Data Structures and Algorithms Project Report
Spring 2016

Data Mining Using K-Means Clustering Algorithms

github: https://github.com/cmb499/DataMiningInFinanceApplications.git

By

Aneesh Abhyankar aneesh.abhyankar@rutgers.edu
Careena Braganza careena.braganza@rutgers.edu
Vishalsingh Hajeri vishalsingh.hajeri@rutgers.edu

Index

1. INTRODUCTION

1.1 Abstract

The amount of raw data stored in corporate databases is exploding with each passing day. From trillions of point-of-sale transactions and credit card purchases to pixel-by-pixel images of galaxies, databases are now measured in gigabytes and terabytes.

Raw data by itself, however, does not provide much information. And this unstructured data in its raw state can be thought of as an engineering obstacle towards extracting gold-mine of knowledge. This is where data-mining comes to the rescue. Data mining, or knowledge discovery, is the computer-assisted process of digging through and analyzing enormous sets of data and then extracting the meaning of the data. Data mining tools predict behaviors and future trends, allowing businesses to make proactive, knowledge-driven decisions. We will cluster this raw data in a meaningful way and apply it towards a business application (Financial sector) so as to study the efficiency of applied algorithm and its effectiveness on the input data set and analyze results.

1.2 Application

Banks have huge databases and large amounts of information containing customer's details and the history of their transactions. Analyzing this data can be used by the bank to find ways of customer Profiling, which in turn can be used for increasing bank's business and profitability. Customer profiling integrates several aspects of customers' information into an evaluation, such as age, dependents, job profile, income, assets, liabilities, education, historical records, contact details, customer attractiveness, or customer satisfaction. One of the most important points for customer profiling is targeting valued customer and ensuring they are satisfied with the bank's operations and services so that they can be retained with the bank amid frequent poaching by the competitors.

1.3 Use case Description

1. Using the data distribution, bank may want to suggest the high net worth/higher account balance cluster customers from the saving account to opt for other products.
2. Campaigns for certain products and services can be targeted at certain types of customer segments. Cash flow information of these customers can be used to determine the appropriate time for having these campaigns. Thus, the bank can get better return on investment on these campaigns.
3. Finding anomalies in bank transactions based on clusters on previous transactions for detection of frauds, misappropriations and embezzlements.

2. Study of Algorithms

In this section, we outline the algorithms implemented in our project, beginning with the most basic implementation of the K-Means Clustering Algorithm.

2.1 Un-optimized K-Means Clustering

The algorithm proceeds as follows:

1. Initialize Cluster Centroids Randomly μ1, μ2, ... , μn
2. Repeat until convergence

>> For each data point i :

Find closest centroid and assign that data point to that cluster

>> For each cluster

Recalculate the value of centroid and update it

The algorithm processes each point and assigns it to a cluster so as to minimize the distance of that point from any other cluster. The algorithm continues to recalculate centroids of each cluster, converging if and only if the points stop changing their assignments to the clusters.

Implementation: The implementation of the un-optimized K-Means Clustering Algorithm relies on the presence of a flag which is set if a point changes its cluster on each iteration and reset at the beginning of the next iteration. Convergence is reached if the flag remains reset at the end of the current iteration, implying that none of the points have moved from their respective clusters.

Insight: While working with large datasets of the order of 100,000 a few stray points may not affect the correctness of our results. In many applications, exactness in clustering can be leveraged for a better run time complexity. Hence, it is worth our time to learn implementations which enable us to control the convergence of our algorithm.

2.2 Optimized K-Means

Having set the stage for the need of a more time-effective algorithm, we now proceed to the the Optimized K-Means Clustering approach where we control convergence of the algorithm using a cut off value. The algorithm follows on similar lines as the unoptimized k-means except that in each iteration, rather than checking a flag for movement of points, we calculate the distance a particular a centroid has moved. If this distance is less than the chosen cut off value, we say our algorithm has converged otherwise we repeat the algorithm steps until our convergence criteria is met. The K-Means algorithms fall prey to the curse of randomness, covered in detail in subsequent sections. Each time, the performance of the algorithms differs based on the initialization of the centroids. The sensitivity of the algorithm on centroid initialization has been the centre of many research papers such as [3]. But for our project, rather than delving into these techniques, we decided to reduce the randomness in performance of the Clustering Algorithm while working with large datasets, which paved the way for our next algorithmic improvisation.

2.3 K-Means++

In this algorithm, we run the optimized K-Means on a small training dataset. On convergence, we record the final centroids and use these points as initialization centroids for our test data. The success and shortcomings of this approach is detailed in the Analysis provided in the next section.

2.4 Fuzzy C-Means

Though the focus of our project is studying the algorithmic performance of K-Means Clustering, understanding the advantages and disadvantages of our algorithm in terms of its application standpoint has generated interesting insights. The purpose of applying K-Means Clustering to our selected Banking dataset is to categorize the points, rather the people so that each audience may be targeted with a well suited set of products/services. Intuitively, these products/services may not be exclusive to a particular group which means a given person may be eligible for products/services in cluster A as well as B. We attempt to incorporate this possibility of belonging to more than one cluster by employing our final algorithm Fuzzy C-Means Clustering. Each point is associated with every cluster by a particular weight, determined by the closeness of that point to the cluster. In this way, we develop a membership matrix which describes the closeness of every point to every other cluster.

3. Experimental Configuration

1. Data Set
   Industry: Finance
   Maximum Number of instances: 100000
   Number of attributes: 7
   Attribute type: Numeric
   Un-normalized: Upper bound = 74990,
   Lower bound = 20
   Normalized: Upper bound - 80,
   Lower bound - 20

2. Attributes
   Age: integer not null
   Credit Rating: integer not null
   Bank balance: double not null
   Transactions/month: integer not null
   Types of Loans: integer not null
   Loan Amount: double not null
   Call Duration: double not null

3. The maximum size of the data set was 100,000.

4. The machine used to run our tests on was 2.7 GHz Intel Core i5, 8 GB 1867 MHz DDR3

4. Results and Analysis

In this section, we present our analysis organized as as per Time Complexity of our central algorithm, Optimized K-means Clustering, analysis of how our algorithm progressing by observing Shifts in Centroids and Euclidian distances and Comparison of of Un-optimized K-Means, Optimized K-Means and K-Means++. Finally, we compare Fuzzy K-Means with Optimized K-Means Clustering Algorithms.

Our motivation for concentrating on Optimized K-means where we control the convergence of the algorithm by checking a cutoff value for shift in centroids stems from the idea that while working with large datasets of the order of 100,000 a few stray points may not affect the correctness of our results.

Our main focus in the results and analysis section was studying the time complexity of the algorithms and also the effect of convergence of the algorithm on the centroids. We further go on to make comparative analysis of the algorithms employed. In the first part of this section we provide the overview of working of the optimized k-means and k-means++ algorithms. This will provide an overall understanding of these algorithms. In the next section we will provide the analysis of how the centroids shift as the algorithm approaches convergence. Next, we will discuss how the algorithms fare when compared with each other and finally, we will go on to discuss the fuzzy c-means algorithm.
Before we delve into the analysis, let us discuss a bit about our data set and the machine we ran the tests on. The data set that we obtained was from the machine learning repository of the University of California, Irwin. The data was basically customer information of a typical bank, complete with the age of the customer, his bank balance, credit rating, transactions that he does in a month, the number of loans that he has taken, amount that he has borrowed from the bank and finally the productive duration of call when he was last called by the bank representatives for an offer. The data set thus has seven dimensions in space. The data in un-normalized form is numeric and ranges from a minimum of 20 to a maximum of 74990. The size of the data set was 100000. The computer we used to run our tests on was 2.7 GHz Intel Core i5, 8 GB 1867 MHz DDR3

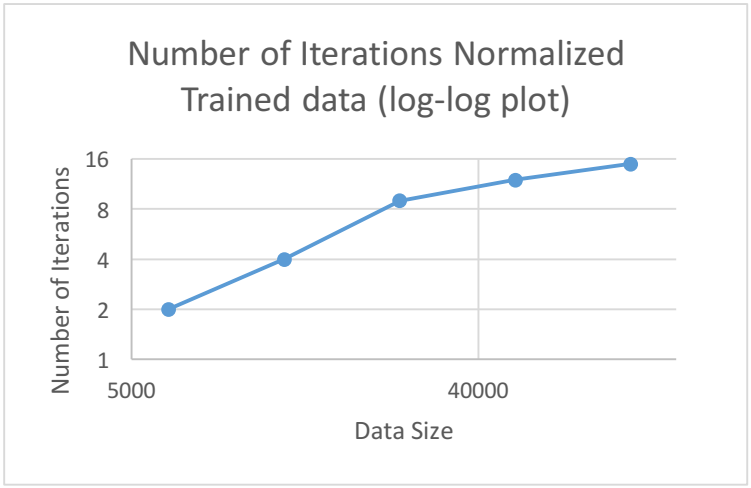
4.1 Overview

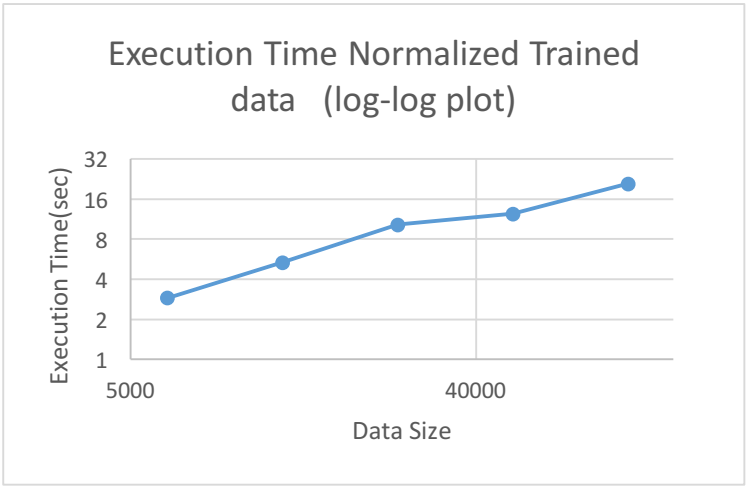
The table below shows the results obtained by testing the optimized version of k-means algorithm on normalized as well as un-normalized data. We varied the sizes of input data and observed the number of iterations it took for convergence and the total execution time of the algorithm. We repeated this for the k-means++ algorithm.

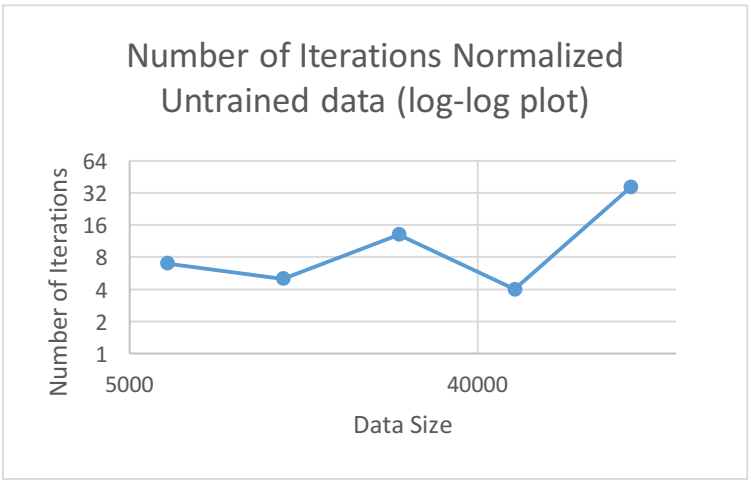
Data Size	Un-normalized Data				Normalized Data			
	Untrained		Trained		Untrained		Trained	
	Iterations	Execution Time	Iterations	Execution Time	Iterations	Execution Time	Iterations	Execution Time
100000	24	30.77077198	12	16.23740792	36	46.74927402	15	20.89065599
50000	22	26.366045	12	16.82754397	4	5.272058964	12	12.43200397
25000	21	27.16395283	13	18.82782602	13	12.99328208	9	10.30275607
12500	23	29.96616983	9	14.11300206	5	6.480598927	4	5.366280794
6250	13	16.98016286	6	12.46045589	7	9.055420876	2	2.890655994

Table 1: Overview of the optimized k-means algorithm (execution time in sec)

We observe from the table above that the k-means++ algorithm works better than the optimized k-means in terms of execution time and number of iterations for convergence. This emerges from the fact that in k-means++, we do not initialize the centroids randomly. We use 20% of the data for training, which is then used to ascertain the initial position of the centroids. This avoids the uncertainty that emerges from randomness, which we will discuss in 'discussions' section of this report. An example of randomness can be seen from the number of iterations of normalized untrained data, which are unusually small. This can be explained by the fact that selection of initial positions of centroids was done randomly. We can also observe from the table that the algorithm performs better when normalized data is input to it, as against un-normalized data. This happens because in case of un-normalized data one of the attribute may overpower the other, merely because the upper bound of values for that attribute happens to be significantly higher than that of others. In order to juxtapose number iterations for convergence and execution time of the algorithm, we should look at the following plots.







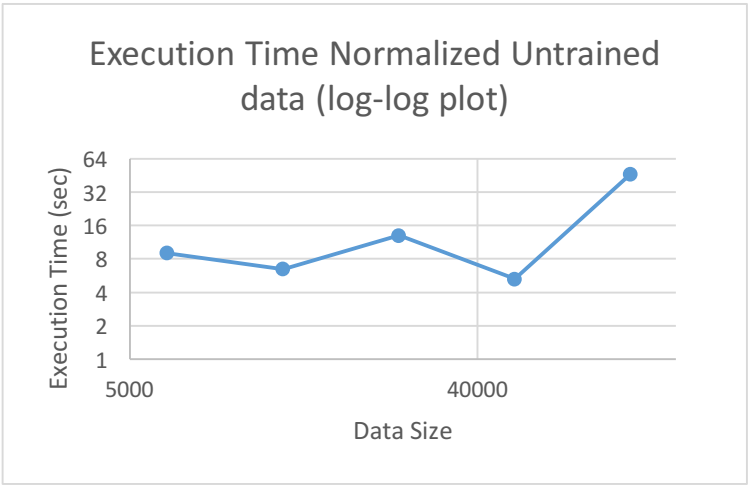


Figure 1: Number of iterations and execution time
for normalized trained data vs normalized untrained data

We can observe from the plots above that time it takes for the algorithm to execute depends directly on the number of times it iterates before achieving convergence. The execution time that we have obtained here, is the time it takes for the algorithm to run, and it does not include the time to read the data file. This is because in that case, the time to read the data file starts dominating as the data size increases. The figure below shows how training affects the execution time. We can see that the execution time of the algorithm decreases significantly for larges data inputs and not so much for smaller ones. This stems from the fact that training requires certain overhead and this becomes comparable to the actual running time in small data sets and hence adds to the running time. The effect of this overhead is not observable in large data sets.

4.2 Time Complexity of Optimized Cut-off

The following results obtained have been averaged out by running the algorithm several times, while fixing the centroids as k=4.
We can observe from the plots above that the execution time and n*k*d*I product follow a similar trend of growth. We see large numbers on Y-axis of the N*k*d*I vs data size plot, because it is just the product of the four quantities and does not have any unit. The plot is just presented to show the trend of expected and observed operation of the algorithm.

Data Size	Iterations	N*k*d*I
100000	36	100800000
50000	4	5600000
25000	13	9100000
12500	5	1750000
6250	7	1225000

Table 2: Execution time for varying data sizes for normalized trained data

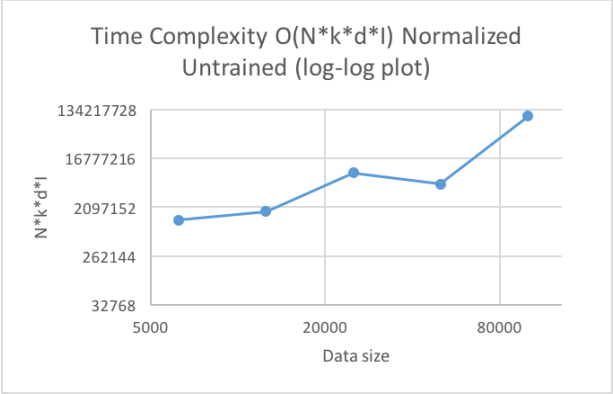


Figure 2: Execution time for varying data sizes for normalized trained data

Data Size	Execution Time
100000	4.293092012
50000	3.432003975
25000	3.302756071
12500	3.366280794
6250	20.89065599

Table 3: Execution time for varying data sizes for normalized untrained data

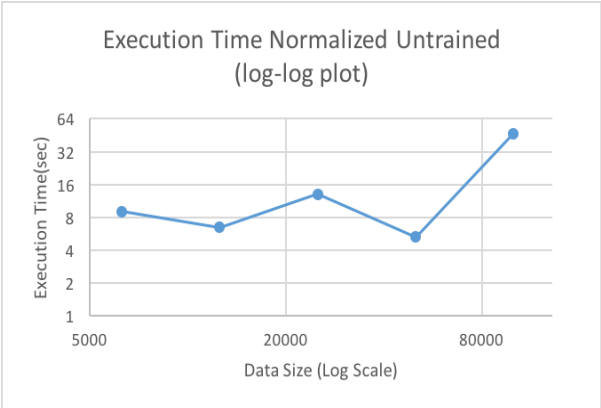


Figure 3: Execution time for varying data sizes for normalized untrained data

4.3 Shift of Centroids

The idea of k-means algorithm is that k number of centroids are initialized in some way before the algorithm starts running. The algorithm calculates distances of all points from all the centroids and assign points to the clusters depending on the closest centroid. The centroid values are updated or recalculated since the points in clusters may have changed positions. Now, depending upon how much the centroids moved from their earlier position, the algorithm decides whether to stop or to continue to the next iteration. Intuitively, we can understand that the centroids will shift more in the earlier stages of the algorithm and less as it matures. In order to qualify the argument, we observed the movement of all the centroids as the algorithm progressed for both untrained and trained data sets, and obtained the following result.

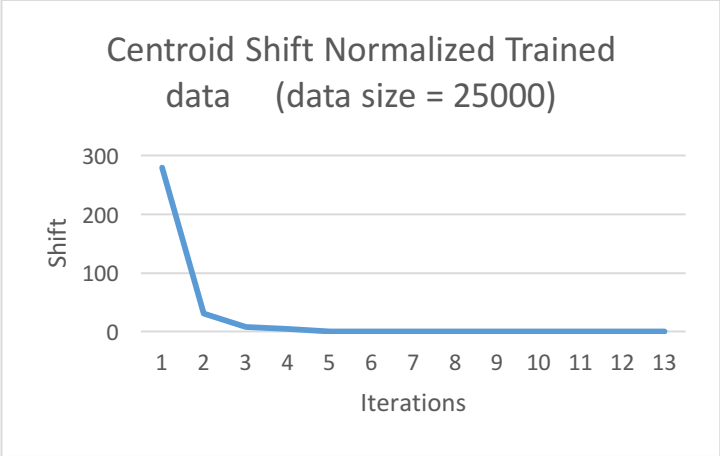


Figure 4: Centroid Shift for each iteration for normalized trained data

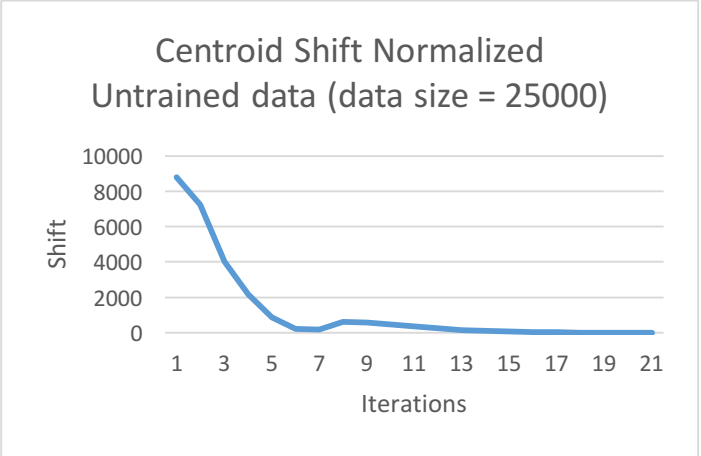


Figure 5: Centroid Shift for each iteration for normalized untrained data

The observation that we can make from the plots above is the shift is high initially for both trained and untrained data, but then it reduces gradually for untrained data and rapidly for trained data. This is as per our expectation, because for untrained data the centroids start off from random positions and hence take more time to get directed towards the resting position. This is also why it takes more number of iterations and consequently more time to execute the algorithm for untrained data as against trained data.

4.4 Euclidean Distance

Quite similar to the centroid shift is the behavior of Euclidean distance and quite understandably so, because the total Euclidean distance of all points from their respective centroids is the objective function that the algorithm seeks to minimize from this algorithm. The objective function is the one given below.

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

Where,

$r_{n,k}$ is a binary indicator which is 1 if n lies in cluster k and 0 otherwise

$\| x_n - \mu_k \|^2$ is the Euclidean distance of data point vector x_n from centroid vector μ_k (these vectors are 'd' dimensional)

Following is the plot for objective function plotted against number of iterations for all data sizes in case of Un-normalized trained data. We make an observation that very much like the shift of centroid graph, the objective function would start off with a high value as the centroids are not in their resting position. In a few subsequent iterations, he centroids shift by large amounts, thus largely decreasing the objective function. We thus find out the acceptable value of the objective function and based on that, define the cutoff threshold in terms of shift of centroid.

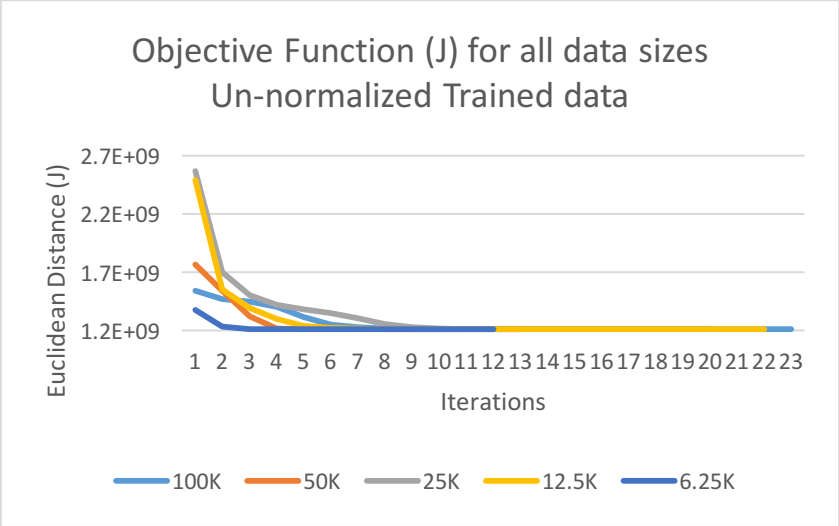


Figure 6: Plot of objective function against number of iterations for all data sizes

4.5 Comparison of Algorithms

The figure below shows the comparison of the algorithms implemented, namely un-optimized k-means, optimized k-means and k-means++ in terms of number of iterations required for convergence.

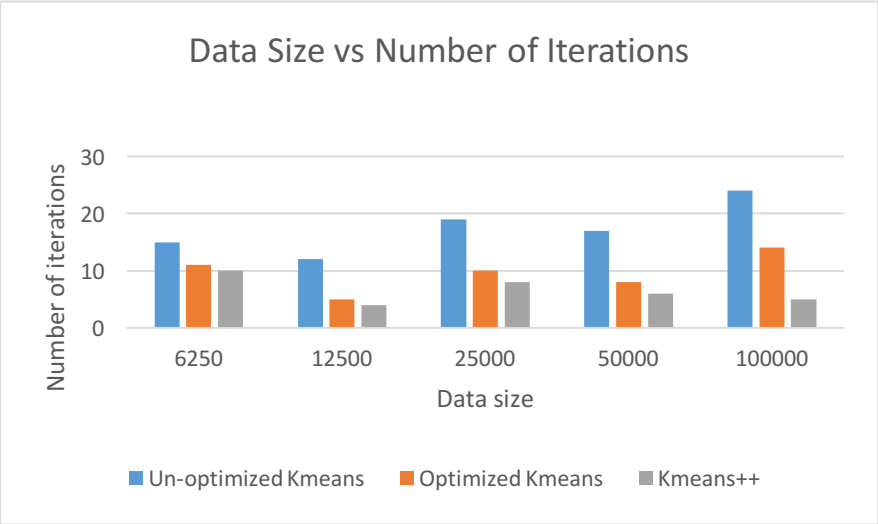


Figure 7: Comparison of algorithms in terms of number of iterations

We can see from the figure that least number of iterations are required by the k-means++ algorithm, i.e. the algorithm that does away with randomness with regards to initialization of centroids by performing some sort of training. We can also observe the stark contrast between the algorithms for large data sets. This is the phenomenon we discussed earlier that the overhead gets dominated by the actual running time of the algorithm. As we have established earlier number of iterations and execution time relate to each other directly. So, we can safely assume that the relative behavior of these algorithms will be similar in terms of execution time.

4.6 Fuzzy c-means vs K-means

The centroid of a cluster is the mean of all points, weighted by their degree of belonging to the cluster The value of wk(x) is related inversely to the distance from x to the cluster center as calculated on the previous pass. It also depends on a parameter m that controls how much weight is given to the closest center. This value of m is usually equal to 1 or 2. K-means algorithm performs better in terms of time complexity and space complexity. However, fuzzy clustering algorithm makes more sense of the clusters formed.

```
Membership matrix:
Data[0]: 0.162945 0.104590 0.349910 0.382555
Data[1]: 0.120697 0.088286 0.180995 0.610022
Data[2]: 0.169752 0.104509 0.436257 0.289483
Data[3]: 0.152146 0.099752 0.303101 0.445001
Data[4]: 0.150878 0.679012 0.098494 0.071616
Data[5]: 0.124378 0.738238 0.079810 0.057574
Data[6]: 0.051399 0.036500 0.082166 0.829936
Data[7]: 0.168778 0.102675 0.461398 0.267150
Data[8]: 0.376773 0.348265 0.167850 0.107111
Data[9]: 0.170588 0.662598 0.098644 0.068169
Data[10]: 0.206697 0.542930 0.143178 0.107195
Data[11]: 0.114745 0.058511 0.738935 0.087809
Data[12]: 0.210123 0.533661 0.146341 0.109875
```

Figure 9: Membership Matrix for Banking dataset

Cluster Size	K-Means	Fuzzy C-Means
1	1.349	1.349
2	1.386	2.322
3	1.597	4.516
4	1.847	6.784

Table 4: Execution time for varying data sizes for K-means vs Fuzzy C-Means sizes

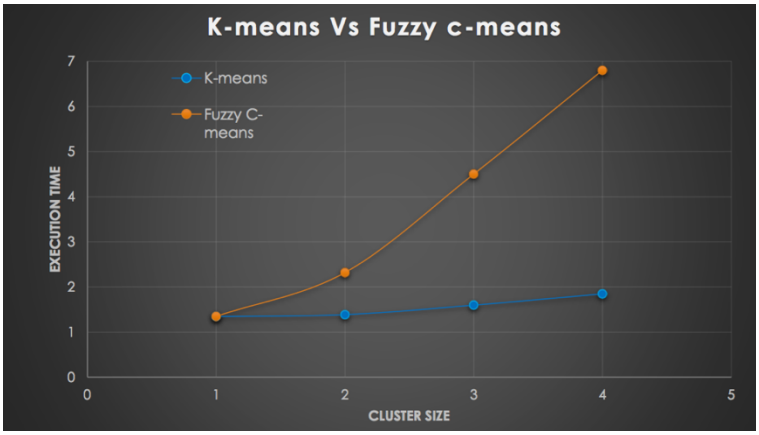


Figure 10: Execution time for varying data for K-means vs Fuzzy C-Means

5.Discussions

5.1 Curse of Randomness

In the K-means algorithm, initial centroid points are picked randomly each time the program is run. These Initial points largely dictate the number of iterations (i) occurring and they in turn dictates the run time of the algorithm when N (Data Set Size), K (Number of Clusters) and d (dimensions) are kept constant.
In our experiment we found that the final number of iterations and thus the execution time largely varied over a wide range due to the random initialization of centroid points.
Another problem associated with Random initialization of centroids is that it affects the final clusters formed. So every time we run our program we end up with different data points in the cluster.

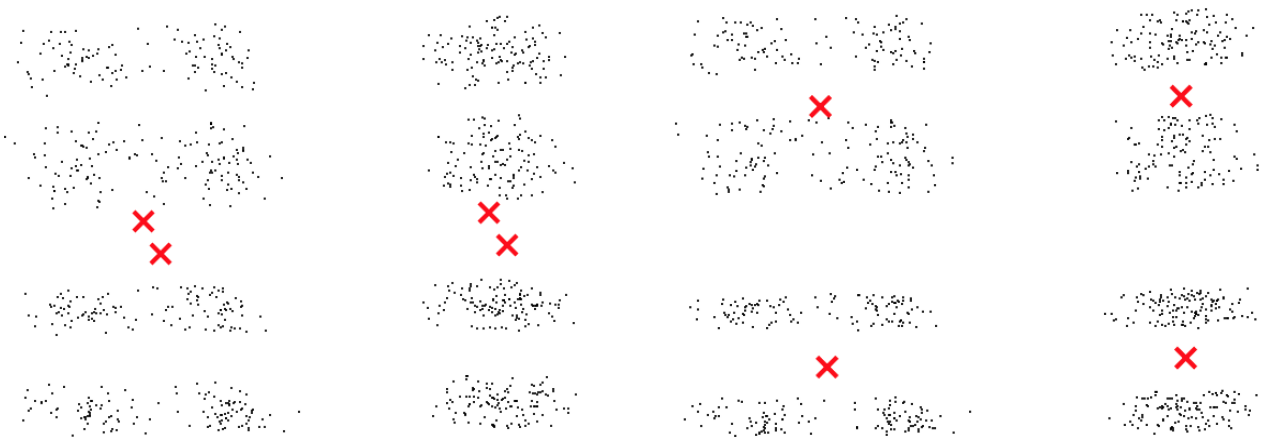


Figure 11: Random initialization of centroids leading to different clustering due to different final positions obtained for centroids

Possible solution

A heuristic analysis has shown that number of iterations are reduced if we choose the centroid points which are far apart from each other. There are multiple techniques for choosing the initialization points

1. Brute Force Method: It involves calculating the distances between every possible combination of points and then choosing tow pair of points which are most far apart.

Time Complexity involved with this method is $O(N^2)$, which is more than O(N*k*d*I). Hence it is not worth to choose this method.

2. Run the program on a training data set which is 20% of the original dataset. The final centroid points retrieved after the final iteration is used at the Initial centroid points while running the program on complete data set. The result is that the number of iterations decreases considerable with marginal additions to execution time.

3. Maintain a list of pairwise distance associated with every data point. This can be achieved in O(N). Then, iterate through this list to choose two pair of points with highest pairwise distance between them. Even though this method does not guarantee that the chosen points are most far apart, but it certainly saves us from the worst case condition.

5.2 Curse of High Dimensionality

Problems with high dimensional data set is that fixed number of data points become increasingly sparse as the dimensionality increases. To explain this, let's assume we have 10 data points. If we try and fit this into an array of size 10, we get no empty spaces. Now increase the dimension to 2 by considering a 10*10 matrix. If we fit our data points in this data structure, we get 90% empty space. Thus, as we go on increasing the dimensions, data points get lost in the space and the clustering results are not consistent.

5.3 Choice of K

The biggest question while trying to cluster data points is the choice of k (Number of clusters). If the dimensions of our dataset is 2, then it may be possible to decide on the choice of k by looking at the scatter plot of data set.

In the below example, it would make sense to choose the value of k = 3, since 3 clusters are distinctly visible. On the other hand choosing a value k = 5, 6 or any higher value may not produce logical results.

Hence, some type of analysis on the data set is necessary to decide on the choice of k, before we run our program.

Solution

Plot a histogram of the data set. The below graph is a histogram of an inconsistent data-set. The presence of one peak at the beginning depicts the presence of only one cluster. We can also conclude that there are some data points far away from the cluster which could be called as stray data points. These stray data points produce erratic results. Thus, it is not worth to implement k-means algorithm on such a dataset.

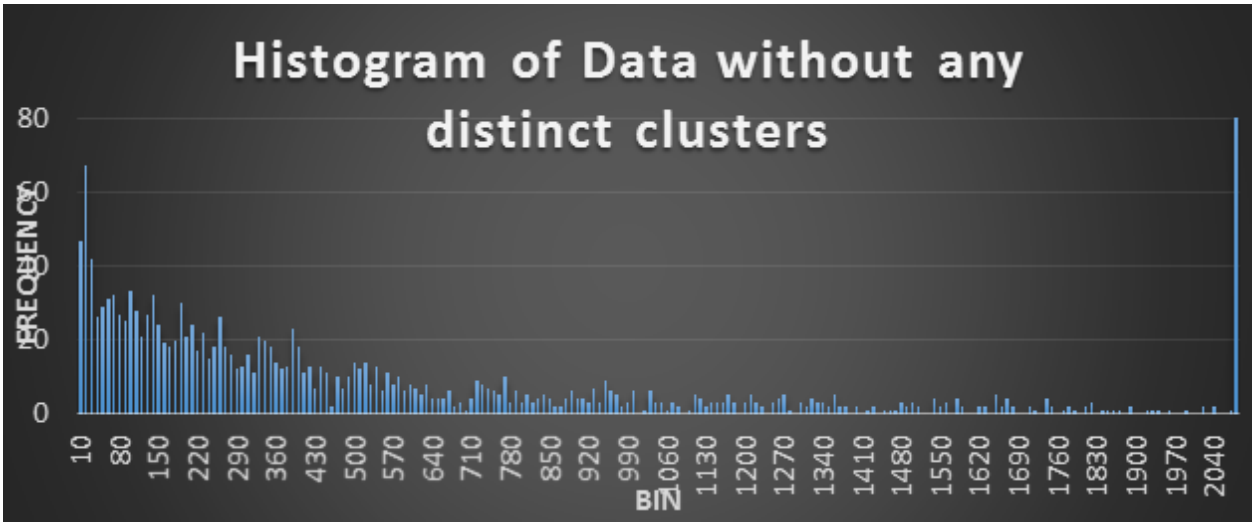


Figure 12: Histogram of data without any distinct clusters

The below graph is a histogram of original bank dataset which depicts peak at several positions. We can conclude by looking at multiple peaks that there are presence of clusters and it is worthwhile to run k-means algorithm on this data set. By experimentation we found that by keeping the value of k=4 , there were almost equal number of data points in each cluster. And hence the value of k-=4 was selected throughout the experiment.

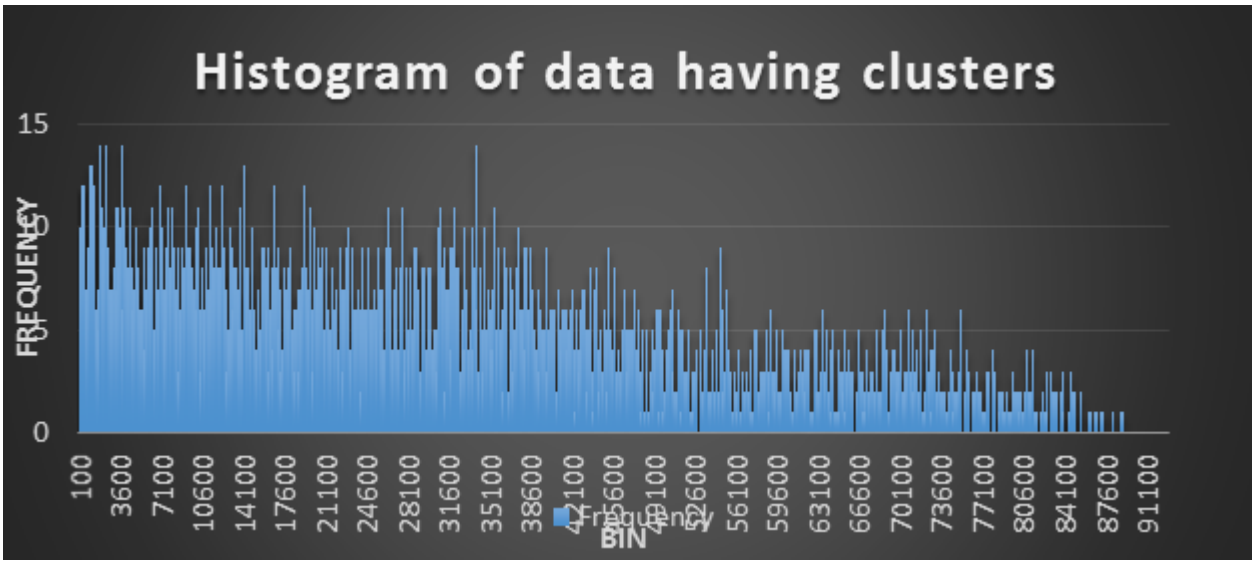


Figure 13: Histogram of data without any distinct clusters

6. Summary

Before Applying unsupervised clustering algorithm, we analyzed the dataset. We shortlisted few columns from a large available dataset which we thought would be crucial in customer profiling. Histogram analysis of dataset proves that there are indeed clusters of points spread in the space with negligible stray points and final choice of k (clusters) was decided by experiments to detect what value of k will provide us with logical clusters. Normalization of data is necessary, otherwise the clustering algorithm is mostly dominated upon by the column with larger values resulting in weighted clustering.

Un-optimized k-means continues until no point changes clusters and there is zero shift in cluster centroids. By plotting a graph of "Shift in centroid vs Number of iterations" we found that clusters have formed in the 4^{th} /5^{th} iteration itself and very few data points are changing clusters. Optimized K-means takes advantage of this fact and improves the run time of the algorithm by setting a cut-off value and making the algorithm stop early. K-means++ further improves the run time by choosing appropriate initialization centroid points.

The above mentioned algorithms fall under the category of hard clustering wherein all points exclusively belong to one cluster. The application may sometime demand that data points (usually the one's at the borders of the clusters) be part of multiple clusters. Thus we implemented Fuzzy C-means wherein all data points partially belong to all clusters.

The existing Customers in the database were divided into 4 clusters according to their shared behaviors and characteristics. Marketers then can infer the profiles of customers in each group and propose management strategies appropriate to each group. Bank thus gains the opportunities to establish better customer relationships.

7. References

[1] “Comparative Analysis of K-Means and Fuzzy C- Means Algorithms”, International Journal of Advanced Computer Science and Applications, Vol. 4, No.4, 2013, Soumi Ghosh, Sanjay Kumar Dubey

[2] “The Challenges of Clustering High Dimensional Data”, Michael Steinbach, Levent Ertöz, and Vipin Kumar

[3] “Efficient and Fast Initialization Algorithm for K- means Clustering”, I.J. Intelligent Systems and Applications, 2012, Mohammed El Agha, Wesam M. Ashour

[4] “Fuzziness parameter selection in fuzzy cmeans : The perspective of cluster validation”, Article in Science China. Information Sciences · November 2014, ZHOU KaiLe, FU Chao & YANG ShanLin