**TASK 1:**

1. First, the centroids are initialized to (4, 6) and (5,4). The scatter plot looks as follows:

Chart, scatter chart

Description automatically generated

Next, using Manhattan distance for 1 iteration, the center is check:

Chart, scatter chart

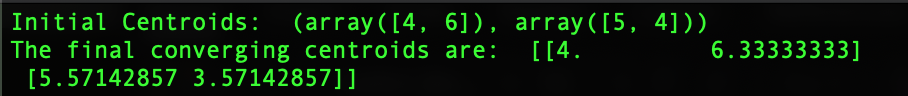
Description automatically generated

After 1 iteration, the distance of the centroids has slightly changed to (4, 6.33) and (5.77, 3.57).

Iterating until convergence, we get the final clusters as shown below. Convergence is proved on the second iteration.

Chart, scatter chart

Description automatically generated



1. Performing the same operations on the same centroids (4, 6) and (5, 4) with Euclidean distance, we get the following output after the first iteration. It can be seen that the centroids of the two clusters are now at (2.50, 6.50) and (5.75, 3.88).

Chart, scatter chart

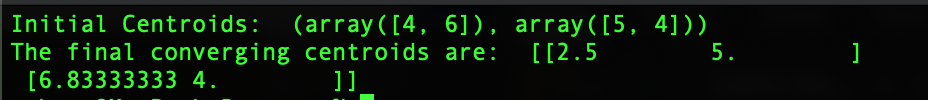
Description automatically generated

Upon convergence, the centroids are now present at (2.5, 5) and (6.83, 4). The final two clusters can be seen in the figure below. Convergence is proven on the third iteration.

Chart, scatter chart

Description automatically generated

The screenshot of the final converging centroids is shown below:



1. The centers are now changed to (3, 3) and (8, 3). The below graph shows the initial centroids.

Chart, scatter chart

Description automatically generated

After 1 iteration with Manhattan distance, we see that the centers are now at (2.5, 5) and (6.38, 4).

Chart, scatter chart

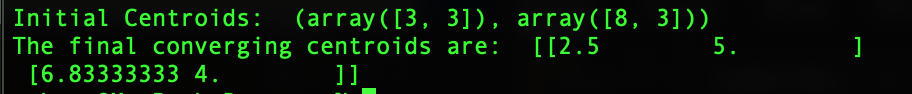
Description automatically generated

Upon executing until convergence, it is seen that the centroids don’t change any further. Convergence is proved on the second iteration. The final clusters are shown below:

Chart, scatter chart

Description automatically generated

Output screenshot showing the centroid points:



1. Considering the centroids at (4, 8) and (3, 2), the graphs looks as below:

Chart, scatter chart

Description automatically generated

After 1 iteration with Manhattan distance, the output looks as follows. The centroids are at (5.67, 6.33) and (4.86, 3.57).

Chart, scatter chart

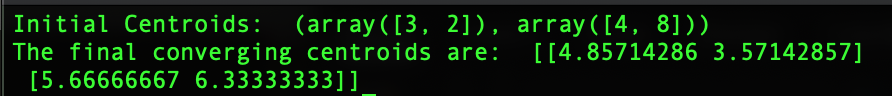
Description automatically generated

Upon executing until convergence, it is noted that the convergence is proved on the second iteration. The final graphs is as follows:

Chart, scatter chart

Description automatically generated

The screenshot of the converging centroids is shown below:



**TASK 2:**

K-Means is executed on Euclidean distance, Cosine distance and generalized Jaccard similarity. The number of categorical values in the iris dataset is 3. The class labels can be Iris Setosa, Iris Versicolour and iris Virginica. Hence, we will have three clusters. We also have 4 X features, sepal length, sepal width, petal length and petal width.

I have implemented K-Means on the iris dataset by picking centroids in three different methods. A) I have picked absolutely random centroids

B) I have assigned fixed centroids

C) I have picked one random centroid, C1 and then calculated the farthest datapoint from C1 to be my next centroid, C2. And for the final centroid, I have taken the largest average of the distance between C1 to the new point and C2 to the same new point.

All the outputs shown are when centroids are picked using method C. Outputs for scenario A and B can be check implement the code.

A circuit board

Description automatically generated

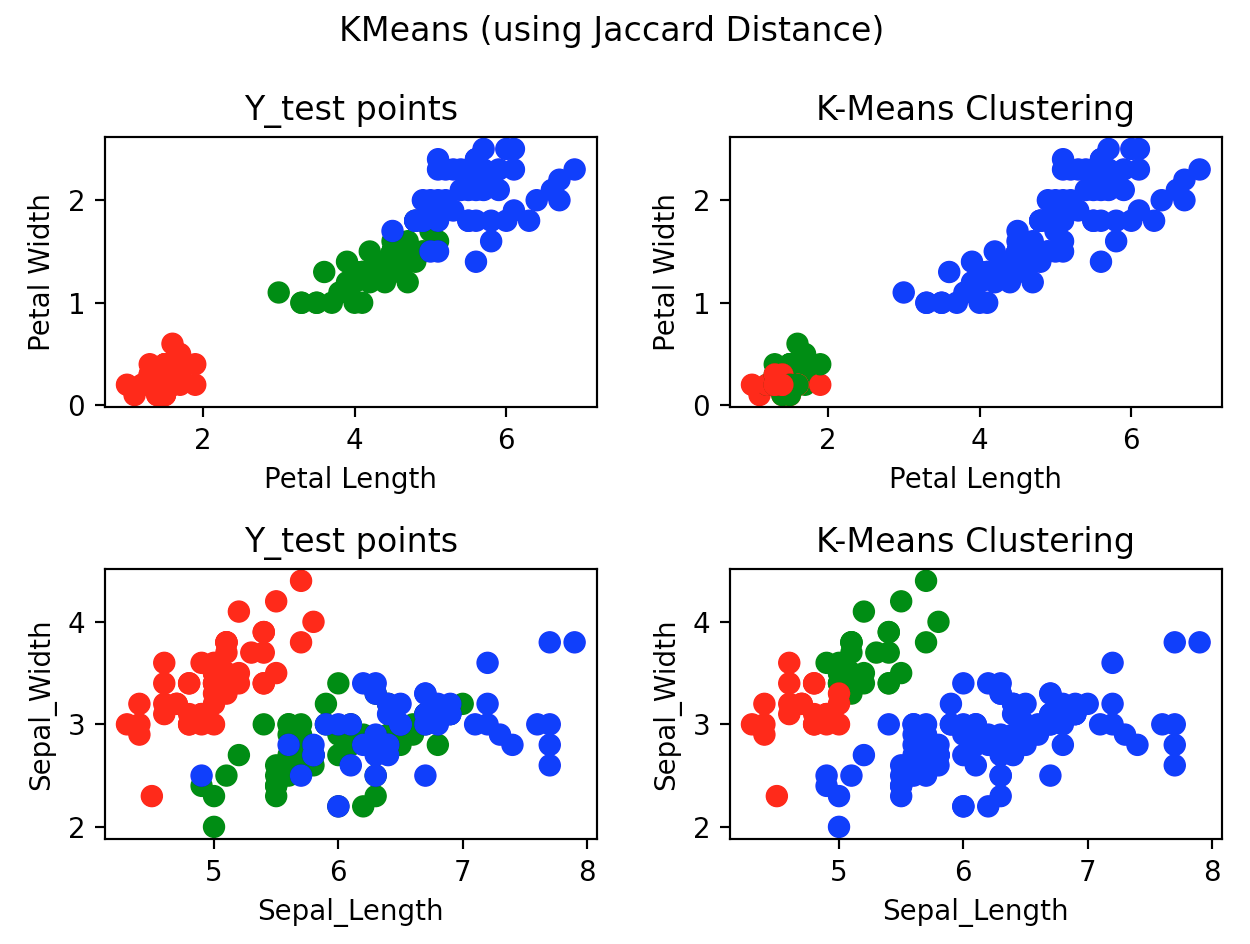
Plotting 2D graphs by taking 2 features at a time:

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated



1. The SSE value changes for every iteration as the centroids change during different runs. Most of the times, it is observed that cosine distance has the lowest SSE. One such output is shown below:

A clock mounted to the side

Description automatically generated

1. The accuracy changes for each individual run as the centroids changes each time. The accuracy one such run is as follows. As seen below, for this iteration, cosine distance seems to have the highest accuracy majority of the times

A picture containing clock, object, monitor, ball

Description automatically generated

1. The number of iterations for K – means to converge and the time taken for execution with each of the distance measures are shown as follows. Jaccard similarity seems to take the least amount of time and the least number of iterations to converge.

**A screen shot of a computer

Description automatically generated**

1. The above implementations ensured that the K-means clustering stopped when there was no further change in the centroids. Hence, for 4a, the implementation and the answer is the same as above. Outputs consistently changes as the centroids picked are random. For the below output, termination criteria is set to “*centroids*” (i.e, when centroids converge). Output from one of the runs is as follows:

**Text

Description automatically generated**

A clock mounted to the side

Description automatically generated

As K-means implementation progresses, it was noticed that SSE never increased. SSE continuously reduced or stayed the same as the previous iteration. Setting this as my termination criteria in the code, the below output was obtained. Once again, it is important to note that since the centroids change, the output obtained at each run changes.

Text

Description automatically generated

**A large clock mounted to the side

Description automatically generated**

Even when the “*max\_iters*” variable was set to a 100, the centroids converged much before that. One such output is as follows:

Text

Description automatically generated

A picture containing clock, object, ball, green

Description automatically generated

On an average, it was noticed that when the termination criteria is to check if SSE value increases in the next iteration or not, then K-means clustering algorithm converged much faster than the other two termination criteria.

Since the output varied so much due to the randomness of the centroids, I decided to run K-means using Euclidean distance, Cosine distance and Jaccard distance for 100 runs to get the average SSE, accuracy, time and number of iterations required to converge over the 100 executions.

Histogram plots are also plotted for the average values and the mean is indicated using the dotted line. The output is as follows:

Graphical user interface, application

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface, application

Description automatically generated

**Based on the mean values over 100 iterations, Jaccard has the lowest SSE and highest accuracy. Euclidean distance provides the lowest average for the number of iterations to converge (rounded to 1), and Jaccard has the lowest execution time.**

Chart, histogram, waterfall chart

Description automatically generated

Chart, histogram

Description automatically generated

Chart, bar chart, histogram

Description automatically generated

For a fixed centroid: (5, 3.4, 1.46, 0.25), (6.5, 2.95, 5.5, 1.99), (5.9, 2.8, 4.2, 1.3), the following output was obtained:

Text

Description automatically generated

Cosine distance provided the highest accuracy and lowest SSE. Jaccard had the lowest execution time. And both Euclidean and Jaccard had the lowest number of iterations.

**TASK 3:**

1. K-Means clustering fails to give efficient clustering results when we have outliers in the data. This is because the center for the clusters is taken by considering statistical mean value of all the datapoints and the outliers can change the overall mean of the datapoint in the cluster. Also, if the density spread of the datapoints across the data space is different and the data points follow non-convex shapes, then the standard K-means will fail.

K-Means can also have bad clustering results when cluster densities and cluster sizes differ. Hence, it has the possibility of creating more clusters.

Chart, scatter chart

Description automatically generated

Figure taken from: <https://www.analyticsvidhya.com/blog/2017/02/test-data-scientist-clustering/>

1. One of the solutions to the limitations of K-means is by putting several centroids in the dataset. Another solution is to consider centroids further away from each other. Consider a situation where K-means has to randomly pick three centers during the initialization process. If these three centers are picked very close to each other, then K means will do a bad job clustering the data points.

C3

C2

C1

However, this problem can be solved by ensuring that the randomly initialized centers are well separated and not grouped together, thus K-means clustering will not be misled. We can do this by picking center C1 and then scan all the data points and pick the data point that has the maximum distance from C1. We can call this data point as center, C2. Further, to pick the third center, take the longest average between the two centers C1 and C2, and call this as C3. These centers are further refined as K-means goes through a few iterations.

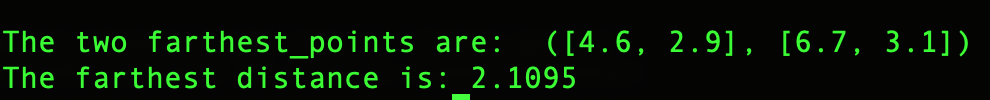
­­

C3

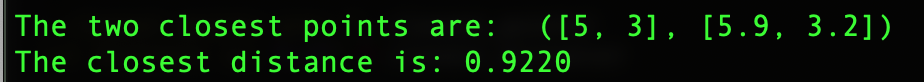
**TASK 4:**

The data points of the two clusters are considered and the Euclidean distance is used to compute the maximum and minimum distance between the cluster points are computed.

1. The farthest datapoints between the clusters are (4.6, 2.9) and (6.7, 3.1) and the distance between them is 2.1095

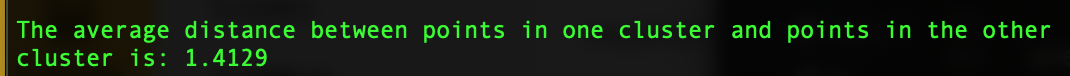


1. The closest datapoints between the clusters are (5,3) and (5.9, 3.2) and the distance between them is 0.9220



1. There are 28 pairs of points in the entire dataset. The average distance between all pairs of datapoints is: 0.9830. However, the average distance between the datapoints of the first cluster and the datapoints of the second cluster is: 1.4128





1. Min (closest) distance is sensitive to noise and outliers. Average distance and max distance are both less sensitive to noise and outliers compared to min (closest) distance. Max (farthest) distance is least susceptible to noise or outlier datapoints.

GitHub link:

Assignment\_05:

https://github.com/monicabernard/CAP-5610\_Machine-Learning.git