Q. Classification
Write a Python program build Decision Tree Classifier using
Scikit-learn
package for diabetes data set (download database from
https://www.kaggle.com/uciml/pima-indians-diabetes-database)#import pandas library
***Soluton**:*

```
import pandas as pd

#loading dataset
df=pd.read_csv("/content/drive/My Drive/Colab Notebooks/diabetes_dataset.csv")
df.head()

#feature variables
x=df.drop(['Outcome'], axis=1)
x

#target variable
y=df.Outcome
y

from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1)


# Create Decision Tree classifer object
model = DecisionTreeClassifier()

# Train Decision Tree Classifer
model = model.fit(x_train,y_train)

#Predict the response for test dataset
y_pred = model.predict(x_test)

#Evaluation using Accuracy score
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
print("Accuracy:",metrics.accuracy_score(y_test, y_pred)*100)

#Evaluation using Confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)

print("Accuracy:",((82+27)/154))

#Evaluation using Classification report
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))

#checking prediction value
model.predict([[6,148,72,35,0,33.6,0.627,50]])

#Import modules for Visualizing Decision trees
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus

features=x.columns
features

dot_data = StringIO()
export_graphviz(model, out_file=dot_data,filled=True,
rounded=True,special_characters=True,feature_names = features,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes_set.png')
Image(graph.create_png())

# Create Decision Tree classifer object
model = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
model = model.fit(x_train,y_train)

#Predict the response for test dataset
y_pred = model.predict(x_test)

# Model Accuracy
print("Accuracy:",metrics.accuracy_score(y_test, y_pred)*100)


#Better Decision Tree Visualisation
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(model, out_file=dot_data,filled=True,
rounded=True,special_characters=True, feature_names = features,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes_set.png')
Image(graph.create_png())
```

Q. Consider following dataset
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny'
,'Sunny','Rainy','Sunny','Overcast','Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild
','Mi
ld','Mild','Hot','Mild']
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Ye
s','Y
es','No']. Use Naïve Bayes algorithm to predict[0:Overcast,
2:Mild]
tuple belongs to which class whether to play the sports or not.

Solution:

```
# Assigning features and label variables
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']

play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']
#predicting to play according to weather
```

```
# Import LabelEncoder
from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
wheather_encoded=le.fit_transform(weather)
print (wheather_encoded)
```

```
# Converting string labels into numbers
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)
print ("Temp:",temp_encoded)
print ("Play:",label)
```

```
#Combinig weather and temp into single listof tuples
features=list(zip(wheather_encoded,temp_encoded))
print (features)
```

```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
```

```
model = GaussianNB()

# Train the model using the training sets
model.fit(features,label)
```

```
#Predict Output
predicted= model.predict([[2,1]]) # 0:Overcast, 2:Mild
print("Predicted Value:", predicted)
```

Q.Write a python program to implement multiple Linear Regression model for a car dataset.
Dataset can be downloaded from:
[https://www.w3schools.com/python/python_ml_multiple_regressi on.asp](https://www.w3schools.com/python/python_ml_multiple_regression.asp)
Solution:

```
# Multiple Regression
# Multiple regression is like linear regression, but with more than one independent value,
# meaning that we try to predict a value based on two or more variables.
import pandas
from sklearn import linear_model
# Car, Model, Volume, Weight, CO2
df = pandas.read_csv("../input/linear-regression-dataset/cars.csv")
# print(df) # 0, Toyota, Aygo, 1000, 790, 99
X = df[['Weight', 'Volume']]
y = df['CO2']
regr = linear_model.LinearRegression()
# Tip: It is common to name the list of independent values with a upper case X,
# and the list of dependent values with a lower case y.
regr.fit(X, y)
# predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300ccm:
predictedCO2 = regr.predict([[2300, 1300]])
print("predictedCO2 [weight=2300kg, volume=1300ccm]:")
print(predictedCO2) # [107.2087328]

# Coefficient
# The coefficient is a factor that describes the relationship with an unknown variable.
print("Coefficient [weight, volume]")
print(regr.coef_) # [0.00755095 0.00780526]
df
predictedCO2 = regr.predict([[3300, 1300]])
print("predictedCO2 [weight=3300kg, volume=1300ccm]")
print(predictedCO2)
```

Q.Write a python program to implement k-means algorithm to build prediction model
(Use Credit Card Dataset CC GENERAL.csv Download from kaggle.com)
Solution:

```
# Credit Card Cluster Problem with K-Means

# Importing Preprocessing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Importing Datasets
dataset = pd.read_csv('../input/CC GENERAL.csv')
X = dataset.iloc[:, 1:].values


# Dataset Contains Multiple Missing values
# Replacing Missing Value by Most Repeated/Frequent Number in that column
# Use Imputer with strategy 'most_frequent'
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values="NaN", strategy="most_frequent", axis=0)
imputer = imputer.fit(X)
X = imputer.transform(X)


# Applying Feature Scalling with StandardScaler
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X = sc_X.fit_transform(X)
# For Finding Optimal Number of Cluster use Elbow Method
from sklearn.cluster import KMeans
wcss = []
for i in range(1,18):
   kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state
=0).fit(X)
   wcss.append(kmeans.inertia_)

plt.plot(range(1,18), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Cluster')
plt.ylabel('WCSS')
plt.show()
```

```
# Apply K-Means Again With Optimal Number of Cluster that we got from Elbow method i.e. 8
kmeans = KMeans(n_clusters=8, init='k-means++', max_iter=300, n_init=10, random_state=
0)
y_kmeans = kmeans.fit_predict(X)
```

```
# Finally Append new Column i.e Cluster to Actual Dataset
dataset['Cluster'] = y_kmeans
dataset.head()
```

Q.Write a python program to implement hierarchical clustering algorithm.

(Download Wholesale customers data dataset from github.com).

Solution:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
# y = dataset.iloc[:, 3].values
# Splitting the dataset into the Training set and Test set
"""from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
0)"""
# Feature Scaling
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""
# Using the dendrogram to find the optimal number of clusters
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
# Fitting Hierarchical Clustering to the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
```

```
# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```