Github link : https://github.com/DarkKnight2000/KMeans

Analysis of Algorithms on K-means Clustering with outlier detection

C. Sai Sreenivas*
Department of Computer Science, IIT Hyderabad

K. Rishinandan†
Department of Computer Science, IIT Hyderabad

(Dated: April 3, 2020)

The clustering algorithm, K-means is known to be highly sensitive to outliers present in the data. Clustering in the presence of outliers has received a lot of attention from the data processing community. Our goal is to study different Kmeans-outlier detection algorithms and evaluate them on both synthetic and real-world data.

I. INTRODUCTION

The standard K-means algorithm is an iterative technique that aims to partition n data points into k (a hyperparameter) clusters such that the variance of each cluster is minimized. The basic idea is to initiate the k centers randomly. After assigning each point to the nearest center, we recalculate the mean of points belonging to each cluster and call these the new cluster centers. Different objectives can be used to determine clustering quality, the most common one being, k-means cost function which minimizes the sum of distances between data points to the nearest cluster center. We iterate till the objective has not improved or if the calculated centers are very near to present centers.

It is also important to note that the way we initialize the centers also causes different results. Choosing the centers as far away from each other as possible is the best choice [1]. K-means++ can be used to achieve exactly this. Finding the optimal solution to this K-means objective is an NP-hard problem, however efficient algorithms converge quickly to a local optimum. The main problem with this approach is that it assumes that every data point belongs to one and only one cluster which is not the case every time.

A. Outliers

Sometimes there may be inconsistencies in the data (possibly due to measurement error or some background noise), and it is desirable to detect and remove these data points (called outliers). K-means is known to be sensitive to outliers. They can affect the average calculations and give rise to an unreliable local optimum.

Consider [2] a dataset with 2 gaussian point clouds and point far away from both of them(outlier) and we want to find 2 clusters in the data using K-means. If we choose centers to be the centers of true clusters, then we have a very high cost because of the outlier. By moving the outlier farther away from the "true" clusters we can get arbitrarily high cost. Alternatively, K-means chooses one centroid to be the outlier and another centroid somewhere in the middle of remaining data, to get a better cost. This configuration clearly doesn't represent the underlying distribution. One way to tackle this issue is to detect these points and ignore them in the objective, thus allowing the algorithm to partition the dataset which is potentially noise-free.

As described in [3] the K-means with outliers problem can be defined as follows :

((k, l)-means): Given a set of points X = x1, ..., xn, a distance function $d : X \times X \to R$ and numbers k and l, find a set of k points C = c1, . . . , ck and a set of l points $L \subseteq X$ so as to minimize the error $E(X, C, L) = E(X \setminus L, C)$. where E(S, C) is the sum of distances of each point in S to the nearest cluster center in C.

The inherent difficulty of this problem is that:

(i) the outlier set L depends on the set of cluster centers C : the points in L should be those points that are "far away" from the points in C, and

(ii) the set of centers C depends on the outlier set L : if we knew the outlier set L we could run a standard k-means algorithm on the remaining points $X \setminus L$.

The best way to handle this problem is to first randomly initiate a set of centers C. Then calculate outliers Z. With remaining points we can update the set C, and loop until we C, Z converge. This is exactly what most of the algorithms do.

In this paper, we would like to look at 3 outlier detection algorithms each with their own advantages and disadvantages and summarize which algorithm is suitable for which kind of data.

* cs17btech11012@iith.ac.in
† cs17btech11021@iith.ac.in

II. ALGORITHMS

A. K-means–

This algorithm follows exactly the procedure we discussed above, by defining outliers to be the farthest points from their respective cluster centers and centers are updated in the same way vanilla K-means does, by taking mean of points belonging to each cluster.

It has two hyper-parameters k, l denoting the required no. of clusters and outliers respectively.

Algorithm 1 K-means– algorithm for outlier detection

Given X, k, l

procedure K-MEANS–
    Initialise C randomly
loop until convergence:
    Assign nearest cluster center in C to each point
    $Z \leftarrow$ Farthest points from their respective centers
    Re-calculate C using remaining points,
    by taking mean of points in each cluster

The paper also proved that this algorithm converges to a local optimum, by proving that after each iteration the cost reduces and there exists a minimum cost, so the algorithm converges to a local minimum.

B. Local search outlier detection algorithm

The algorithm proposed in [4] is summarised as follows: It is based on a local search process wrapped by an outlier removal step. Starting with an arbitrary set of k(no of clusters) points, the algorithm iteratively checks if swapping one of the current centers with a non-center would improve the objective, and makes the local step if it is profitable. Once the algorithm has converged on a locally optimal solution, it checks if any further local swap, with the additional removal of z outliers, can significantly improve the solution. If such a swap exists, it performs the swap and removal and goes back to the simple local search mode. This algorithm takes as input, the desired no of clusters (k) and the no of outliers (z)

Algorithm 2 local search algorithm for k-means with outliers

Given X, k, l, ϵ

procedure LS-OUTLIER
    Initialise C randomly
loop until cost hasn't improved more than by a factor ϵ:
    Check if adding extra outliers reduces cost
    Check if swapping a non-center and center improves cost

As the algorithm looks at all possible center sets, it has a quadratic term in no of data points in its running time approximation. In step 4, there is a chance that the algorithm discards more outliers than the specified no of outliers. Although there is an upper bound on how many points the algorithm discards as outliers (Lemma 4 in [3]), it generally discarded close to the specified number in our experiments. It also outputs the final centers of the clusters. As the algorithm looks at every swap possible, like a brute force method, it generally outputs the correct outliers but at the cost of runtime.

C. Outlier Removal Clustering

Outlier removal cluster or ORC (proposed in [5]) addresses the issue of overlapping clusters along with outliers. It removes the points in overlapping regions along with outliers. The paper defines an outlyingness measure for each data point. The higher this factor, the higher the chance that the data point is an outlier. First, the maximum distance between a cluster center and a point in the cluster is calculated(d_max). Then outlyingness factor of a point is defined as o_i = distance to its cluster center/d_max. We can see that theses factor always lies between 0 and 1.

The algorithm has 2 hyperparameters. I for no of iteration of filtering to perform, and T for the threshold on outlyingness factor. In an iteration, if a point has its outlyingness more than T, the point is discarded as outlier. In each iteration, the outliers are calculated using this threshold.

Algorithm 3 Improving K-means using ORC

Given X, I, T

procedure ORC
    Initialise C randomly by running multiple times
loop for I times:
    Recalculate outliers
    Recalculate centers after discarding the outliers

Instead of simply taking the z farthest points as outliers, this algorithm uses a relative measure (the outlyingness factor) and discards the points with values above a threshold. This setting is useful in cases where the clusters are not dense and distinguishing the actual points from outliers is difficult just by taking distance to the nearest cluster center, which was essentially the problem with the LS-outlier algorithm.

This algorithm is also effective in detecting "local outliers". Consider an example where we have 2 clusters one very dense(C1) and another being very sparse(C2). A point at distance 'x' from the C1 center can be considered an outlier while a point at the same distance from the C2 center may not be an outlier as it exhibits similar distances as its neighbors. So a better approach to detecting these local outliers is to use a local density-based factor [6]. As this is not desirable in some cases, we need to tune the hyperparameter 'I' to get the required results.

This algorithm can result in many false positives (data points are incorrectly labeled as outliers) even with slight variations in 'I'. The main problem with the ORC approach is that C should be initialized with some other

K-means algorithm. Consider the outlier example we discussed earlier. If C is initialized with the "bad" clustering in the example we discussed earlier, then this algorithm fails to detect outliers. So it needs another K-means type algorithm to work effectively.

III. EXPERIMENTS

The LS-outlier algorithm concentrates mainly on detecting the outliers whereas the ORC also gives the final cluster centers. Although LS-outlier outputs k centers, they are just potential centers within the data points. So after discarding outliers, we performed K-means with these centers as initial seeds. This makes it even for all algorithms in terms of runtime.

The LS-outlier algorithm is taking a lot of time even for comparably smaller datasets. It was also very sensitive to the hyperparameter ϵ. In some cases, the algorithm discarded more than required no. of points to reduce the cost and the results were not good. Due to these reasons we did not test this algorithm in our experiments.

Metrics: We compare the algorithms based on the criterion mentioned in both papers, as each criterion measures a different capability of algorithms. We use precision and recall from [4]. Assuming that we know the set of true outliers Z* and that an algorithm returns the set Z as outliers. The precision for the algorithm is the fraction of true outliers within the set of outliers claimed by the algorithm, similarly, Recall is the fraction of true outliers that are returned by the algorithm. These metrics together measure the accuracy of outlier detection. We also compared these values with vanilla K-means values.

And we use the Distance ratio measures from [3] to check the quality of the clustering of remaining points. R_N = Distance to nearest calculated center / Distance to actual center averaged over all non-outlier points. If this metric is ¡ 1, it means that the algorithm found a better set of centers than the true centers. R_O is the same as R_N but it is averaged over actual outliers. The greater this measure the further is the algorithms centers set from outliers.

We run experiments on both synthetic data. For each dataset, the hyperparameters were tuned to get nearly the same no of outliers.

A. Synthetic Data

We generate synthetic data as mentioned in [4]. We choose k real centers uniformly at random from a hypercube of side length 100. Centered at each of these real centers, we add points from a Gaussian distribution with unit variance. This gives us k well-separated clusters. Finally, we sample z outliers uniformly at random from the hypercube of side length 100. Some of these outlier points may fall in the middle of the already selected Gaussians. Due to this, once the entire data has been generated we consider the points furthest from the real centers as the true outliers.

We ran the tests on 3 types of datasets. A1 where all inter-cluster distance is very high and variance of each is very low. For A2, A3 we reduce the inter-cluster distance and increase variance slightly. So the data A3 is going to have more local outliers compared to A1 and A2.

These are the results we got for each dataset:

As our datasets do not count local outliers as outliers, the ORC algorithm looks like it performed badly on the A3 dataset because of precision and recall. But in reality it did a better job than Kmeans–. It easily outperformed Kmeans– on both A1 and A2 datasets. Kmeans– has a chance of converging to a "bad" local optimum if the initialization of centers is not good. It had very low precision in some cases. So to get consistent results, ORC algorithm is a better alternative.

TABLE I. Results after running the algorithms on A1

Metric	ORC	Kmeans–
Precision	1	0.8
Recall	1	0.8
R_N	3.91	40.85
R_O	3.64	49.13
Cost	175.63	2412.66

TABLE II. Results after running the algorithms on A2

Metric	ORC	Kmeans–
Precision	0.8	0.66
Recall	0.8	0.8
R_N	3.29	3.23
R_O	3.14	10.13
Cost	111.81	130.82

TABLE III. Results after running the algorithms on A3

Metric	ORC	Kmeans–
Precision	0.66	0.8
Recall	0.8	0.8
R_N	3.82	4.04
R_O	11.16	13.71
Cost	172.33	172.10

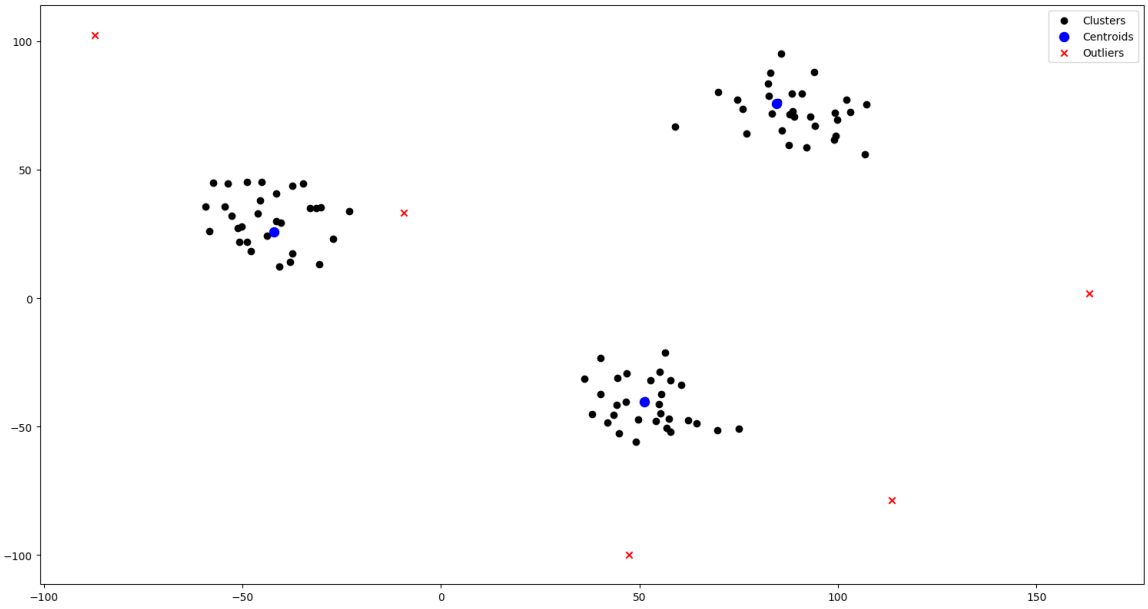

FIG. 1. A representation of A1 dataset

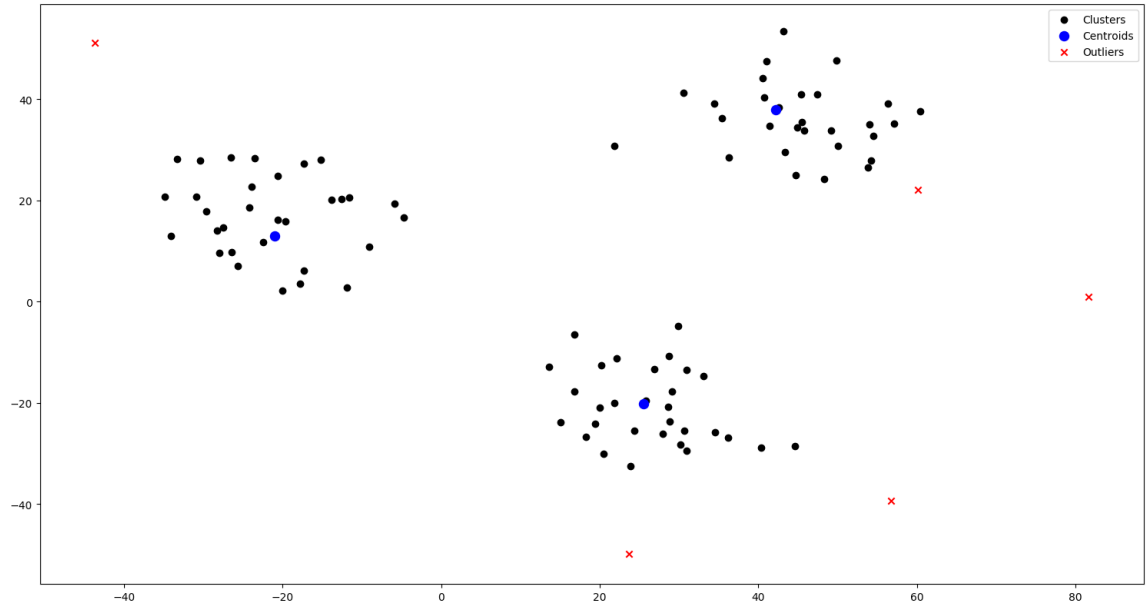

FIG. 2. A representation of A2 dataset

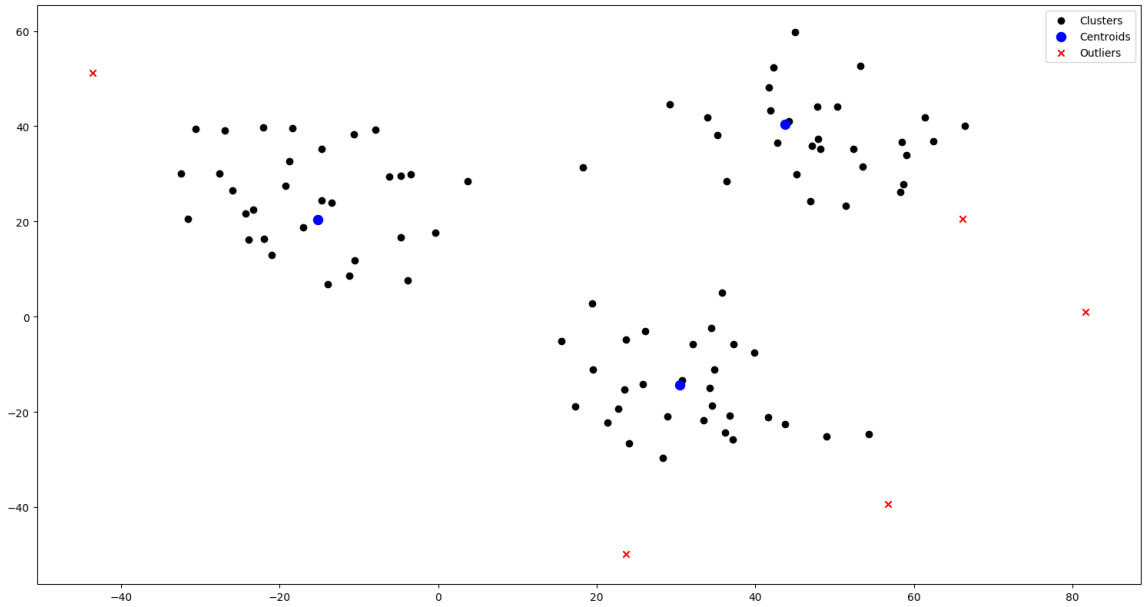

FIG. 3. A representation of A3 dataset

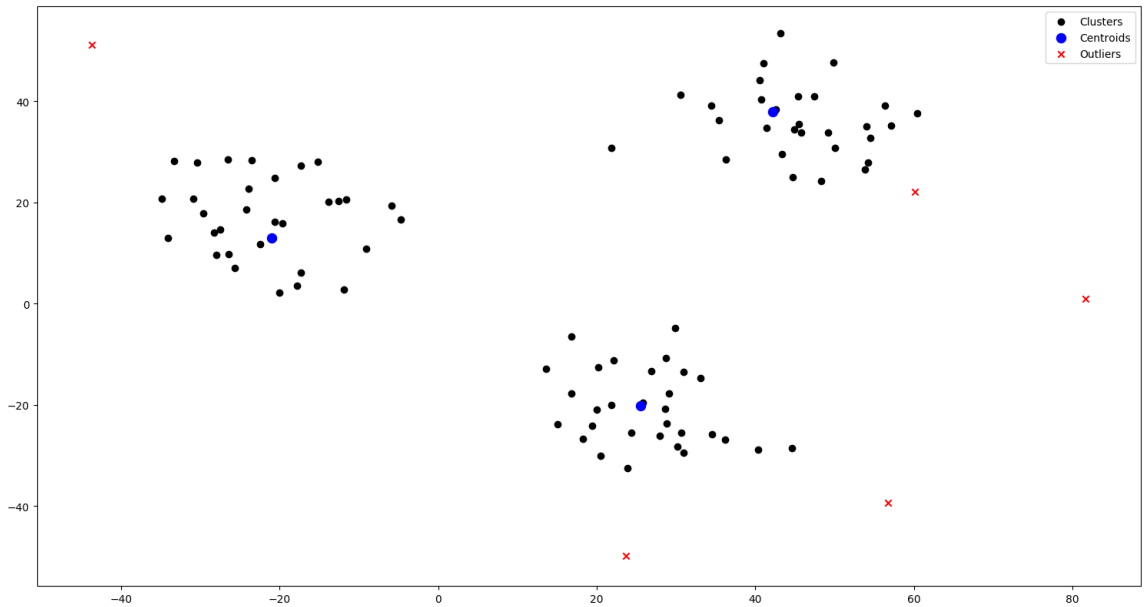

FIG. 4. A ouput representation of ORC Algorithm for A2 Dataset

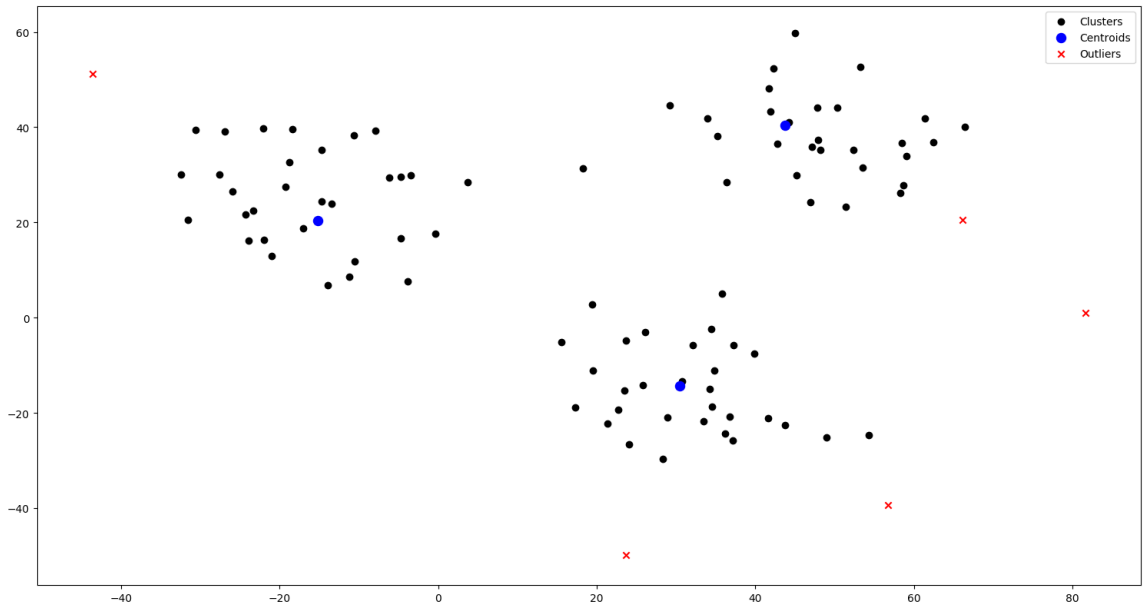

FIG. 5. A output representation of Kmeans– Algorithm for A3 Dataset

[1] Extract from https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a.
[2] Example from https://stats.stackexchange.com/.
[3] S. Chawla and A. Gionis, K-means-: A unified approach to clustering and outlier detection, , 189 (2013).
[4] S. Gupta, R. Kumar, K. Lu, B. Moseley, and S. Vassilvitskii, Local search methods for k-means with outliers, Proceedings of the VLDB Endowment **10**, 757 (2017).
[5] V. Hautamäki, S. Drapkina, I. Kärkkäinen, and T. Kinnunen, Improving k-means by outlier removal (2005) pp. 978–987.