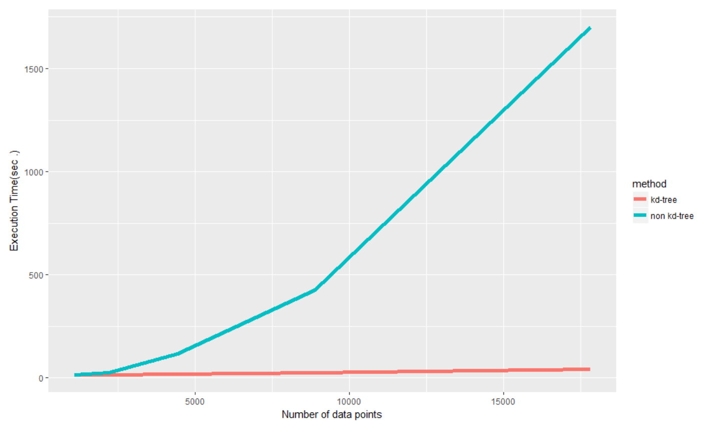

Figure 4: Performance result for kd-tree and non kd-tree

We also test the performance result of the DBSCAN and compared it with the LOF method (both method use the kd-tree algorithm).

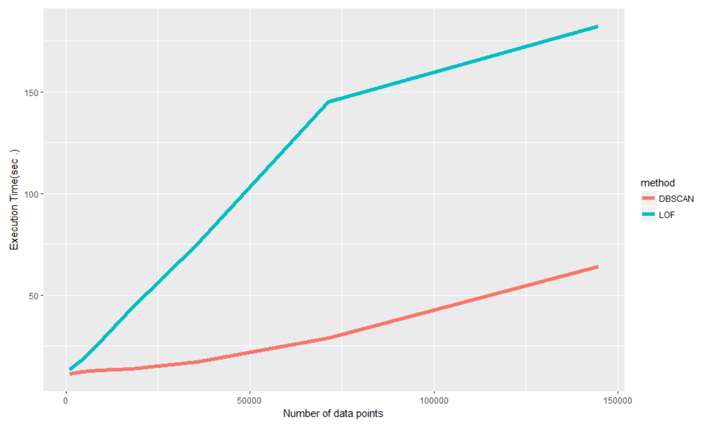

Figure 5: Fig Performance result of LOf and DBSCAN

7 CONCLUSION AND CODE REPOSITORY

In our project, we compare the four outlier detection methods. These four methods could be categorized as two groups: based on cluster method and based on distance.

K-means method partitions the data into K clusters. The outlier is the one whose distances to the cluster centers are largest. The DBSCAN method is the improvement of the K-means method. The traditional cluster bound is a hypersphere in the higher-dimensional space. In the DBSCAN method, data points which belongs to one cluster has a path to each other. The decision bound of the DBSCAN is the outer enveloping profile of several hyperspheres. The distance based method calculated the distance of all points pair which is time consuming. The kd-tree method is an improvement of the time efficiency. The time complexity is reduced from $O(N^2)$ to $O(NlogN)$.

The nest-loop method considers the distance from all data pairs. The Local Outlier Factor (LOF) method is focused on the local density. If the neighbors of one data point have more data around them, then the data has a higher chance to be identified as an outlier.

Code repository: https://github.com/PrinceNathaniel/NYUBigDataProject. We simply select two datasets separately for testing our algorithms.

Correctness, and Readability: All of the files have been tasked. And the preliminary results are shown in the .ipynb files, the temporary results are shown clearly at each step.

8 REFERENCES

[BL94] V. Barnett and T. Lewis. *Outliers in Statistical Data.* John Wiley and Sons, New York, 1994.
[KN98] Edwin Knorr and Raymond Ng. Algorithms for mining distance-based outliers in large datasets. In Proc. of the VLDB Conference, pages 392-403, New York, USA, September 1998.
[CHT00] Palo Alto, Murray Hill and Taejon. *Efficient Algorithms for Mining Outliers from Large Data Sets.* ACM Sigmod Record. ACM, 2000, 29(2): 427-438.
[DBSCAN] Density-based spatial clustering of applications with noise (DBSCAN)https://en.wikipedia.org/wiki/DBSCAN
[BKNS00]Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; Sander, J. (2000). LOF: Identifying Density-based Local Outliers (PDF). Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. SIGMOD. pp. 93-104. doi:10.1145/335191.335388. ISBN 1-58113-217-4.