Avoiding Objects with few Neighbors in the K-Means Process and Adding ROCK Links to Its Distance

Hadi A. Alnabriss
The Islamic University-Gaza
Alremal Neighborhood
Gaza, Palestinian Territories

Wesam Ashour
The Islamic University-Gaza
Alremal Neighborhood
Gaza, Palestinian Territories

ABSTRACT

K-means is considered as one of the most common and powerful algorithms in data clustering, in this paper we're going to present new techniques to solve two problems in the K-means traditional clustering algorithm, the 1st problem is its sensitivity for outliers, in this part we are going to depend on a function that will help us to decide if this object is an outlier or not, if it was an outlier it will be expelled from our calculations, that will help the K-means to make good results even if we added more outlier points; in the second part we are going to make K-means depend on Rock links in addition to its traditional distance, Rock links takes into account the number of common neighbors between two objects, that will make the K-means able to detect shapes that can't be detected by the traditional K-means.

General Terms

Data Clustering Algorithms, K-means, ROCK, Centroids' Initialization.

Keywords

Robust K-means, Rock links, Optimized K-means, Initializing K-means, electing centroids, Optimizing K-means distance measurement.

1. INTRODUCTION

K-means is considered as one of the most powerful and popular algorithms in the data clustering field[1], it has the ability to make good results in discovering similar data and separate them in different groups[2][3], one of the main attractive reasons for this algorithm –in addition to its power- its simplicity, it is a very simple algorithm, it depends on defining a number of clusters, then the algorithm will initialize a number of centroids, one for each cluster, every point in the dataset will be assigned to the nearest centroid; after applying n iterations, we'll find out that objects with the same characteristics are assigned to the same centroid.

Unfortunately K-means is vulnerable for some issues, its first problem is its sensitivity for outliers, a distant object can be assigned to a particular centroid, this outlier object will make the centroid move to the wrong way, many researches tried to solve this problem by detecting and removing outliers[5][7].

The second problem is its dependability on the distances regardless of any other factors, that makes K-means very weak when it faces non-globular shapes.

In Figure 1, it's obvious that object A is connected to the centroid C2, the connectivity here depends on the number of common neighbors between the object and the centroid, but if you measured the distance, the Euclidian distance for example, you'll find that object A is closer to centroid C1, in K-means it will be assigned to cluster 2, and that doesn't seem right.

The third problem with K-means is its sensitivity for the initial state of the centroids location, which will make K-means stuck in local minima instead of finding the global minima or the right groups.

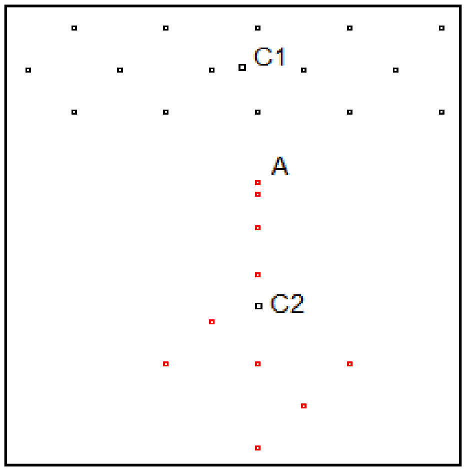

Figure 1, A is closer to C1, but it is more connected to C2

Another drawback in the K-means operation is its dependency on the number of clusters as an input, in data clustering we prefer algorithms that have only one input: the data, the algorithm should not know any idea about the number of clusters, the type of the data, the priority of the attributes, etc.

In section 3 we are going to present two techniques, the first is used for eliminating the effect of outliers in the K-means process,in this part we will depend on a function that decides if this object has to be eliminated or not, this function depends on the number of neighbors for this object, the second technique will involve the ROCK similarity measurement in the K-means process, this technique will help the K-means to detect clusters like the one in Figure 1.

The two techniques are applied for four artificial datasets and another two real datasets, the results show that the proposed algorithm is more robust for outliers, and can detect some types of non-globular shapes.

2. RELATED WORK

Many papers tried to make the K-means depend on other factors in addition to the traditional distance, Some algorithms tried to solve this problem by using novel ideas for measuring distance, ROCK[6] for example depends on the number of common neighbors to measure the similarity instead of the traditional distance to solve this problem.

The similarity in ROCK is called links and it depends on the number of shared neighbors between the centroid and the object.

Other papers proposed new ideas to cluster data, The proposed algorithm in [13][14] adopts a new non-metric measure based on the idea of symmetry.

Many papers proposed new ideas for initializing the centroids in K-means to make good results[4][11].

In [7] new techniques were added to eliminate outliers, these methods tried to make K-means more robust for outliers, to solve the K-means problem of sensitivity for outliers.

Other ideas were proposed in the field of prototypes' initialization[4][15], these ideas tried to solve the problem of K-means sensitivity for initial location of centroids.

3. PROPOSED ALGORITHM

In this section we are going to talk about our proposed algorithm, in these techniques we will try to solve the problem of outliers' sensitivity in K-means by making it more robust for outliers, this will be done by evaluating each object, and counting the number of its neighbors, eventually some objects will look like sole or isolated objects, that means that the closest neighbor is further than it should be.

The second technique will alter the way of measuring distances in K-means by adding some techniques inspired from the ROCK algorithm, the rock algorithm depends on a new concept of distance, it does not depend on the traditional concept of K-means, it counts the number of neighbors between two points, the more the number the strongest the relation and the similarity[6], that concept will be used in our technique to make K-means able to discover non-globular shapes.

We implemented our proposed techniques in Java, our application receives the dataset in a csv file, and the data should be normalized in another application in advance.

3.1 Eliminating Outliers

In this part we are going to depend on a novel function that calculates the number of neighbors for each point, if *f(N) = 5*, then each point should has at least 5 points as neighbors, if it has only 4 points, then it should be eliminated; our proposed function is:

$$f(N) = \ln(N)$$

Where N is the number of the points or objects in the dataset, so if we have 10 points in the data set, then each point must has at least 2 neighbors, when the number of the objects increases too much –a billion for example- this function will not make the number that we might think, so we have to solve this problem by raising this function to a particular power, say two, but we still have another problem!, what if we have 10 objects, and the number of wanted clusters is *k=5*?, it's obvious that one neighbor for each object is enough, so we have to involve the number of clusters in this function, so our improved function will be according to the formula:

$$f(N) = \frac{(\ln(N))^2}{k}$$

where N is the number of objects in the dataset, and k is the number of clusters.

The following table shows some samples for different N values and number of clusters k=2:

Table 1. f(N) for different datasets and k=2

N	Billion	Million	1000	100	20	10
f(N)	214	95	23	10	4	2

Table 2 shows the results for the same function but when k=3

Table 2. f(N) for different datasets and k=3

N	Billion	Million	1000	100	20	10
f(N)	143	63	15	7	2	1

Note:*f*(N)will be mentioned as ϴ in this paper.

But how can we decide if object x_i is neighbor of x_j or not?

In the beginning of our algorithm we measure a value called the average distance d_{avg}, this value calculates the average distance between each point and the others, divided by the number of the points in the dataset, then the average of the calculated distances will be calculated, by summing the distances and dividing them on the number of the objects.

Now, if the distance between objects x_i and x_j is less than d_{avg}, they are considered neighbors.

In the beginning of our algorithm we have to measure the d_{avg} value, and then count the number of neighbors for each object, and according to our function we have to eliminate each object with neighbors < f(N) or ϴ, that can be done by assigning these objects to cluster -1, which contains all the noise.

These eliminated objects will not take a part on the future process, that will make centroids follow the points of interest instead of following far objects, these objects can be assigned to the nearest centroid in the end of the process.

K-means initialization depends on selecting random objects as centroids, in our case, the points of cluster -1 –the eliminated ones- will not be selected as centroids, that will help K-means in selecting core points as centroids instead of wasting time and processing in the case of selecting outlier objects.

The following code illustrates the above process:

Code 1 – Eliminate Outliers

```
procedurecompute neighbors(S)
begin
1. Compute $d_{avg}$ , ϴ
2. Compute nbrlist[i] for every point i in S
3.      for i := 1 to n do {
4.      N := nbrlist[i]
5.      if(N<ϴ)
6.      cluster[i]= -1
7. }
end
```

The eliminated objects will not be involved in any part of our algorithm, they will not be selected as centroids, and they will not be used in the process of calculating the new location for a centroid.

3.2 Initializing K-means

In the beginning we have selected k random points as centroids, where k is the number of clusters, but in this case we have to run our application for many times till we have the right clustering, to avoid this problem –without involving ourselves in this field- we modified our application to select R points for each cluster randomly, and calculate the average of the selected points as our centroid, we can repeat that for the number of clusters.

For example if we have to select two clusters k=2 and the number of attributes p=2, then to calculate the first centroid we have to select two random points (when R=2):

P1={ 5, 8}

P2={ 7, 10}

The first centroid M1 is the average of the selected points:

M1 = { (5+7)/2 , (8+10)/2 } = {6, 9}

The second centroid will be calculated by selecting other two random points, and calculating their average.

If we changed R to be 3, then we have to select three random points and calculate their average.

In our Java application we found that selecting two random points is enough, we implemented that and it showed good results, now we can get the perfect solution without running the application more than once or twice at most.

Selecting more than one point will give us more chances that the centroid is located in the middle of our map, then when we start the K-means algorithm, each centroid will rush toward its right cluster.

3.3 Modifying K-means distance measurement

In this part we will involve the ROCK links concept in our work, but we will depend also on the traditional K-means similarity in measuring the distance between two points, we can use the Euclidian distance:

$$|x-y| = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

In ROCK, links are the number of shared neighbors between two objects, so if object A has the set of neighbors *{ d, e, f, g}* while object B has the neighbors *{ f, g, h, i, j}* then *link{A, B} = |{f, g}| = 2.*

To implement this part we have to build two matrices with size*k * n*, the first matrix will be for the Euclidian distance between each point and each centroid, the second matrix will be for the number of common neighbors between each object and each centroid.

$$\begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{k1} & \cdots & x_{kn} \end{bmatrix}$$

After preparing these matrices we have to normalize them by changing the values in the two sets to a value between 0 and 1, this can be done by dividing the whole matrix by the maximum value in it.

Note that the first matrix is distance while the second matrix is similarity, so we have to change the first matrix to a similarity matrix, this can be done simply by subtracting each value from one.

After preparing the two matrices we can sum them and divide them by 2, that will give us a new similarity value depends on the distance between any two points and the number of the common neighbors between them.

The following formula describes the summation of the two normalized matrices, the result is our final similarity matrix.

Final_sim[][]=((Norm(links[][])+Norm(similarity[][]))/2

Now we can use the final similarity to assign objects to the nearest centroid.

The following code illustrates this process:

Code 2 – Calculate similarity and links

```
procedurecompute shared_links(S) between nodes and each centroid
begin
1. findNnbrlist[i] for every point i in S
2. findCnbrlist[j] for every centroid j in C
3. Set shared_link[i; j] to be zero for all i; j
4. Set similarity[i; j] to be zero for all i; j
5.    for i := 1 to C do {
6.    for i := 1 to S do
7.           calculate shared_neighbors[i][j]
8.    }
9.    for i := 1 to C do {
10.    for i := 1 to S do
11.          calculate similarity[i][j]
12.    }
13. Normalize(similarity)  // All values from 0 to 1
14. Normalize(shared_links)
15. final_sim =( shared_links + similarity ) / 2
16. assign nodes according to final_sim
end
```

3.4 Fixing eliminated objects

At the end of our operation we measure the similarity between each object in cluster -1 and each centroid, then we can assign each object to its nearest centroid.

4. SIMULATION AND RESULTS

We have implemented a Java code to test our algorithm, our code receives the data in csv files, each line is an object and each column is an attribute, though our application accepts multi-dimensional data, we used only two dimensional artificial datasets that can be shown in 2D shapes.

We also brought two real datasets for testing, the first one is the Iris and the second is the glass identification dataset.

4.1 Artificial Datasets

To test our algorithm we used four artificial datasets, we began our tests with two simple datasets to see if our algorithm is working as we planned or not, the results of the first data set is shown in Figure2.

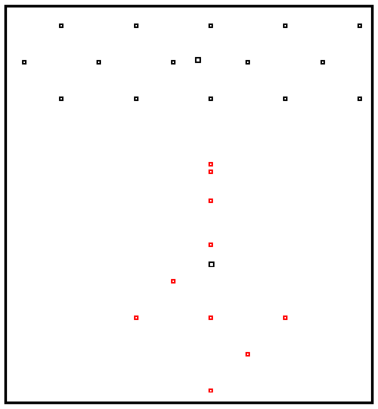

Figure 2, Simple Artificial dataset

In the above dataset we tried to make an object closer to the above cluster and more connected to the lower cluster, according to the results in Figure 2, our algorithm was able to take number of shared neighbors into account in addition to the distance.

In the second test, we tried to make a dataset that contains an object closer in distance to cluster A, but more connected by links –neighbors- to cluster B, the results are shown in Figure 3.

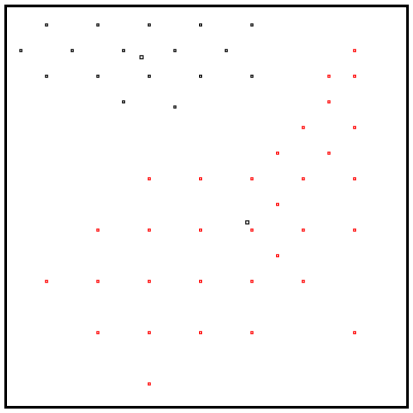

Figure 3, Simple dataset for testing the proposed Algorithm

In the dataset in Figure 3, we tried to add more objects and more complexity to our dataset, we can see here –also- that some objects are closer in distance to centroid A, but they are connected to centroid B because of the stronger connectivity.

In Figure 4 we can see a dataset distributed as two different-sized clusters, the first cluster is large while the other one is small, in this case K-means cannot catch the large cluster without including some objects from the small one.

Our algorithm in Figure 4 worked as well as we planned, it was able to detect two clusters with different sizes, as we see in the figure there are two clusters one is large while the other is small with some noise between them, the two clusters were separated successfully.

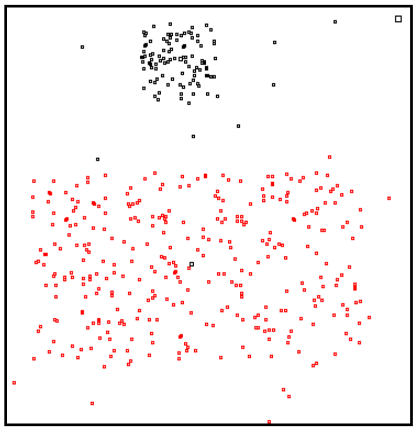

Figure 4, dataset with two clusters, small and large

The fourth dataset is divided in 3 clusters, with different sizes, our algorithm could detect the intended clusters.

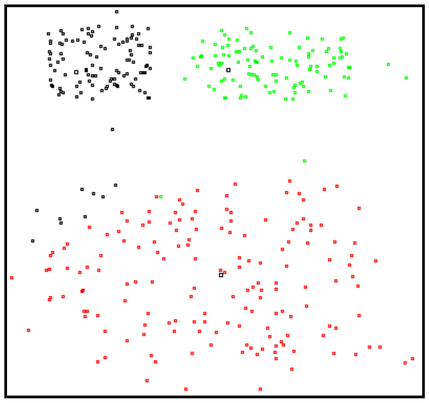

Figure 5, dataset divided in 3 clusters

In the fourth dataset we can see that some objects are not connected to the true clusters, this problem occurred because we still depend on the K-means traditional distance in addition to the ROCK links distance, that will make such errors occur.

In all the above cases we added some isolated objects to watch its effect on our algorithm, we found that the added objects were not involved in the calculations because of our proposed algorithm (section 2.1), which eliminates the outliers.

4.2 Real Datasets

We also applied our algorithm for two real datasets, the first is the common Iris dataset, and the second is the Glass Identification dataset.

The datasets are brought in different values and orders, to begin we normalized all the attributes values to be in the range zero to 1, then we entered the data to be processed by our algorithm.

Table 3 is composed of five columns, the first three columns describe the dataset, the number of attributes, the number of objects, and the intended clusters, the fourth column presents the number of objects that was not classified in the right sector, and the fifth column shows the error ratio which is the number of errors divided by the number of objects.

The Iris dataset is composed of 4 attributes, 150 objects, the objects have to be distributed in 3 clusters.

In this test we got 17 errors in objects classification, they were not clustered as the UCI website suggests them, and the error ratio is calculated as 11.3%.

The Glass Identification dataset is composed of 10 attributes, and 214 objects distributed in 6 clusters.

In the second test we got an error ratio of 16.8%, for 36 objects out of 214 were not clustered right.

The following table shows the results after running the clustering application for 3 times for the Iris and the Glass identification datasets.

Table 3. Real Datasets Results

	attributes	objects	clusters	errors	Error Ratio
Iris	4	150	3	19	12.6%
				21	14%
				17	11.3%
Glass Identification	10	214	6	41	19.1%
				39	18.2%
				36	16.8%

Table 4 shows the results for the traditional K-means algorithm, the results showed that the number of errors for the Iris dataset 32 is more than it for our algorithm which was 19; the second row shows the results for the Glass identification dataset, which has 95 errors out of 214 objects.

Table 4. K-means Results

	attributes	objects	clusters	errors	Error Ratio
Iris	4	150	3	32	21.6%
Glass Identification	10	214	6	95	44.8%

The results showed that our proposed algorithm optimized the K-means detection for the intended clusters in the above datasets, the optimization ratio in the Iris dataset was 33.3%, and more than 50% in the glass identification dataset case.

5. CONCLUSION

In this paper we have introduced a new technique to optimize the K-means measurement for distance, we added the ROCK links concept to the traditional K-means distance, the optimized K-means will take into account the connectivity between the centroid and each node, that has been applied by counting the number of shared neighbors between them, that helped the K-means to detect shapes that can not be detected by the traditional K-means.

We also introduced a new function to decide if the object is an outlier or not, if it was considered as an outlier, it will not take a part in the K-means process.

In our experiments we added outlier objects to see if it will affect the work of our algorithm or not, the final results showed that our proposed algorithm is more robust for outliers compared with the K-means traditional algorithm.

The results showed that our proposed technique detected the intended clusters in the Iris and the glass identification datasets, and the error ratio was less than the traditional K-means ratio.

6. REFERENCES

[1] J. Hartigan and M. Wang. A K-means clustering algorithm. Applied Statistics, 28:100{108, 1979.

[2] S. P. Lloyd. Least squares quantization in pcm. Technical note, Bell Laboratories, 1957. Pub- lished in 1982 in IEEE Transactions on Information Theory 28, 128-137.

[3] J. MacQueen. Some methods for classification and analysis of multivariate observations. Proceedings of the fifth Berkeley symposium on mathematical statistics and probability 1967.

[4] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In Bay Area Theory Symposium, BATS 06, 2006.

[5] Hautamaki, V., Karkkainen, I., Franti, Outlier detection using k-nearest neighbour graph. In: 17th International Conference on Pattern Recognition (ICPR 2004), Cambridge, United Kingdom (2004) 430–433.

[6] Sudipto Guha, Rajeev Rastogi, Kyuseok Shim, ROCK: A Robust Clustering Algorithm for Categorical Attributes.

[7] Ville Hautam¨aki, Svetlana Cherednichenko, Ismo Karkkainen, Tomi Kinnunen, and Pasi Franti, Improving K-means by Outlier Removal.

[8] Mu-Chun Su and Chien-Hsing Chou, A K-means Algorithm with a Novel Non-Metric Distance.

[9] Wesam Barbakh, Similarity Graphs.

[10] A. Likas, N. Vlassis and J. J. Verbeek, The Global K-means Clustering Algorithm. Pattern Recognition, vol. 2, pp. 451-461, 2002.

[11] Xiaoping Qing, Shijue Zheng, A new method for initialising the K-means clustering algorithm, 2009 Second International Symposium on Knowledge Acquisition and Modeling.

[12] G. H. Ball and D.I. Hall, “Some Fundamental Concepts and Synthesis Procedures for Pattern Recognition Preprocessors,” in Proc. of Int. Conf. Microwaves, Circuit Theory, and Information Theory, Tokyo, Japan, pp. 281-297, Sep. 1964.

[13] Mu-Chun Su and Chien-Hsing Chou, A K-means Algorithm with a Novel Non-Metric Distance.

[14] D. Reisfeld, H. Wolfsow, and Y. Yeshurun”,Context-Free Attentional Operators: the Generalized Symmetry Transform,” international Journal of Computer Vision, vol. 14, pp. 119 -130, 1995.

[15] Xiaochuan Wu and Colin Fyfe, On initializing prototypes for clustering.

[16] L. Breiman. Bagging predictors. Machine Learning, 24(2):123-140, 1996.

[17] W. Barbakh, M. Crowe, and C. Fyfe. A family of novel clustering algorithms. In 7th international conference on intelligent data engineering and automated learning, IDEAL2006, pages 283–290, September 2006. ISSN 0302-9743 ISBN-13 978-3-540-45485-4.

[18] M. Khalilian, N. Mustapha, M. N. Sulaiman, and F. Z. Boroujeni, "K-Means Divide and Conquer Clustering," in ICCAE, Thiland, Bangkok, 2009, pp. 306-309.

[19] Girolami, M. (2002). Mercer kernel based clustering in feature space. IEEE Transactionson Neural Networks (13(3)), 780-784.

[20] Kaufman, L., & Rousseuw, P. J. (1990). Finding Groups in Data. An Introduction to Cluster Analysis. John Wiley & Sons, Inc.