DS 503
Distributed Local Outlier Detection on Hadoop

Lei Li, Nai-Tan Chang

I. Overrview

Outlier detections is one fundamental step in data quality, data management and data analysis tasks. Many applications are related to it, such as Fraud detection and network intrusion, and data cleaning. Frequently, outliers are removed to improve accuracy of the estimators.

One popular technique in outlier detection, the Local Outlier Factor(LOF), addresses challenges caused when data is skewed and outliers may have very different characteristics across data regions. As datasets increase radically in size, highly scalable LOF algorithms leveraging modern distributed infrastructures are required. [1] In this paper, inspired by the work done by Yizhou Yan et al, we focus on designing LOF algorithm in distributed computing paradigm to satisfy the stringent response time requirements of modern applications. All the works are based on Map-Reduce, written in Java.

The paper is organized as follows: In section 2, we discussed the related works and their drawbacks. In section 3, we described the problem statement and challenges. In section 4, we introduced Distributed LOF (DLOF) framework and specified the map-reduce jobs. In section 5, we introduced Data-Driven LOF (DDLOF) framework, which optimizes the DLOF for introducing supporting area.

II. Previous Work

Given a set of objects in feature space, a distance measure can be used to quantify the similarity between objects. Intuitively, objects that are far from others can be regarded as outliers. Proximity-based approaches assume that the proximity of an outlier object to its nearest neighbors significantly deviates from the proximity of the object to most of the other objects in the data set. The proximity-based algorithm can be divided into distance-based (an object is an outlier if its neighborhood doesn’t have enough points) and density-based methods (An object is an outlier if its density relatively much lower than that of its neighbours)

2.1 Distance-based

Distance-Based Outlier Detection approach, it is defined as: for each object o, examine the other objects in the r-neighborhood of o, where r is a user-specified distance threshold. An object o is an outlier if most of the objects are not in the r-neighborhood of o. However, this approach struggles in cases that have clusters of varying density, as a single threshold can fail to capture the complexity of the data.

2.2 Density-based

Density-based approach determines the density of points around the object and terms any value that exceeds a certain threshold to be an outlier. However, this simplistic approach again fails to properly parse and interpret clusters of varying density, with corresponding shortcomings in the classification of outliers.

2.3 Local Outlier Factor

Local outlier factor is a density-based method that relies on nearest neighbors search. The LOF method scores each data point by computing the ratio of the average densities of the point's neighbors to the density of the point itself. To compute LOF, it needs three steps, compute k-distance, Local reachability density and finally local outlier factor.
The k-distance neighborhood of **o** contains all objects of the distance to **o** is not greater than *dist_k(o)* the k-th distance of **o**

$$N_k(o) = [o' \mid o' \in D,\ dist(o, o') \leq dist_k(o)]$$

We can use the average distance from the objects in Nk(o) to o as the measure of the local density of o. If o has very close neighbors o’ such that *dist(o,o’)* is very small, the statistical fluctuations of the distance measure can be undesirably high. To overcome this problem, we can switch to the following reachability distance measure by adding a smoothing effect.

$$reachdist_k(o, o') = max[dist_k(o),\ dist(o, o')]$$

k is a user-specified parameter that controls the smoothing effect. Essentially, *k* specifies the minimum neighborhood to be examined to determine the local density of an object.Reachability distance is not symmetric.

Local reachability density of an object o is

$$lrd_k(o) = \frac{\|N_k(o)\|}{\sum_{o' \in N_k(o)} reachdist_k(o, o')}$$

We calculate the local reachability density for an object and compare it with that of its neighbors to quantify the degree to which the object is considered an outlier.

$$LOF_k(\boldsymbol{o}) = \frac{\sum_{\boldsymbol{o'} \in N_k(\boldsymbol{o})} \frac{lrd_k(\boldsymbol{o'})}{lrd_k(\boldsymbol{o})}}{\|N_k(\boldsymbol{o})\|}$$

The local outlier factor is the average of the ratio of the local reachability density of o and those of o's k-nearest neighbours. The lower local reachability density of o and the higher the local reachability densities of the *k*-nearest neighbors of o, the higher the LOF value is. This exactly captures a local outlier of which the local density is relatively low compared to the local densities of its k-nearest neighbors.

III. Problem Statement and Challenges

As described in section 2.3, the LOF score of p is determined by its kNN o, it's kNN's kNN $o^{'}$, and it's kNN's kNN's kNN $o^{''}$ in total k + k2 + k3 points. When scaling big data, centralized LOF can no longer hold the complexity of the computation. Therefore, implementing distributed solutions for LOF is a necessity.

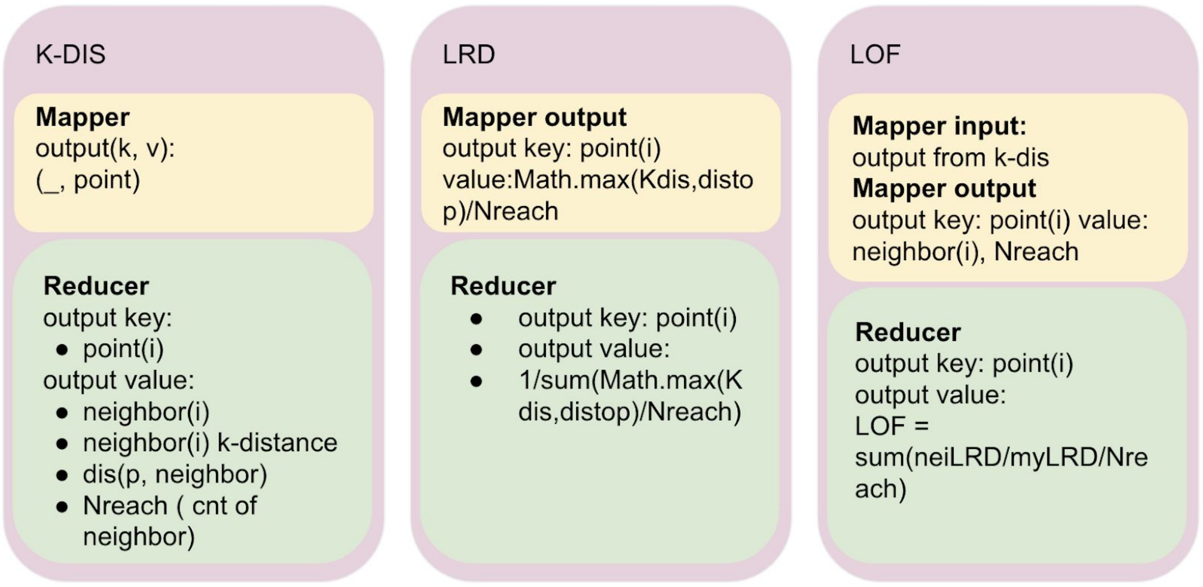
The research challenges in our approach arise from: When using open-source distributed infrastructures like Map-Reduce, within each machine in the compute cluster, data is distributed among machines according to some partitioning criteria, and only part of the dataset can then be accessed locally. However, LOF relies much on KNN search. So, how to locate the input data and required values on the same machine can be a big challenge.

IV. Methodology - DLOF (Distributed LOF)

According to the word done by Yizhou Yan et al, we apply DLOF from assigning all the points to the same machine first, and follow the 3-step pipeline:
Step 1: calculate K-distance of each point and materialize them as intermediate values.
Step 2: calculate LRD, which is the inverse of the average reachability distance based on the number of nearest neighbors of p.
Step 3: calculate LOF of each point from the intermediate values saved in previous steps.
We use the concepts Intermediate Value Management and Invariant Observation proposed by Yan .
We noticed that although each step requires different types of intermediate values, these intermediate values are only related to the direct kNN of p. Given a point p, as long as the intermediate values associated with the kNN of

p are correctly updated and passed to the machine that p is assigned to in the next step, the LOF score of p can be correctly computed without having to be aware of its indirect kNN.

We applied the whole concepts with three mapreduce jobs as following figure:



K-DIS

Mapper
output(k, v):
(_, point)

Reducer
output key:
- point(i)

output value:
- neighbor(i)
- neighbor(i) k-distance
- dis(p, neighbor)
- Nreach (cnt of neighbor)

LRD

Mapper output
output key: point(i)
value:Math.max(Kdis,distop)/Nreach

Reducer
- output key: point(i)
- output value:
- 1/sum(Math.max(Kdis,distop)/Nreach)

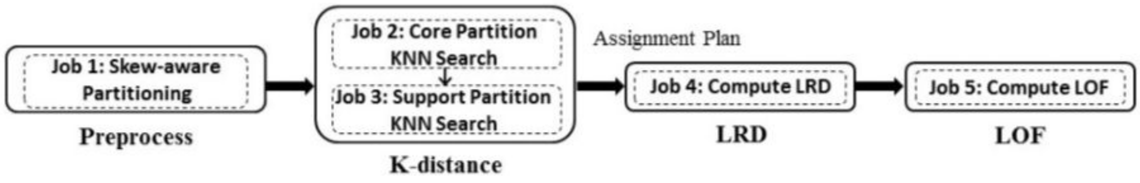
LOF

Mapper input:
output from k-dis
Mapper output
output key: point(i) value: neighbor(i), Nreach

Reducer
output key: point(i)
output value:
LOF = sum(neiLRD/myLRD/Nreach)

V. Optimization

To optimize D-LOF, we implement the DD-LOF to bound the support points. It decomposes the KNN search into multiple rounds to minimize the number of points that introduce data duplication. The following figures shows the general schema of the framework.



Job 1: Skew-aware Partitioning
Preprocess
Job 2: Core Partition KNN Search
Job 3: Support Partition KNN Search
K-distance
Assignment Plan
Job 4: Compute LRD
LRD
Job 5: Compute LOF
LOF

In the preprocess step, we divided the whole dataset into grids, and group the points to each grid. According to three conditions, we set different flags to each point. Flag 1 means that it is able to find all neighbors in its belonging grid, while Flag 2 means that its k distance is shorter than the distance of point p to the grid line, which indicates that the k distance needs to update. After this step, each point will return the list, contains its flag, grid, neighbor list.

Next step, we mainly focus on updating the k distance. First, we considered 8 conditions, by considering how many other grids the circle with radius r (as the below figure indicates) touches. The most complex case is the radius happens to be diagonal of the grid.

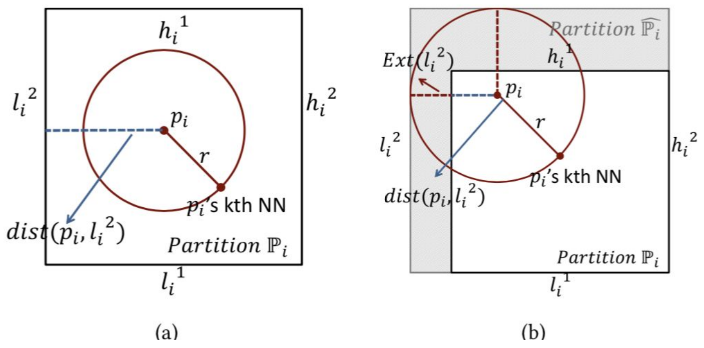


Fig 1. DDLOF: Supporting Area

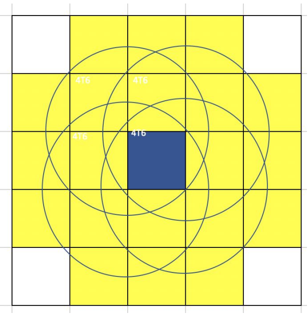
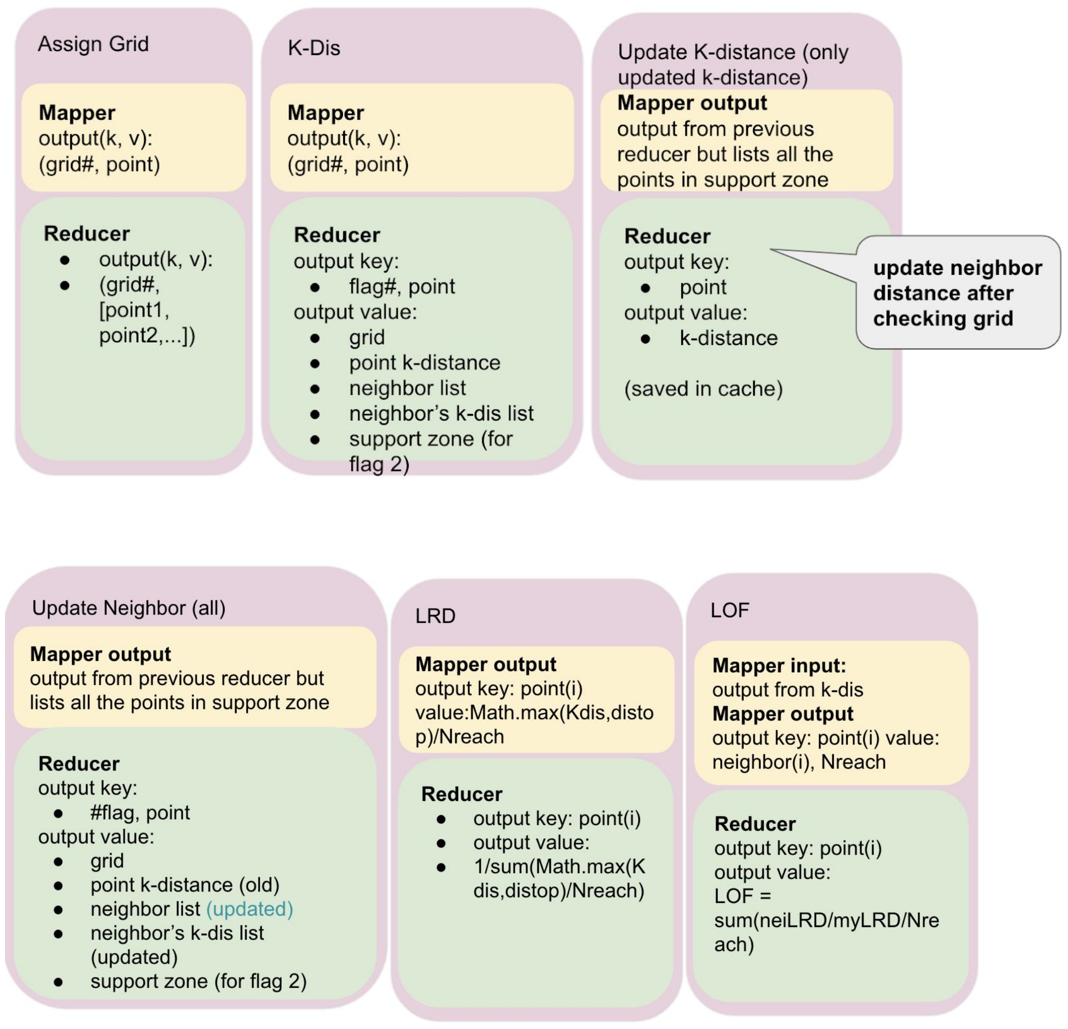


Fig 2. Different Conditions of Supporting Area

After we updated k-distance, the neighbor information also needs to be updated. When both k-distance and neighbor list has the latest version, it’s ready to compute LRD, and finally LOF.

We applied the whole concepts with six mapreduce jobs as following figures:



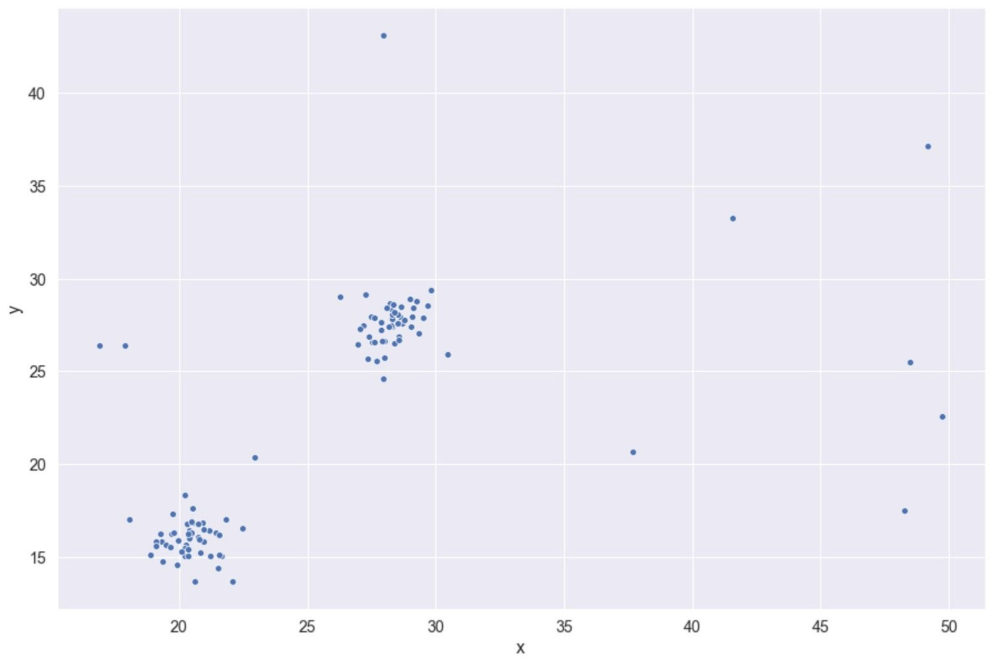
VI. Experiments and Results

For the data generation, we constructed a Python function that allowed a defined number of random points to generate within a defined region. In the code, we simulated clusters using make_blobs function in sklearn by defining the parameters of clusters. Lastly, to ensure outliers in the dataset, some points were randomly distributed across the whole region.

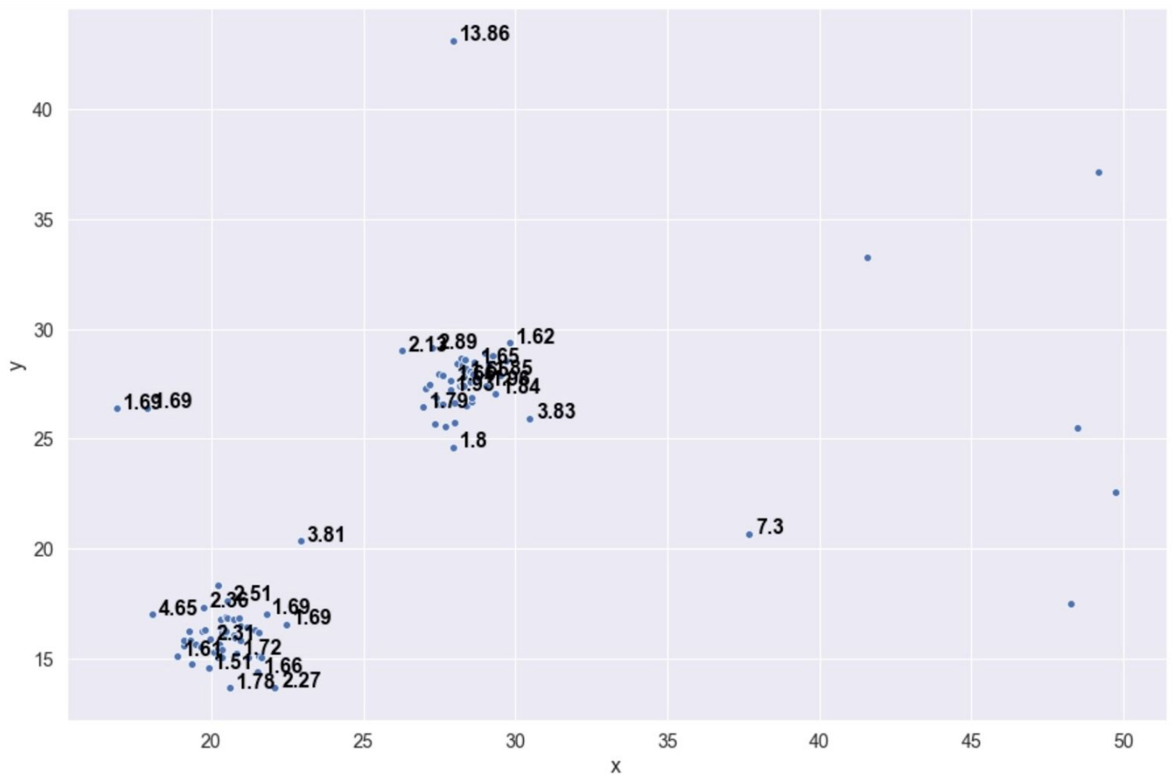
We generated two datasets for test convenivence. One is only having 110 points with 10 outliers, another one contains 1,0,50 points with 50 outliers.

6.1 Small dataset

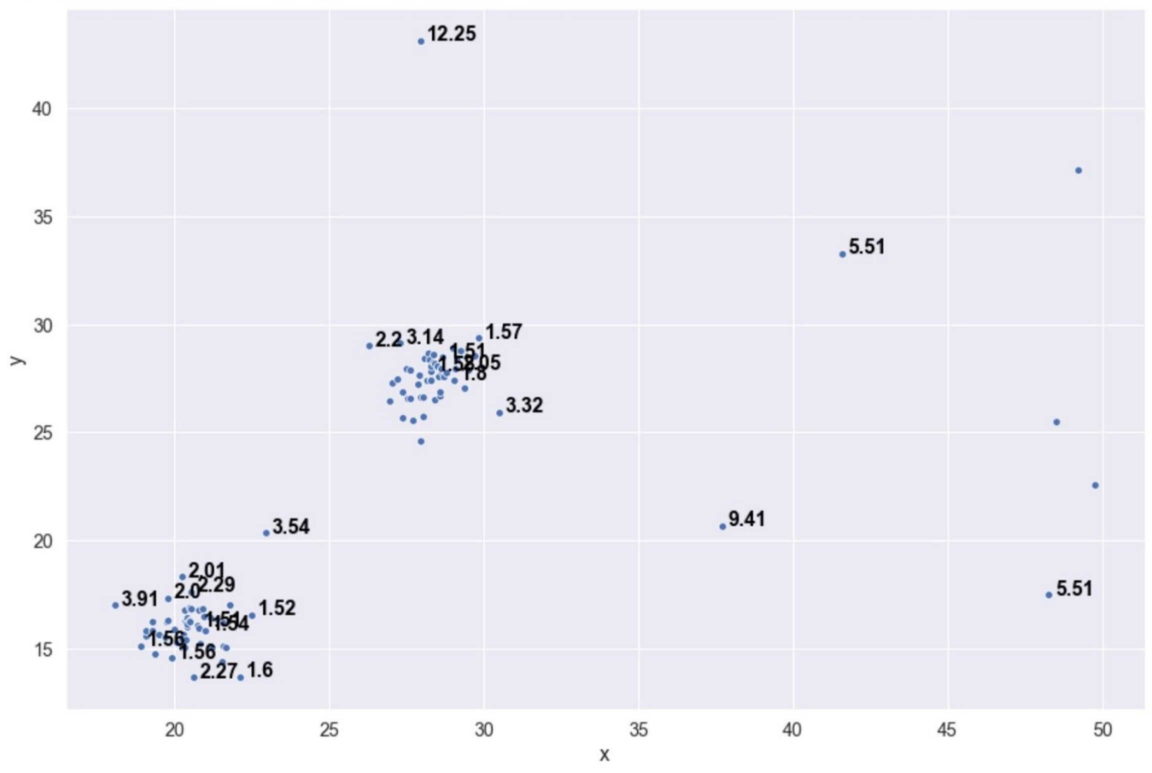
The dataset is visualized in a scatter plot below.



Implementing DLOF:
We filtered points whose LOF value is greater than 1.5. We can see that the points with large LOF value 13.86 and 7.3 fit into the outliers. However, this algorithm fails to capture the rightmost points. The reason maybe is that when k is little (we chose 2 in this case), these points share similar sparse density with its neighbors, which leads to the LOF is near 1.

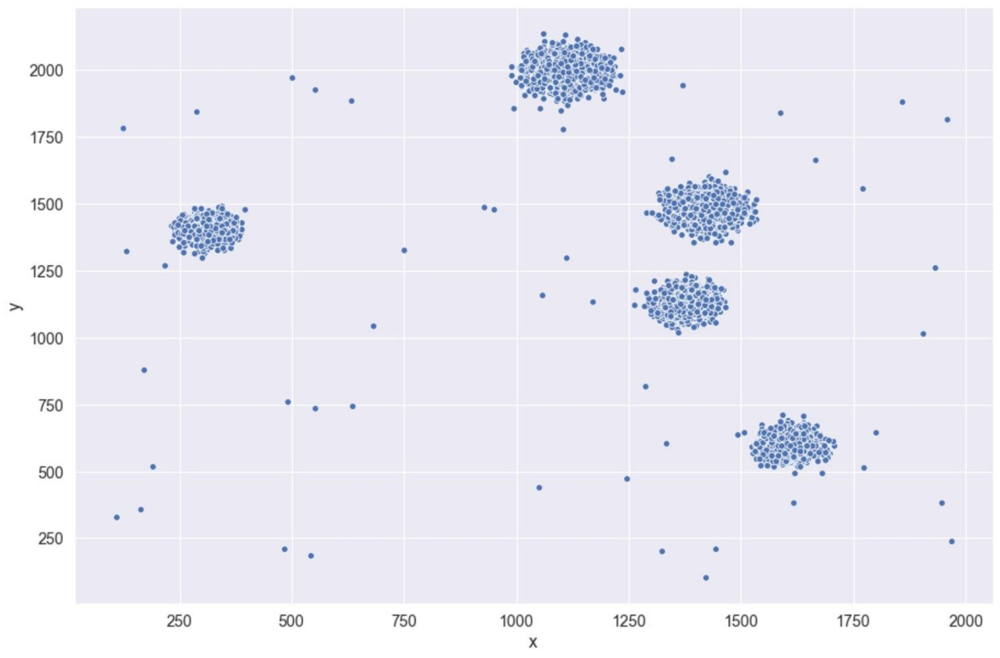


Implementing DDLOF:
Comparing to DLOF, DDLOF can capture the few rightmost points, but it still fails to the farthest points. It may have the same reason as DLOF. This indicates that choosing k-value is critical to outlier detection. The small k-value may classify the farthest points into non-outlier, while the larger k-value may make the points belonging to one cluster have the relatively large LOF. So, how to choose k-value can be a trade-off.



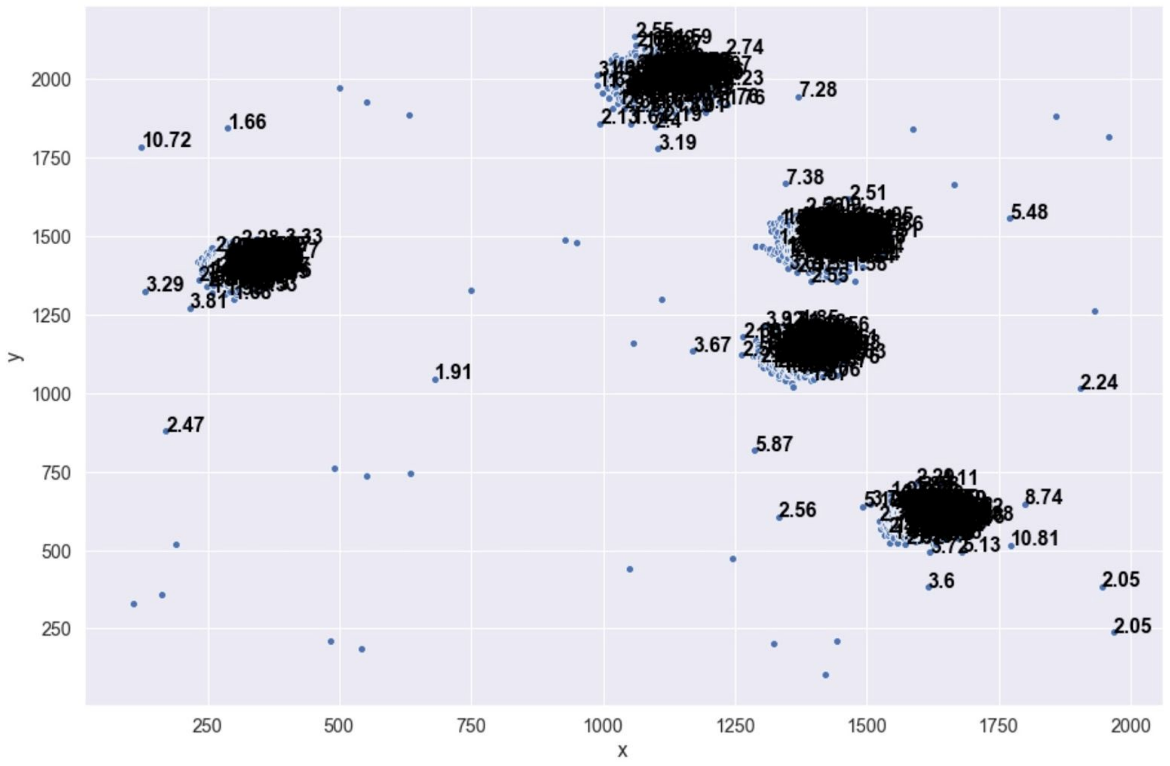
6.2 Large dataset

The dataset is visualized in a scatter plot below.



Implementing DLOF:

The following figure marks the points whose LOF value is greater than 1.5. DLOF successfully captures few outliers, but fails to detect all of them. To be noticed, there are some points belonging to the clusters also has relatively high LOF. This may due to located on the edge of one high density cluster, the average of the ratio of the local reachability density of this point and those of o’s k-nearest neighbours may still large, even though they are close to its neighbors.



VII. Conclusion

In our project, we apply DLOF and DDLOF to detect outliers. The points which lay in the center of cluster mostly have LOF close to 1, while the points get higher LOF when they are far from the center of the cluster. Both the methods we applied performs the properties in finding outliers, while there still exists some missing outliers and misclassification. More work needs to be done regarding choosing the value of k, and processing the points in the edge of the cluster.

References:

[1] Y. Yan, L. Cao, C. Kuhlman, and E. A. Rundensteiner, “Distributed local outlier detection in big data,” in SIGKDD, 2017, pp. 1225–1234.
[2]https://towardsdatascience.com/density-based-algorithm-for-outlier-detection-8f278d2f7983