HW_K_Means
Professor: Dr. Thomas Kinsman
Student: Akash Venkatachalam

Abstract:

The idea is to implement standard K-means algorithm and find the number of clusters present in the given dataset. The given dataset has three attributes with all numeric values. We will extend the discussion by plotting the Sum of Squared Errors (SSE) for all the clusters versus K, the number of clusters. The value of K will range from 1 to 20, to see the variation in the plot and find the knee point value.

Overview:

This assignment is programmed in Python 3 language. It uses many libraries like csv for reading in the CSV file, math for using the square root value, sys for initializing the maximum size of integer, json was used for dumping and loading list to dictionary keys. Randint, numpy and stats were used for choosing random value, faster computation and finding the mode of array respectively.

We will implement the standard K-means algorithm and find the number of clusters present in the given dataset. This is done by plotting the Sum of Squared Errors (SSE) for all the clusters versus K, the number of clusters. The value of K will range from 1 to 20. The value of K = 1 gives a common basis of comparison for the other K values. Euclidean distance was used to find the distance between the points.

Assignment Explained:

The program starts with reading the csv file and loading them into a variable of list type. We now have all the data points in a list format. The given dataset has three attributes with all numeric values. Then we need to find some random points in the space and fix it as the initial cluster centroid. The number of random points depend on the size of cluster we want to choose.

After finding the random cluster centroids, we will cluster the entire dataset based on these centroids. This is done by finding the shortest distance between the cluster centroid and each data point and assigning them to the destined cluster.

The distance metric used for calculating the distance between cluster centroid and the data point is Euclidean distance. The general form of Euclidean distance *d(p,q)* is for n-dimensional space is given by:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2}$$

where *p* and *q* are the two separate corresponding points on each cluster.
Now all the observed data points are assigned to their destined cluster. We use a dictionary for storing the data point as key and its respective cluster as value. After this, we need to run the iterations until we find that the cluster centroids don't move anymore.

After this we calculate the SSE for each K value. For this step, we need to find the SSE value at least 1000 times, since the initial centroid is chosen at random. We then fix the SSE of that K value by finding the mode of these 1000 obtained SSE values. Using this SSE value, we plot the graph of SSE vs K, clusters. The initial choice of centroids can affect the output clusters, so the algorithm is often run multiple times with different starting conditions in order to get a fair view of what the clusters should be.

Conclusion:

From this assignment, I learned a lot about how the k-means, one of the popular clustering algorithm works. The initial choice of centroids can affect the output clusters, so the algorithm is often run multiple times with different starting conditions in order to get a fair view of what the clusters should be. I used Euclidean distance metric for measuring the distance between the cluster centroid and the data point. The break point occurred at K = 5, but for safety and for better accuracy of the data set, I would choose K = 7 as the number of cluster. The value of SSE at K=7 is 748.

I also learned how to use different libraries like stats, rand, json dumps and loads. I was using dictionary to store the data point and its cluster as key. Since the data point was in list format, I was having a hard time to use list as dictionary key. Luckily, I came across json dumps and loads which helped me resolve this issue.

Answer to questions:

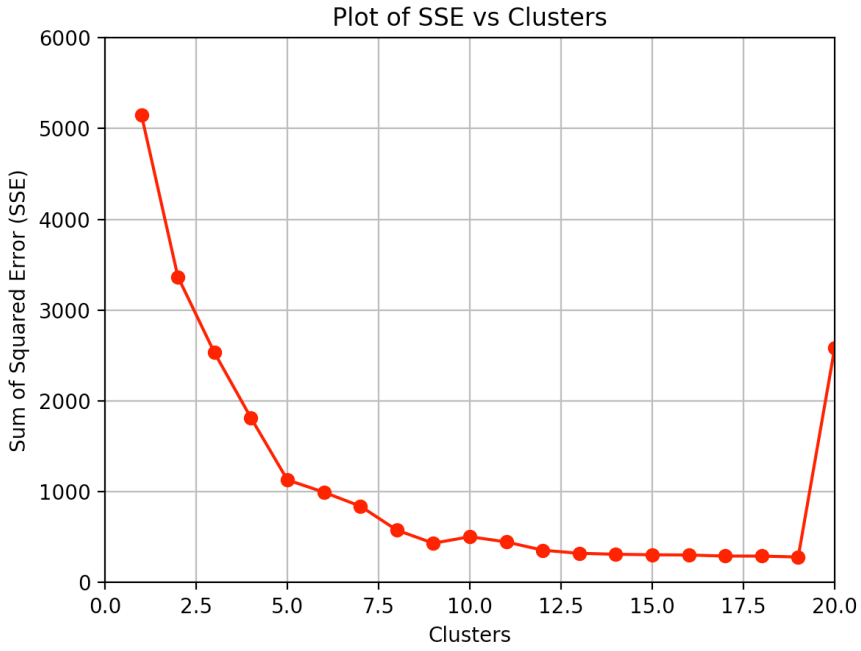
a) Did you do any pre-processing or noise cleaning of the data?
Answer: No, I didn't do any pre-processing of data. I used the given data and converted these data points into list.

b) Which algorithm did you implement?
Answer: I implemented the standard K-Means algorithm, by randomly assigning the initial cluster centroid points.

c) What distance metric did you use? Did you try using any other distance metrics and get different result?
Answer: I used Euclidean distance metric for measuring the distance between the cluster centroid and the data point. No, I did not use any other distance metrics.

d) What prototype did you use for each cluster?
Answer:
Here is the prototype I would choose for the dataset. Since my K = 7, I have 7 cluster centroids as follows:
[[2.198, 6.722, 3.010],
[5.035, 2.316, 4.889],
[7.688, 2.59, 7.684],
[2.502, 6.082, 7.349],
[6.362, 7.115, 7.813],
[7.732, 5.797, 2.275],
[2.018, 2.394, 5.302]]

e) Plot the SSE versus K.
Answer:



f) What value of K did you select and what was the associated SSE value?
Answer: The break point occurs at K = 5, but for safety and the better accuracy of the data set, I would choose K = 7 as the number of cluster. The value of SSE at K=7 is 748.

g) What are the resulting cluster statistics?
Answer:
The answer is printed as the output of my program. The value of cluster K can be changed accordingly in line 84 by altering the range value in the for loop. The last line of my program will print the SSE values.

h) What was the hardest part of getting all this working? Did anything go wrong?
Answer: I was using dictionary to store the data point and its cluster as key. Since the data point was in list format, I was having a hard time to use list as dictionary key. Luckily, I came across json dumps and loads which helped me resolve this issue. And hard part was running it for 1000 times x 20 K cluster values took a long amount of time.

i) What did you learn about all of this?

Answer: From this assignment, I learned a lot about how the k-means, one of the popular clustering algorithm works. The initial choice of centroids can affect the output clusters, so the algorithm is often run multiple times with different starting conditions in order to get a fair view of what the clusters should be. I also learned how to use different libraries like stats, rand, json dumps and loads.

Credits:

[1] http://www.saedsayad.com/clustering_kmeans.htm Used this website to understand the concept of K-Means in a much better way.

[2] Professor's lecture