Classification, Prediction and Clustering

Classification and Prediction:

The method used in the data analysis of the admitted students is called Decision Tree. This method(Its algorithm) allows the efficient use of resource by classifying data and learning about it to make predictions about future events that may occur. It involves the use of only features(Independent features) that are going to be used to predict an outcome(Target feature). The features are first checked to confirm that each feature is relly necessary and needed to predict an outcome. For example, a customer buying a product does not guarantee the weather condition of a given geographical area.

In this analysis, Independent features are:

1. Sex
2. Age
3. English score
4. Total score;

which are then analyzed both collectively and individual both to predict a department(Column ‘Dept’) which a given student might be admitted to. For analysis, the data is split into two sets: Train data which is used to train the model for prediction purposes; Test data which is used for actual classification and plays an important and final role in the trainning of the model.

### Program:

import pandas as pd

from sklearn.tree import DecisionTreeClassifier # for the construction of the decision tree

from sklearn.model\_selection import train\_test\_split # to split data into two

from sklearn.tree import export\_graphviz # for data visualization

from six import StringIO # for initialization of byte objects that make up the visualized data

from IPython.display import Image # for saving byte object as image

import pydotplus # for construction of byte objects

from sklearn import metrics #for accuracy calculation

#feature columns are data columns received by the program to predict a target feature

feature\_cols = ['SerialNo', 'Sex', 'Age', 'StateOfOrigin', 'TotalScore']

#open the file containing the data

file = pd.read\_csv('striped\_data.csv')

#assign features needed(X) to predict an outcome(Y)

X = file[feature\_cols]

#target column

Y = file.Dept

#split the dataset into two parts: X\_train are column features for training the program, y\_train is the corresponding target variable. X\_test is used to test the accuracy of the program in terms of predicting data. It is splitted in the ratio of 3:7 [test:train]. 70% of the data is used to train the program while the other 30% tests the program’s accuracy

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.3, random\_state=1)

#a DecisionTreeClassifier instance is created for creating a decision tree

clf = DecisionTreeClassifier(criterion="entropy", max\_depth=6)

#the decision tree is plotted and commited to memory using the train dataset

clf = clf.fit(X\_train,y\_train)

#the program is asked to predict a set of data

y\_pred = clf.predict(X\_test)

#the prediction output is checked against the actual target variable to test its accuracy.

print("Accuracy:", metrics.accuracy\_score(y\_test, y\_pred))

#the plotted decision tree is converted to byte object and then is used to create a picture

dot\_data = StringIO()

export\_graphviz(clf, out\_file=dot\_data,

filled=True, rounded=True,

special\_characters=True,feature\_names = feature\_cols,class\_names=['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19','20','21','22','23','24','25','26','27', '28'])

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

graph.write\_png('done5.png')

Image(graph.create\_png())

The program creates a png file containing a decision tree of how the program classified the data into different data sets. The program classifies data into the target set by selecting the feature which neatly classifies the data into two distinctive parts. This is done frequently as the data sets are then split based on the above until the specified tree length is reached(if given or else it continues until all data has been classified)

Clustering:

K-Means clustering is used since it the feature the data is to be clustered based on is not specified i.e the program clusters the data based on similarities in data.

The program(It) first maps the two variables(both are iterables) to be used for both axis on the clustered data as x and y values using the *zip* function. It then selects 6 points (as specified by th user to be used as number of clusters) at randon by selecting random x and y points within the range of 0 and the largest value from the variable used for the x-axis.

These 6 points are then fixed on a graph. The position of the points will be adjusted and the points are used as centroids of each cluster. On the same graph, the two mapped variables are used as x and y co-ordinates to plot points for each item in the data set.

The kmeans module is then involved which classifies the data based on features that are related to each other into 6 different clusters. The 6 random points to be used as centroids are then adjusted programmatically to be positioned in the center of each cluster.

The number of clusters has already been selected by the user but there are cases whereby the data would be clustered better if a different number of clusters were used instead. The program then analyses the already clustered data and draws a graph showing what number of clusters would be suitable for the data. The initial number of clusters is then changed by the user.

**Program:**

import numpy as np #for numbers processing and manipulation

import pandas as pd

from matplotlib import pyplot as plt

from sklearn.cluster import Kmeans

from scipy.spatial.distance import cdist

#variable containing name of file

train = 'striped\_data.csv'

#read the data from the file

data = pd.read\_csv(train)

#to confirm content of data

shape = (data.shape)

print(f"Number of items: {shape[0]}\nShape: {shape[1]}")

#extract values from the data as a list to be used for both axis in the clustered graph: Total score and Department.

f1 = data['TotalScore'].values

f2 = data['Dept'].values

#map each value in both lists as corresponding x and y values

X = np.array(list(zip(f1, f2)))

#select number of clusters

k = 6

#select random points for x and y axis for 6 random centroid points

C\_x = np.random.randint(0, np.max(X)-20, size=k)

C\_y = np.random.randint(0, np.max(X)-20, size=k)

#map 6 points as x and y points

C = np.array(list(zip(C\_x, C\_y)), dtype=np.float32)

#display both points on a graph

plt.scatter(f1, f2, c='#050505', s=20)

#place random centroids as generated above on the same graph

plt.scatter(C\_x, C\_y, marker='\*', s=200, c='g')

#display label on the x axis

plt.xlabel("Total Score feature")

#display label on the y axis

plt.ylabel("Dept feature")

#display label on the title

plt.title("Output data")

#create a kmeans instance

kmeans = Kmeans(n\_clusters=k)

#use kmeans to rearrange the data on the graph

kmeans = kmeans.fit(X)

#put the data in clusters

labels = kmeans.predict(X)

#create class instance for clusters

centroids = kmeans.cluster\_centers\_

#specify colors to be used for data in each cluster

colors = ['r', 'g', 'b', 'y', 'c', 'm', 'o', 'w']

#create an object representation of the image of the graph

fig2 = plt.figure()

#add centroids to graph

kx = fig2.add\_subplot(111)

for i in range(k):

points = np.array([X[j] for j in range(len(X)) if labels[j] == i])

kx.scatter(points[:, 0], points[:, 1], s=20, cmap='rainbow')

kx.scatter(centroids[:, 0], centroids[:, 1], marker='\*', s=200, c='#050505')

#display the clustered graph with centroids included

print("Final centroids")

print(centroids)

plt.xlabel("Total score feature")

plt.ylabel("Dept feature")

plt.title('Number of clusters={}'.format(k))

#to predect numbers of clusters to present accurate data using the Elbow method

K\_range = range(1, 10)

distortions = []

for i in K\_range:

kmeanModel = KMeans(n\_clusters=i)

kmeanModel.fit(X)

distortions.append(sum(np.min(cdist(X, kmeanModel.cluster\_centers\_, 'euclidean'), axis=1)) / X.shape[0])

fig1 = plt.figure()

ex = fig1.add\_subplot(111)

ex.plot(K\_range, distortions, 'b\*-')

plt.grid(True)

plt.ylim([0, 45])

plt.xlabel("Number of clusters")

plt.ylabel("Average distortion")

plt.title("Selecting K with the Elbow Method")