KMeans Clustering Algorithm

Abstract

Clustering is a type of Unsupervised Learning algorithm which is often used when we don't have a labeled data, as in we have no information which class does the subset of data belongs to. There are various clustering algorithms such as K-Median, Expectation maximization etc. The algorithm I have implemented is of K-Means for the assignment. K-Means is one of the popular clustering algorithms, it clusters the data into **k** number of clusters

Concept

The K-Means algorithm clusters the data into **K** number of clusters which can then be used to classes or labels for the unlabeled data. The basic concept is that is finds pattern by fixing the K mean Centroid and finding the nearest data points around the centroid. This algorithm can be achieved in 4 steps:

STEP 1 INITIALIZATION:

- This is done by initializing k values of centroid randomly at first
- Randomly pick K cluster centers, let's say there are
- $C = C_1, C_2, C_3, \ldots\ldots C_k$
- Where C is the set of randomly initialized cluster points

STEP 2 ASSIGNING INPUT WITH NEAREST DISTANCE:

- In this step, we assign each input value to the closest center
- This is achieved by calculating the **Euclidean Distance** (L2) between each input point and each centroid

$$\arg\min_{C_i \in C} dist(x, C_i)^2$$

- Where dist(.) is the Euclidean distance

$$\sqrt{(x - C_i)^2 + (y - C_i)^2}$$

STEP 3 FIND MEAN AND UPDATE CLUSTER:

- In this step we find the mean of all the clusters that had the minimum distance to the centroid and was classified within C_i in the previous step

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i$$

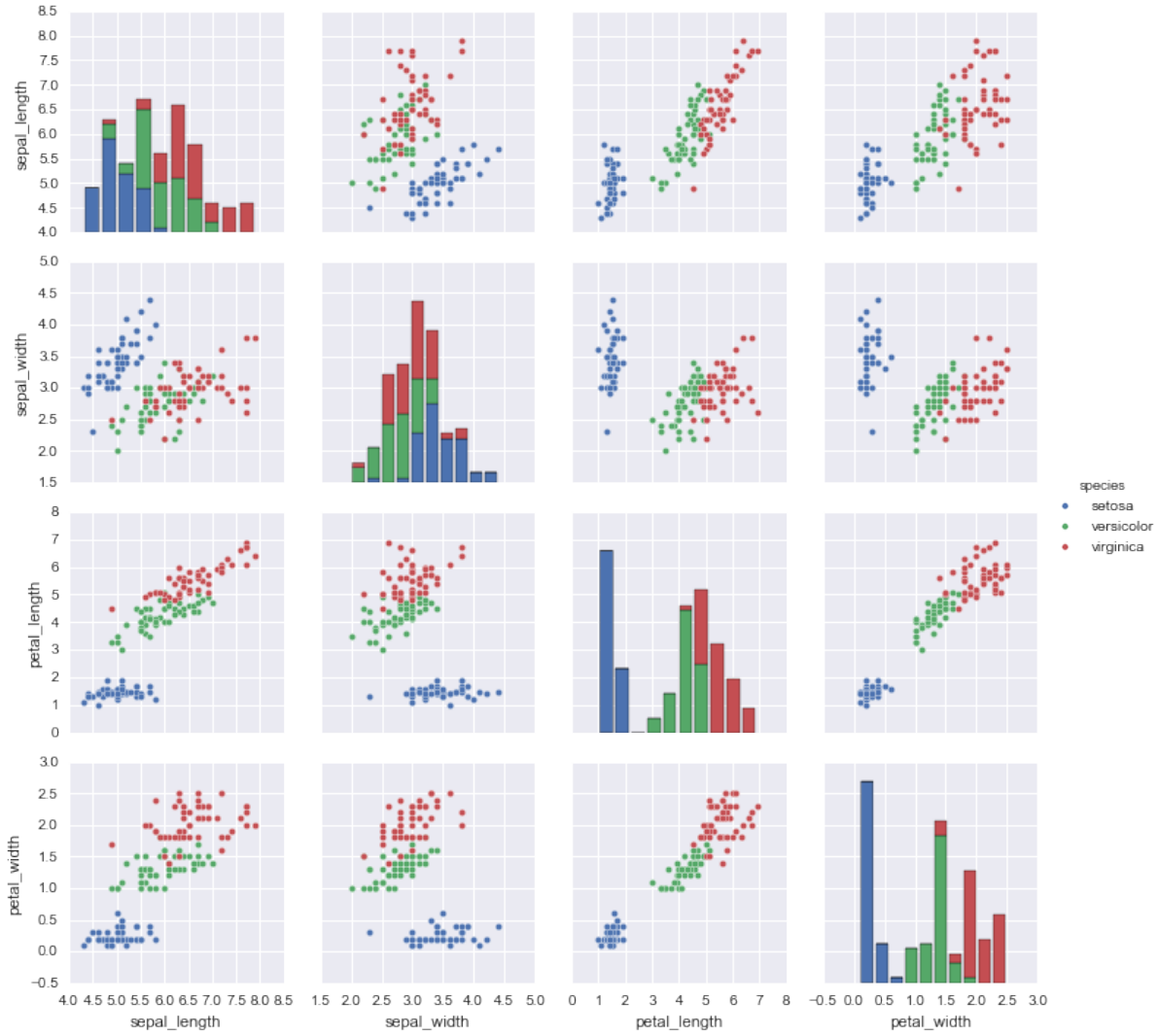
- After finding the mean we update the previous Centroid points to the mean of the centroids after a iteration

STEP 4 REPEAT STEP 2 & STEP 3:

- We repeat steps 2 and step 3 until there is no more change in the Centroid positions

OBERVATION & DATA INTERPRETATION

The feature matrix of Iris dataset was plotted against each other as seen below.



We can notice from here that petal_width and petal_length have clearly defined correlation compared to other feature vectors.

I have taken **petal_length** and **petal_width** for my code test but for clustering since we are not bound by the data correlation values and class labels, we have freedom to explore with any combination. The algorithm would perform same on any data, and tester is free to experiment with different combination.

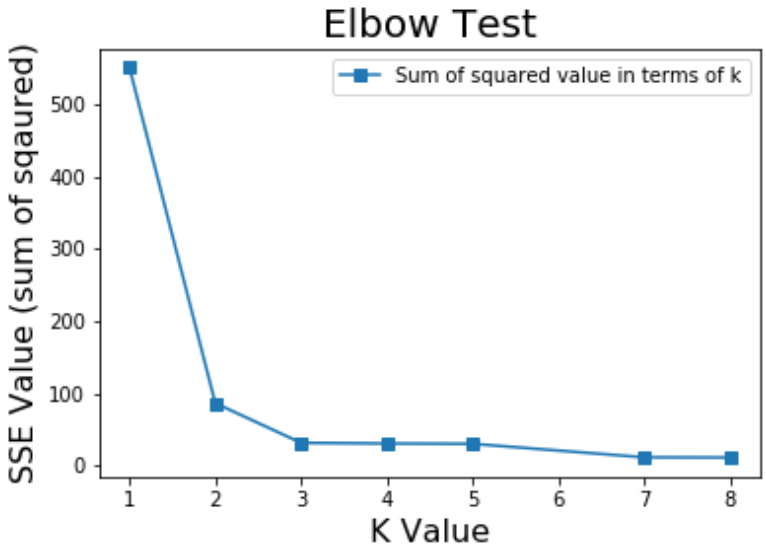
CHOOSING K VALUE

The most important part of K-Means algorithm is choosing the right K value as we have unlabeled data therefore we do not have the basic idea as to how many clusters are appropriate. Now there are various methods that are present and implemented to find the right K value. The method I chose to determine the K value is the **Elbow Test** method.

The Elbow test computes the Sum of Squared errors for some values of k say k = 1 , 2 , 3, 5, 7, 9 where sum of squared error is defined as sum of the squared distance between each member of the cluster and its centroid.

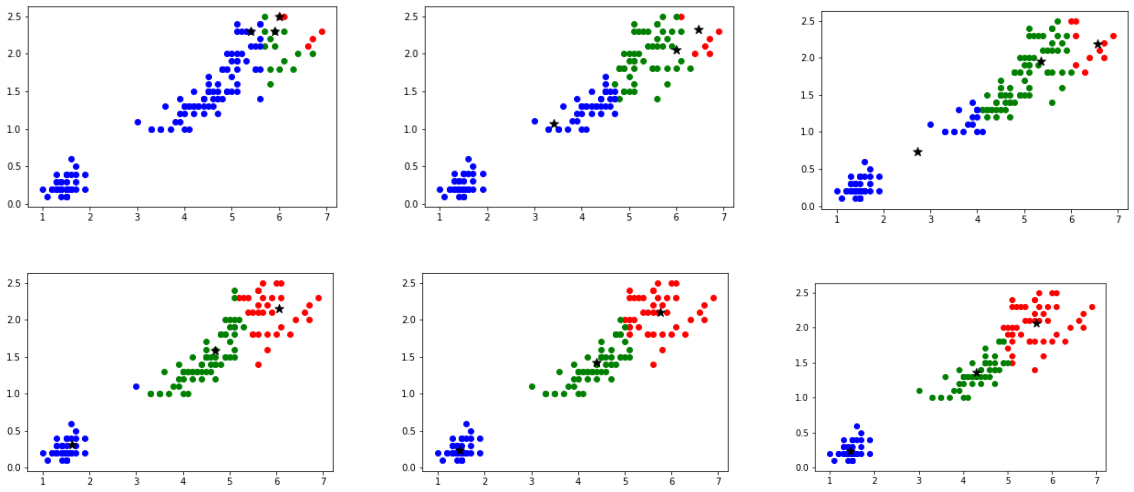
$$SSE = \sum_{i=1}^{K} \sum_{x \in c_i} dist(x, c_i)^2$$

I ran the test on my data with couple of K values and below is the graph result that I achieved

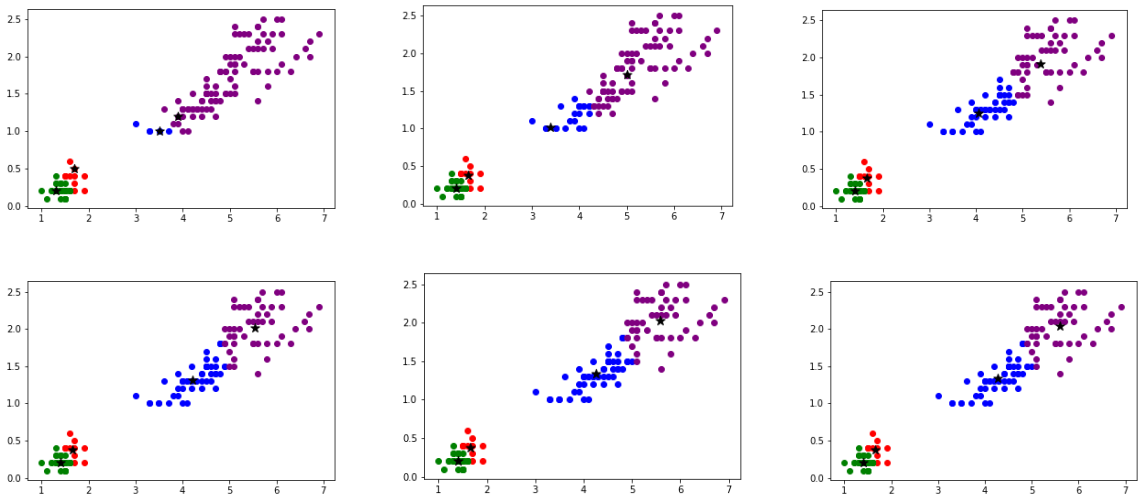


As we notice that as K increases the *error* values decreases this happens as there are more number of clusters therefore the errors should be smaller as the data points are tightly or overly fitted with centroid point. We take the K value where the distortion of error is abruptly decreased as we can see from the above graph any value from 2 to 4 therefore I decided to use **K=3** as my K value (which is exactly the right amount of class labels for our data)

Using K=3



Using K = 4



K_means(K_clusters,[elbow_test]) ->
This function takes in a K value to determine the number of clusters to form and optional Elbow test Boolean argument (False by default) which is set to true during Elbow test performance so only necessary output is printed out and this acts as the main function where K means clustering algorithm is performed.

Sum_of_squared_error(centroids, resultant_dataframe_table) ->
Calculates the sum of squared errors for Elbow test from the resultant final Centroid and assigned labels data table.

Update_centroid(centroids, resultant_dataframe_table) ->
This function calculates the mean value of the resultant squared distance calculation and updates the centroid.

Euclidean_distance(data_x_points, data_y_points, centroids) ->
This function calculates the Euclidean distance between the data points and the centroid and return the resultant data frame table

Plot_centers() ->
This function is used to visualize the performance of Elbow Test and give a visual run of K means working to find the centroid.

NOTE: I chose K value as **3** by performing the Elbow Test explained above