PROJECT 3 REPORT
SHREYAS MOHAN 1001669806

Project Description

- We use the Iris Dataset. This data consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). It has four features from each sample: length and width of sepals and petals.
- We use K Means algorithm to cluster the data

K means works through the following iterative process:

- Pick a value for k (the number of clusters to create).

  We use **Elbow Method** to determine the value of k and choose k as 3 as it is optimum. Also k=3, as we have **3 classes**.

- Initialize k 'centroids' (starting points) in your data

  We initialize k=3

- Creating clusters. Assign each point to the nearest centroid.

- Making clusters better. Move each centroid to the center of its cluster.

- Repeat the above steps until your centroids converge.

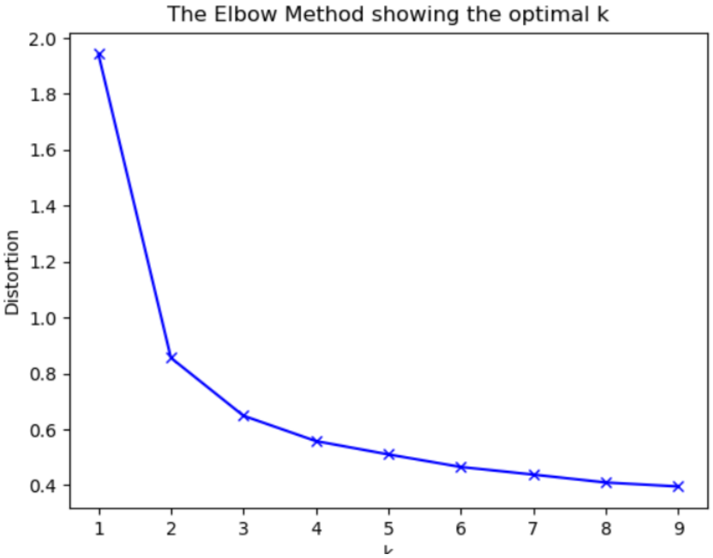
Structure of Code:

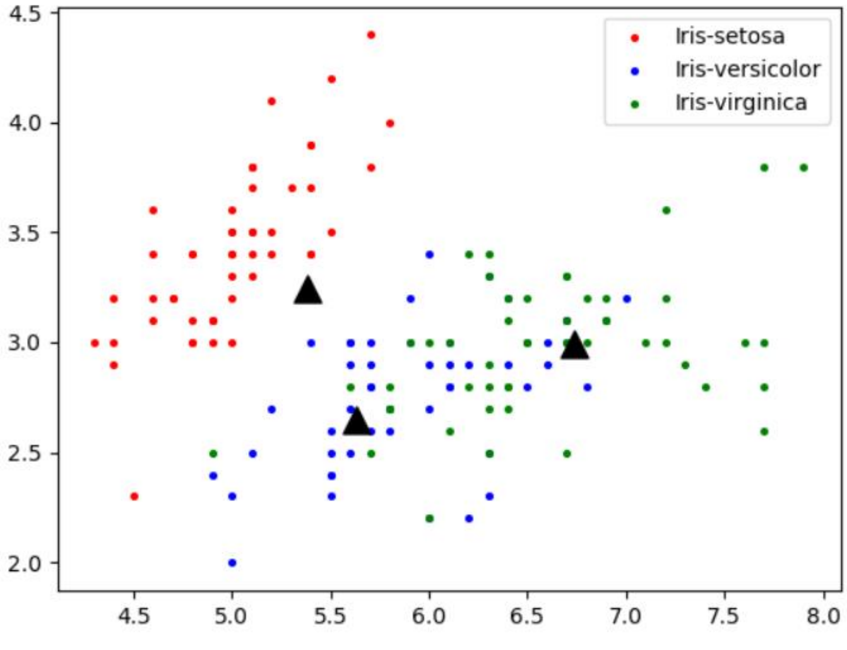
- First, the iris data file is copied to an iris csv file and headers 'SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth', and 'Species' are added to the data points for accessing the data points.
- Features of the iris are stored in a list while categories are stored in a separate list.
- We use elbow_method() to determine the value of k and as there are only three classes, we initialize k to 3.
- We find the centroids of the cluster using the mean and standard deviation formula which includes error.
- We remove the error by normalizing the distance between each centroid and data points and assign it to the closest centroid and repeating this until the error comes to zero.
- Output displays the number of data points that are assigned to an incorrect cluster and also displays the centroids.
- Plotting the data points and centroids in the graph and saving it in an external file 'KMeans.png'

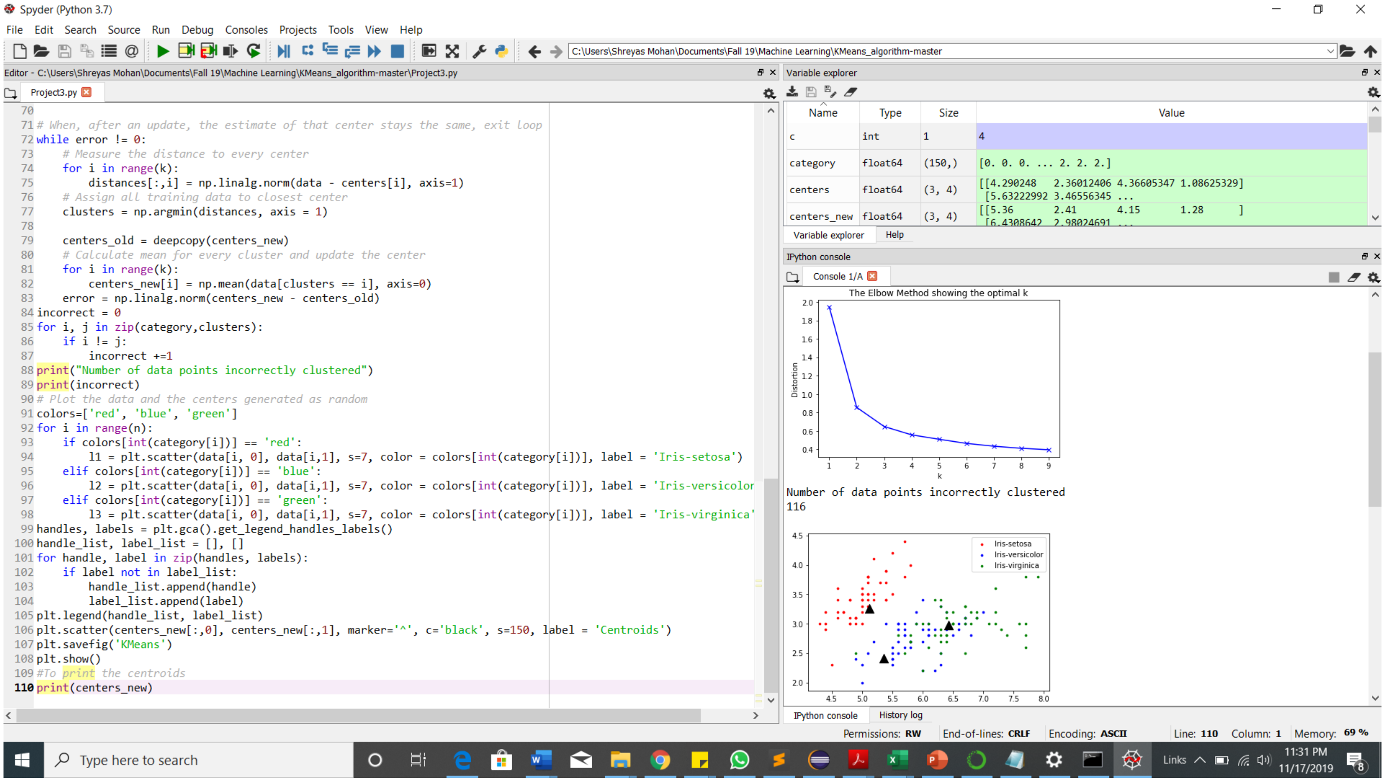
Screenshot of the output:

You can run on command line or any Python IDE.

Refer readme file for execution instructions.







```

# When, after an update, the estimate of that center stays the same, exit loop
while error != 0:
    # Measure the distance to every center
    for i in range(k):
        distances[:,i] = np.linalg.norm(data - centers[i], axis=1)
    # Assign all training data to closest center
    clusters = np.argmin(distances, axis = 1)

    centers_old = deepcopy(centers_new)
    # Calculate mean for every cluster and update the center
    for i in range(k):
        centers_new[i] = np.mean(data[clusters == i], axis=0)
    error = np.linalg.norm(centers_new - centers_old)
incorrect = 0
for i, j in zip(category,clusters):
    if i != j:
        incorrect +=1
print("Number of data points incorrectly clustered")
print(incorrect)
# Plot the data and the centers generated as random
colors=['red', 'blue', 'green']
for i in range(n):
    if colors[int(category[i])] == 'red':
        l1 = plt.scatter(data[i, 0], data[i,1], s=7, color = colors[int(category[i])], label = 'Iris-setosa')
    elif colors[int(category[i])] == 'blue':
        l2 = plt.scatter(data[i, 0], data[i,1], s=7, color = colors[int(category[i])], label = 'Iris-versicolor
    elif colors[int(category[i])] == 'green':
        l3 = plt.scatter(data[i, 0], data[i,1], s=7, color = colors[int(category[i])], label = 'Iris-virginica'
handles, labels = plt.gca().get_legend_handles_labels()
handle_list, label_list = [], []
for handle, label in zip(handles, labels):
    if label not in label_list:
        handle_list.append(handle)
        label_list.append(label)
plt.legend(handle_list, label_list)
plt.scatter(centers_new[:,0], centers_new[:,1], marker='^', c='black', s=150, label = 'Centroids')
plt.savefig('KMeans')
plt.show()
#To print the centroids
print(centers_new)
```

Name	Type	Size	Value
c	int	1	4
category	float64	(150,)	[0. 0. 0. ... 2. 2. 2.]
centers	float64	(3, 4)	[[4.290248 2.36012406 4.36605347 1.08625329] [5.63222992 3.46556345 ...
centers_new	float64	(3, 4)	[[5.36 2.41 4.15 1.28] [6.4308642 2.98024691 ...

```
Number of data points incorrectly clustered
116
```

Permissions: RW   End-of-lines: CRLF   Encoding: ASCII   Line: 110   Column: 1   Memory: 69 %