MACHINE LEARNING LAB - TUTORIAL 11

Juan Fernando Espinosa

303158

DATA IMPORTING

The data chosen for this tutorial is **Movielens 100k movie ratings**.

```
 import pandas as pd
import numpy as np
%matplotlib inline
import math
import matplotlib.pyplot as plt
from google.colab import files
from google.colab import drive
from sklearn import datasets
from pandas.plotting import scatter_matrix


drive.mount('/content/drive')
!ls "/content/drive/My Drive/Colab Notebooks/LAB/tutorial 11/iris1.csv"
iris = datasets.load_iris()

arrays = np.array([[0,0],
                   [0,1],
                   [-1,2],
                   [2,0],
                   [3,0],
                   [4,-1]])
arrays.dtype
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
'/content/drive/My Drive/Colab Notebooks/LAB/tutorial 11/iris1.csv'
dtype('int64')
```

```
X = iris.data
y = iris.target
```

EXERCISE 1: K-MEANS

```
# IDENTIFYING THE K CENTROIDS

Cluster1 = np.array(X[0])
ClusterA = []
ClusterB = []
ClusterC = []
distances = []
new_list = []
Cluester_new = 0
for i in range(len(X)):
  distance =0
  for j in range(len(X[i])):
    distance += np.power((X[i][j] - Cluster1[j]),2)
  distances.append(distance)
max_distance = np.amax(distances)
result = np.where(distances == max_distance)
distances.remove(max_distance)
second_max_distance = np.amax(distances)
result1 = np.where(distances == second_max_distance)
Cluster2 = X[result]
Cluster3 = X[result1]
Clusters = np.append([Cluster1], Cluster2,axis = 0)
Clusters = np.append(Clusters, Cluster3, axis = 0)
print('Initial Cluster',Clusters)
Distance_cluster = []

# INITIALIZATION OF THE K MEANS ALGORITHM
for _ in range(15):
  ClusterA = []
```

```
ClusterB = []
ClusterC = []
for i in range(len(Clusters)):
  partial_distance = []
  for k in range(len(X)):
    distance =0
    for j in range(len(X[i])):
      distance += np.power((X[k][j] - Clusters[i][j]),2)
    partial_distance.append(distance)
  Distance_cluster.append(partial_distance)
min_Values = np.argmin(Distance_cluster, axis=0)
for i in range(len(min_Values)):
  if min_Values[i] == 0:
    ClusterA.append(X[i])
  elif min_Values[i] == 1:
    ClusterB.append(X[i])
  else:
    ClusterC.append(X[i])
ClusterA = np.array(ClusterA)
ClusterB = np.array(ClusterB)
ClusterC = np.array(ClusterC)

C1_mean = np.mean(ClusterA, axis=0)
C2_mean = np.mean(ClusterB, axis = 0)
C3_mean = np.mean(ClusterC, axis=0)

Cluster_new = np.append([C1_mean], [C2_mean], axis =0)
Cluster_new = np.append(Cluster_new, [C3_mean], axis = 0)
if (1/len(Clusters))*(np.sum((Clusters - Cluster_new)**2)) < 0.1:
  print('ClusterA :',ClusterA)
  print('ClusterB :',ClusterB)
  print('ClusterC :',ClusterC)
  break
Clusters = Cluster_new
```

```
Initial Cluster [[5.1 3.5 1.4 0.2]
 [7.7 2.6 6.9 2.3]
 [7.7 3.8 6.7 2.2]]
ClusterA : [[5.1 3.5 1.4 0.2]
 [5.  3.6 1.4 0.2]
 [5.1 3.5 1.4 0.3]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.9 3.6 1.4 0.1]
 [5.  3.5 1.3 0.3]]
ClusterB : [[7.7 2.6 6.9 2.3]]
ClusterC : [[4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]
 [5.7 2.8 4.5 1.3]
 [6.3 3.3 4.7 1.6]
 [4.9 2.4 3.3 1. ]
 [6.6 2.9 4.6 1.3]
 [5.2 2.7 3.9 1.4]
 [5.  2.  3.5 1. ]
 [5.9 3.  4.2 1.5]
 [6.  2.2 4.  1. ]
 [6.1 2.9 4.7 1.4]
 [5.6 2.9 3.6 1.3]
 [6.7 3.1 4.4 1.4]
 [5.6 3.  4.5 1.5]
 [5.8 2.7 4.1 1. ]
 [6.2 2.2 4.5 1.5]
 [5.6 2.5 3.9 1.1]
 [5.9 3.2 4.8 1.8]
 [6.1 2.8 4.  1.3]
 [6.3 2.5 4.9 1.5]
 [6.1 2.8 4.7 1.2]
 [6.4 2.9 4.3 1.3]
 [6.6 3.  4.4 1.4]
 [6.8 2.8 4.8 1.4]
 [6.7 3.  5.  1.7]
 [6.  2.9 4.5 1.5]
 [5.7 2.6 3.5 1. ]
 [5.5 2.4 3.8 1.1]
 [5.5 2.4 3.7 1. ]
 [5.8 2.7 3.9 1.2]
 [6.  2.7 5.1 1.6]
 [5.4 3.  4.5 1.5]
 [6.  3.4 4.5 1.6]
 [6.7 3.1 4.7 1.5]
```

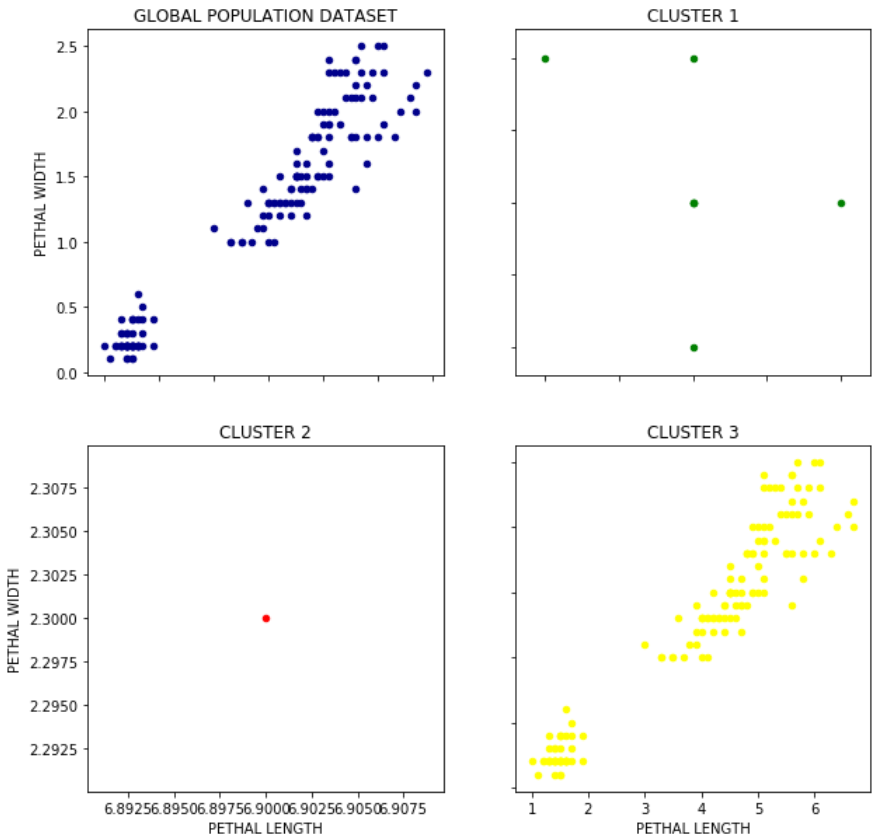
```
      [6.7 3.1 4.7 1.5]
      [6.3 2.3 4.4 1.3]
      [5.6 3.  4.1 1.3]
      [5.5 2.5 4.  1.3]
      [5.5 2.6 4.4 1.2]
      [6.1 3.  4.6 1.4]
      [5.8 2.6 4.  1.2]
      [5.  2.3 3.3 1. ]
      [5.6 2.7 4.2 1.3]
      [5.7 3.  4.2 1.2]
      [5.7 2.9 4.2 1.3]
      [6.2 2.9 4.3 1.3]
      [5.1 2.5 3.  1.1]
      [5.7 2.8 4.1 1.3]
      [6.3 3.3 6.  2.5]
      [5.8 2.7 5.1 1.9]
      [7.1 3.  5.9 2.1]
      [6.3 2.9 5.6 1.8]
      [6.5 3.  5.8 2.2]
      [7.6 3.  6.6 2.1]
      [4.9 2.5 4.5 1.7]
      [7.3 2.9 6.3 1.8]
      [6.7 2.5 5.8 1.8]
      [7.2 3.6 6.1 2.5]
      [6.5 3.2 5.1 2. ]
      [6.4 2.7 5.3 1.9]
      [6.8 3.  5.5 2.1]
      [5.7 2.5 5.  2. ]
      [5.8 2.8 5.1 2.4]
      [6.4 3.2 5.3 2.3]
      [6.5 3.  5.5 1.8]
      [7.7 3.8 6.7 2.2]
      [6.  2.2 5.  1.5]
      [6.9 3.2 5.7 2.3]
      [5.6 2.8 4.9 2. ]
      [7.7 2.8 6.7 2. ]
      [6.3 2.7 4.9 1.8]
      [6.7 3.3 5.7 2.1]
      [7.2 3.2 6.  1.8]
      [6.2 2.8 4.8 1.8]
      [6.1 3.  4.9 1.8]
      [6.4 2.8 5.6 2.1]
      [7.2 3.  5.8 1.6]
      [7.4 2.8 6.1 1.9]
      [7.9 3.8 6.4 2. ]
      [6.4 2.8 5.6 2.2]
      [6.3 2.8 5.1 1.5]
      [6.1 2.6 5.6 1.4]
      [7.7 3.  6.1 2.3]
      [6.3 3.4 5.6 2.4]
      [6.4 3.1 5.5 1.8]
      [6.  3.  4.8 1.8]
      [6.9 3.1 5.4 2.1]
      [6.7 3.1 5.6 2.4]
      [6.9 3.1 5.1 2.3]
      [5.8 2.7 5.1 1.9]
      [6.8 3.2 5.9 2.3]
      [6.7 3.3 5.7 2.5]
      [6.7 3.  5.2 2.3]
      [6.3 2.5 5.  1.9]
      [6.5 3.  5.2 2. ]
      [6.2 3.4 5.4 2.3]
      [5.9 3.  5.1 1.8]]


# TRANSFORM EACH CLUSTER INTO DATAFRAMES
Cluster_A = pd.DataFrame({'Column1': ClusterA[:, 0], 'Column2': ClusterA[:, 1], 'Column3': ClusterA[:, 2], 'Column4': ClusterA[:, 3]
Cluster_B = pd.DataFrame({'Column1': ClusterB[:, 0], 'Column2': ClusterB[:, 1], 'Column3': ClusterB[:, 2], 'Column4': ClusterB[:, 3]
Cluster_C = pd.DataFrame({'Column1': ClusterC[:, 0], 'Column2': ClusterC[:, 1], 'Column3': ClusterC[:, 2], 'Column4': ClusterC[:, 3]
X1 = pd.DataFrame({'Column1': X[:, 0], 'Column2': X[:, 1], 'Column3': X[:, 2], 'Column4': X[:, 3]})


# PLOTTING THE DISTRIBUTION OF THE GLOBAL DATASET AND FOR EACH CLUSTER.
fig, axs = plt.subplots(2, 2,figsize=(10,10))
X1.plot.scatter(x='Column3',y='Column4',c='DarkBlue', ax=axs[0, 0])
axs[0, 0].set_title('GLOBAL POPULATION DATASET')
Cluster_A.plot.scatter(x='Column3',
                       y='Column4',
                       c='green', ax=axs[0, 1])
axs[0, 1].set_title('CLUSTER 1')
Cluster_B.plot.scatter(x='Column3',
                       y='Column4',
                       c='red', ax=axs[1, 0])
axs[1, 0].set_title('CLUSTER 2')
Cluster_C.plot.scatter(x='Column3',
                       y='Column4',
                       c='yellow', ax=axs[1, 1])
axs[1, 1].set_title('CLUSTER 3')
for ax in axs.flat:
```

```
    ax.set(xlabel='PETHAL LENGTH', ylabel='PETHAL WIDTH')

for ax in axs.flat:
    ax.label_outer()
```



CONCLUSIONS:

> The classification of each flower is not executed correctly since only one fower belongs to cluster 2 and a few belongs to cluster 1.

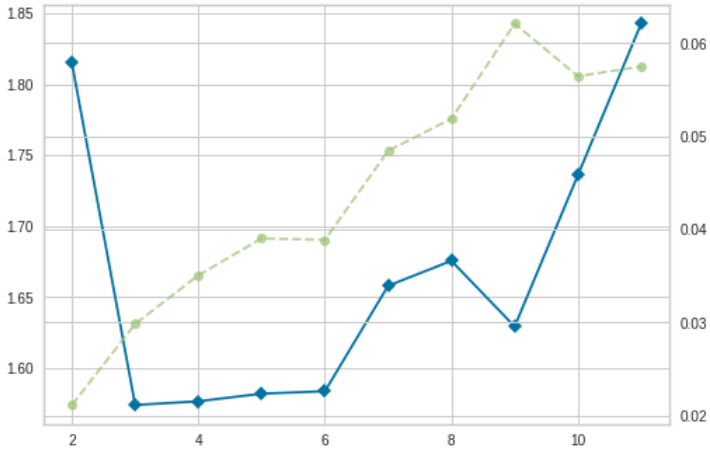
> The methodology used to select the third cluster is not the ideal one.

> The criterion chosen to define k = 3 is based on the Elbow method. This method estimates that in the point of inflection of the dataset remains the model fits best at that point. Below it is printed the ideal K by using an easy build-in function since it wasn't a requirement to plot it.

```
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer
from sklearn.datasets import make_blobs
model = KMeans()
visualizer = KElbowVisualizer(model, k=(2,12))

visualizer.fit(X)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: The sklearn.metrics.classification modu
  warnings.warn(message, FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/base.py:197: FutureWarning: From version 0.24, get_params will raise an Attribut
  FutureWarning)
KElbowVisualizer(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x7fd057d98828>,
                 k=None, metric=None, model=None, timings=True)
```



DIFFERENCE BETWEEN K-NN AND K-MEANS

> K-NN is a supervised algorithm used for classification. The labels of the data **are known** which are used to train the model and produce some learning to classify the unseen data. The data is discrete.

> K-Means is an unsupervised algorithm used for clustering. The data is **not labeled** therefore the algorithm relies on the independent features to make predictions for unseen data.

BIBIOGRAPHY:

> https://www.scikit-yb.org/en/latest/api/cluster/elbow.html

> https://www.quora.com/How-is-the-k-nearest-neighbor-algorithm-different-from-k-means-clustering

/