UNIT 16 MACHINE LEARNING – PROGRAMMING USING PYTHON

16.0 INTRODUCTION

In this unit we will see the implementation of various machine learning algorithms, learned in this course. To understand the codes you need to have understanding of the respective Machine learning algorithms along with that understanding of Python programming is must. The codes are readily using various libraries of Python programming language viz. Scikit Learn, Matplotlib, numpy etc., you can execute these codes through any of the Python programming tools. Most of the machines learning algorithms, you learned in this course, are implemented here, just try to execute them and analyse the results.

16.1 OBJECTIVES

After going through this unit, you should be able to:

- Understand the implementation aspect of various machine learning algorithms

16.2 CLASSIFICATION ALGORITHMS

The starting units of this course primarily focused on the various classification algorithms viz. Naïve Bayes classifiers, K-Nearest Neighbour (K-NN), Decision Trees, Logistic Regression and Support Vector Machines.The theoretical aspects of

the same is already discussed in the respective units, now we will see the implementation part of the mentioned classifiers, in Python programming language.

16.2.1 NAIVE BAYES

It is a method of classification that is founded on Bayes' Theorem and makes the assumption that predictors are free to act independently of one another. A Naive Bayes classifier, to put it in layman's words, makes the assumption that the existence of one particular characteristic in a class is unrelated to the presence of any other feature.

We have already discussed this classifier in detail in Block 3 Unit 10 of this course, you may refer to Block 3 Unit 10 to understand the concept.

The following procedures need to be carried out in order to classify data using the Naive Bayes method.

- In the first step, we will begin by importing the dataset as well as any necessary dependencies...
- The second step is to get the prior probability of each class using the formula P(y).
- The Third Step is to Determine the likelihood of each characteristic using the table you just created...
- Final and the Fourth Step is to Calculate the Posterior Probability for each class by applying the Naive Bayesian equation.

The screenshot of the executed code is given below

Implementation code in Python

```
# load the iris dataset
from sklearn.datasets import load_iris
iris = load_iris()
# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target
# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,  test_size=0.4, random_state=1)
# training the model on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
# making predictions on the testing set
y_pred = gnb.predict(X_test)
# comparing actual response values (y_test) with predicted response  values (y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):",
metrics.accuracy_score(y_test, y_pred)*100)
```

```
Gaussian Naive Bayes model accuracy(in %): 95.0
```

OUTPUT :

Gaussian Naive Bayes model accuracy(in %): 95.0

16.2.2 K-Nearest Neighbour (K-NN)

You have already discussed this classifier in detail in Block 3 Unit 10 of this course, you may refer to Block 3 Unit 10 to understand the concept.

We learned that Suppose the value of K is 3. The KNN algorithm starts by calculating the distance of point X from all the points. It then finds the 3 nearest points with least distance to point X

In the example shown below following steps are performed:

• In Step 1, the scikit-learn package is used to import the k-nearest neighbour algorithm.
• Step 2. is to create the feature variables and the target variables.
• Step 3. Separate the data into the test data and the training data.
• Step 4.Generate a k-NN model using neighbours value.
• Step 5. Train the model using the data or adjust the model based on the data.
• Proceed to Step 6, which is to make a forecast.

Now, in this section, we will see how Python's Scikit-Learn library can be used to implement the KNN algorithm

Implementation code in Python

The screenshot of the executed code is given below

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
# Loading data
irisData = load_iris()
# Create feature and target arrays
X = irisData.data
y = irisData.target
# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size = 0.2, random_state=42)
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
# Predict on dataset which model has not seen before
print(knn.predict(X_test))
```

```
[1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
```

16.2.3 Desicion Tree Implementation

A decision tree is a type of supervised machine learning algorithm that may be used for both regression and classification tasks. It is one of the most popular and widely used machine learning techniques.

In this case, the decision tree method creates a node for each attribute present in the dataset, with the attribute that is considered to be the most significant being placed at the top of the tree. When we first get started, we will think of the entire training set as the root. There must be a categorical breakdown of the feature values.

Before beginning to develop the model, the values are discretized in order to determine whether or not they are continuous. A recursive process distributes records according to the attribute values of each record. A statistical method is utilised in order to determine which qualities should be placed at the tree's root and which should be placed at internal nodes.

You have already discussed this classifier in detail in Block 3 Unit 10 of this course, you may refer to Block 3 Unit 10 to understand the concept.

Implementation code in Python

The screenshot of the executed code is given below

```
# Importing the required packages
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
# Function importing Dataset
def importdata():
 balance_data = pd.read_csv( 'https://archive.ics.uci.edu/ml/machine-learning-'+
'databases/balance-scale/balance-scale.data', sep= ',', header = None)
# Printing the dataset shape
 print ("Dataset Length: ", len(balance_data))
 print ("Dataset Shape: ", balance_data.shape)
# Printing the dataset obseravtions
 print ("Dataset: ",balance_data.head())
 return balance_data
# Function to split the dataset
def splitdataset(balance_data):
```

```
# Function to split the dataset
def splitdataset(balance_data):
# Separating the target variable
 X = balance_data.values[:, 1:5]
 Y = balance_data.values[:, 0]
# Splitting the dataset into train and test
 X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.3, random_state = 100)
 return X, Y, X_train, X_test, y_train, y_test
# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):
# Creating the classifier object
 clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100,max_depth=3, min_samples_leaf=5)
# Performing training
 clf_gini.fit(X_train, y_train)
 return clf_gini
# Function to perform training with entropy.
def train_using_entropy(X_train, X_test, y_train):
# Decision tree with entropy
 clf_entropy = DecisionTreeClassifier(
 criterion = "entropy", random_state = 100,
 max_depth = 3, min_samples_leaf = 5)
# Performing training
```

```
 # Performing training
  clf_entropy.fit(X_train, y_train)
  return clf_entropy
 # Function to make predictions
 def prediction(X_test, clf_object):
 # Predicton on test with giniIndex
  y_pred = clf_object.predict(X_test)
  print("Predicted values:")
  print(y_pred)
  return y_pred
 # Function to calculate accuracy
 def cal_accuracy(y_test, y_pred):
  print("Confusion Matrix: ",
  confusion_matrix(y_test, y_pred))
  print ("Accuracy : ",
  accuracy_score(y_test,y_pred)*100)
  print("Report : ",
  classification_report(y_test, y_pred))
 # Driver code
 def main():
 # Building Phase
```

```
 classification_report(y_test, y_pred))
# Driver code
def main():
# Building Phase
 data = importdata()
 X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
 clf_gini = train_using_gini(X_train, X_test, y_train)
 clf_entropy = train_using_entropy(X_train, X_test, y_train)
# Operational Phase
 print("Results Using Gini Index:")
# Prediction using gini
 y_pred_gini = prediction(X_test, clf_gini)
 cal_accuracy(y_test, y_pred_gini)
 print("Results Using Entropy:")
# Prediction using entropy
 y_pred_entropy = prediction(X_test, clf_entropy)
 cal_accuracy(y_test, y_pred_entropy)
# Calling main function
if __name__=="__main__":
 main()
```

```
Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
Results Using Gini Index:
Predicted values:
['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 67 18]
 [ 0 19 71]]
```

```
Accuracy :  73.40425531914893
Report :               precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.73      0.79      0.76        85
           R       0.74      0.79      0.76        90

    accuracy                           0.73       188
   macro avg       0.49      0.53      0.51       188
weighted avg       0.68      0.73      0.71       188

Results Using Entropy:
Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 63 22]
 [ 0 20 70]]
Accuracy :  70.74468085106383
```

```
Accuracy :  70.74468085106383
Report :               precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.71      0.74      0.72        85
           R       0.71      0.78      0.74        90

    accuracy                           0.71       188
   macro avg       0.47      0.51      0.49       188
weighted avg       0.66      0.71      0.68       188

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score
  _warn_prf(average, modifier, msg_start, len(result))
```

16.2.4 Logistic Regression

Logistic Regression (LR) is a classification algorithm that is used in Machine Learning to predict the likelihood of a categorical dependent variable. It is also known as "logistic regression." The dependent variable in logistic regression is a binary variable, which means that it comprises data that is either recorded as 1 (yes, success, etc.) or 0. (no, failure, etc.).

It should be brought to your attention that the Naive Bayes model is a generative model, whereas the LR model is a discriminative model. LR performs better than naive bayes when it comes to colinearity. This is because naive bayes expects all of the characteristics to be independent, while LR does not. Naive bayes works well with small datasets.

You have already discussed this classifier in detail in Block 3 Unit 10 of this course, you may refer to Block 3 Unit 10 to understand the concept.

Implementation code in Python

The screenshot of the executed code is given below

```
# Logistic Regression

# Importing the libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the datasets

datasets = pd.read_csv('Social_Network_Ads.csv')
X = datasets.iloc[:, [2,3]].values
Y = datasets.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set

from sklearn.model_selection import train_test_split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.25, 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)

# Fitting the Logistic Regression into the Training set

from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_Train, Y_Train)

# Predicting the test set results

Y_Pred = classifier.predict(X_Test)

# Making the Confusion Matrix
```

```
# Making the Confusion Matrix

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_Test, Y_Pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap
X_Set, Y_Set = X_Train, Y_Train
X1, X2 = np.meshgrid(np.arange(start = X_Set[:,0].min() -1, stop = X_Set[:, 0].max() +1, step = 0.01),
                     np.arange(start = X_Set[:,1].min() -1, stop = X_Set[:, 1].max() +1, step = 0.01))

plt.contourf(X1,X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X2.max())
plt.ylim(X2.min(), X2.max())
```

```
for i, j in enumerate(np.unique(Y_Set)):
    plt.scatter(X_Set[Y_Set == j, 0], X_Set[Y_Set == j,1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression ( Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which
*c* argument looks like a single numeric RGB or RGBA sequence, which
```

16.2.5 Support Vector Machine

Support Vector Machine, more usually referred to as SVM, is a technique for supervised and linear machine learning that is most frequently utilised for the purpose of addressing classification issues. Support Vector Classification is another name for SVM. In addition, there is a subset of SVM known as SVR, which stands for Support Vector Regression. SVR applies the similar concepts to the problem-solving process when addressing regression issues. SVM also offers a method known as the kernel method, which is also known as the kernel SVM. This method enables us to deal with non-linearity.

The following are the steps involved in implementation:

- Import the Libraries
- Make sure the Dataset is loaded.
- Dataset will be divided into X and Y.
- Create a Training set and a Test set from the X and Y Datasets.
- Scaling the features should be done.
- Ensure that the SVM is adjusted to the Training set.
- Make a guess about the results of the test set.
- Construct the Matrix of Confusion.

You have already discussed this classifier in detail in Block 3 Unit 10 of this course, you may refer to Block 3 Unit 10 to understand the concept.

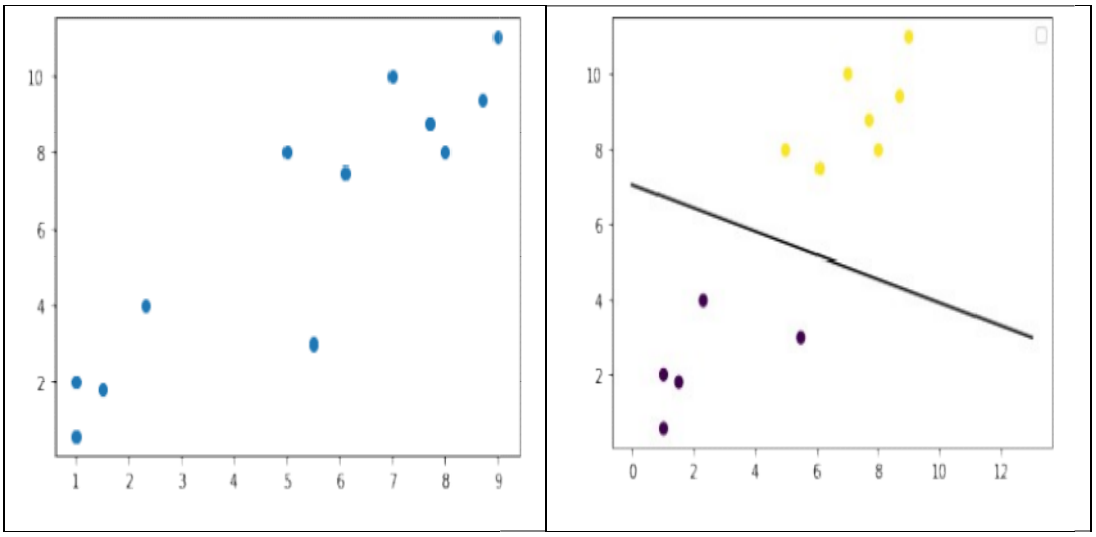
Implementation code in Python

The screenshot of the executed code is given below

```
# SUPPORT VECTOR MACHINES
# Importing Libraries
import matplotlib.pyplot as plt
import numpy as np
from sklearn import svm
# linear data
X = np.array([1, 5, 1.5, 8, 1, 9, 7, 8.7, 2.3, 5.5, 7.7, 6.1])
y = np.array([2, 8, 1.8, 8, 0.6, 11, 10, 9.4, 4, 3, 8.8, 7.5])
# show unclassified data
plt.scatter(X, y)
plt.show()
# shaping data for training the model
training_X = np.vstack((X, y)).T
training_y = [0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1]
# define the model
clf = svm.SVC(kernel='linear', C=1.0)
# train the model
clf.fit(training_X, training_y)
# get the weight values for the linear equation from the trained SVM  model
w = clf.coef_[0]
# get the y-offset for the linear equation
a = -w[0] / w[1]
# make the x-axis space for the data points
XX = np.linspace(0, 13)
# get the y-values to plot the decision boundary
yy = a * XX - clf.intercept_[0] / w[1]
```

```
# get the y-values to plot the decision boundary
yy = a * XX - clf.intercept_[0] / w[1]
# plot the decision boundary
plt.plot(XX, yy, 'k-')
# show the plot visually
plt.scatter(training_X[:, 0], training_X[:, 1], c=training_y)
plt.legend()
plt.show()
```

OUTPUT:



Check Your Progress - 1

1. Make Suitable assumptions and modify the python code of following Classification algorithms:
    a. K-NN
    b. Decision Tree
    c. Logistic Regression
    d. Support Vector Machines

16.3 REGRESSION ALGORITHMS

We learned about the basic concept of regression in the respective unit of this course, in this unit we will implement the Linear regression and Polynomial regression in Python language. Lets start with the Linear regression.

16.3.1 Linear Regression

The purpose of a linear regression model is to determine whether or not there is a connection between one or more characteristics (also known as independent variables) and a target variable that is continuous (dependent variable). Linear Regression is referred to as Uni-variate Linear Regression when there is only one feature, and it is referred to as Several Linear Regression when there are multiple features.

Following are the stages involved in the implementation of a linear regression model:

- Firstly, initialise the parameters.
- Given the value of an independent variable, predict what the value of a dependent variable will be.
- Determine the amount of error that each forecast has for each data point.
- Using a0 and a1, perform the calculation for the partial derivative.

- Add up the individual costs that you have determined for each of the numbers.

You have already discussed this classifier in detail in Block 3 Unit 11 of this course, you may refer to Block 3 Unit 11 to understand the concept.

Implementation code in Python

The screenshot of the executed code is given below

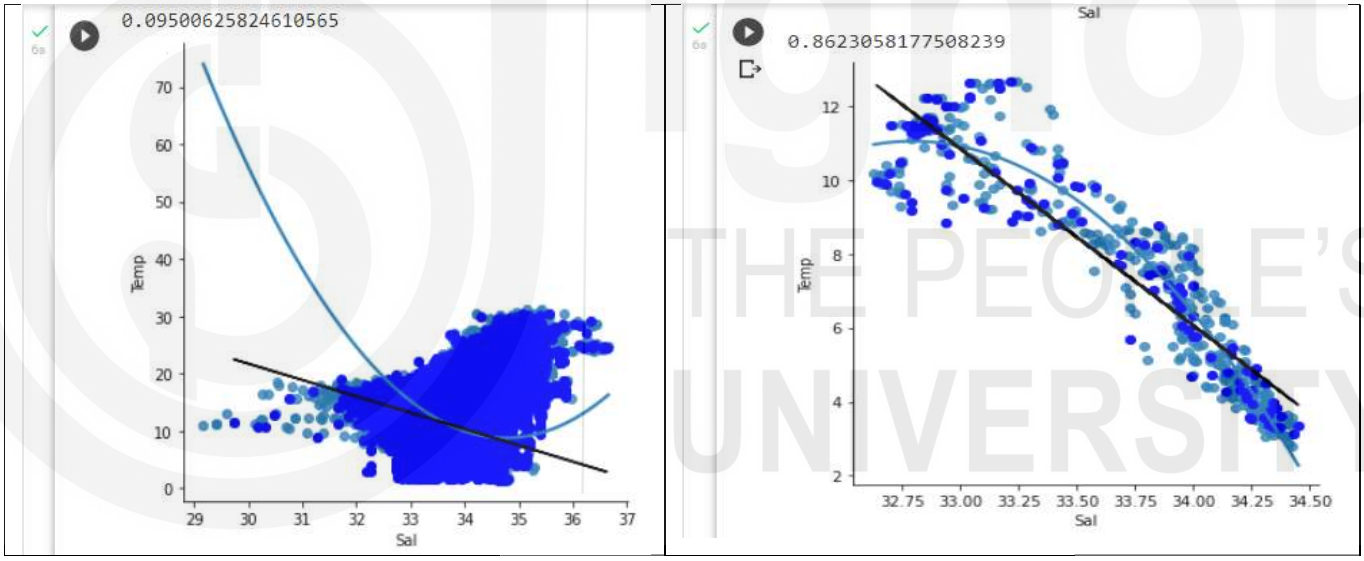
```
# LINEAR REGRESSION

# Importing Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing, svm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
#Reading Dataset, and Changing the file read location to the location of the dataset
df = pd.read_csv('bottle.csv')
df_binary = df[['Salnty', 'T_degC']]
# Taking only the selected two attributes from the dataset
df_binary.columns = ['Sal', 'Temp']
# Renaming the columns for easier writing of the code
df_binary.head()
# Displaying only the 1st rows along with the column names
##Exploring data scatter
sns.lmplot(x ="Sal", y ="Temp", data = df_binary, order = 2, ci = None)
# Plotting the data scatter
##Data Cleaning
# Eliminating NaN or missing input numbers
df_binary.fillna(method ='ffill', inplace = True)
##Training Model
X = np.array(df_binary['Sal']).reshape(-1, 1)
```

```
##Training Model
X = np.array(df_binary['Sal']).reshape(-1, 1)
y = np.array(df_binary['Temp']).reshape(-1, 1)
# Separating the data into independent and dependent variables # Converting each dataframe into a numpy array
# since each dataframe contains only one column
df_binary.dropna(inplace = True)
# Dropping any rows with Nan values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
# Splitting the data into training and testing data
regr = LinearRegression()
regr.fit(X_train, y_train)
print(regr.score(X_test, y_test))
##Exploring our results
y_pred = regr.predict(X_test)
plt.scatter(X_test, y_test, color ='b')
plt.plot(X_test, y_pred, color ='k')
plt.show()
# Data scatter of predicted values
##Working with a smaller dataset.
df_binary500 = df_binary[:][:500]
# Selecting the 1st 500 rows of the data
sns.lmplot(x ="Sal", y ="Temp", data = df_binary500,order = 2, ci = None)
df_binary500.fillna(method ='ffill', inplace = True)
X = np.array(df_binary500['Sal']).reshape(-1, 1)
y = np.array(df_binary500['Temp']).reshape(-1, 1)
df_binary500.dropna(inplace = True)
```

```
df_binary500.fillna(method ='ffill', inplace = True)
X = np.array(df_binary500['Sal']).reshape(-1, 1)
y = np.array(df_binary500['Temp']).reshape(-1, 1)
df_binary500.dropna(inplace = True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
regr = LinearRegression()
regr.fit(X_train, y_train)
print(regr.score(X_test, y_test))
y_pred = regr.predict(X_test)
plt.scatter(X_test, y_test, color ='b')
plt.plot(X_test, y_pred, color ='k')
plt.show()
```

OUTPUT:



16.3.2 Polynomial Regression

Polynomial Regression is a type of linear regression in which the relationship between the independent variable x and the dependent variable y is described as an nth degree polynomial. This type of regression is also known as "extended" linear regression. Polynomial regression is used to model a nonlinear relationship between the value of an independent variable x and the conditional mean of a dependent variable y. This relationship is represented by the notation E(y |x). solely as a result of the non-linear relationship that exists between the dependent and the independent variables When we want to transform linear regression into polynomial regression, we just add some polynomial terms.

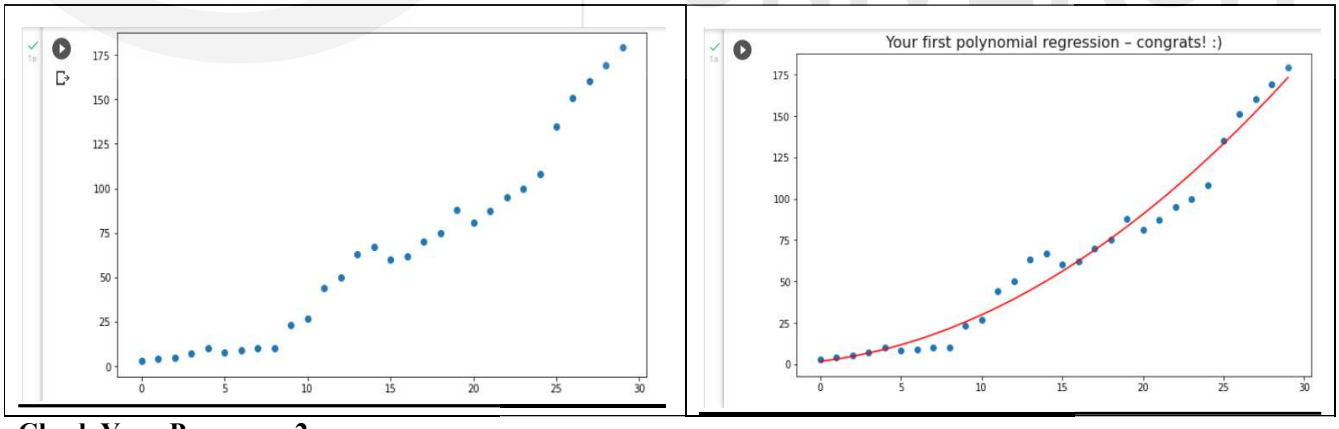
You have already discussed this classifier in detail in Block 3 Unit 11 of this course, you may refer to Block 3 Unit 11 to understand the concept.

Implementation code in Python

The screenshot of the executed code is given below

```
#POLYNOMIAL REGRESSION
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
x = np.arange(0, 30)
y = [3, 4, 5, 7, 10, 8, 9, 10, 10, 23, 27, 44, 50, 63, 67, 60, 62, 70, 75, 88,
     81, 87, 95, 100, 108, 135, 151, 160, 169, 179]
plt.figure(figsize=(10,6))
plt.scatter(x, y)
plt.show()
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2, include_bias=False)
poly_features = poly.fit_transform(x.reshape(-1, 1))
poly_features = poly.fit_transform(x.reshape(-1, 1))
from sklearn.linear_model import LinearRegression
poly_reg_model = LinearRegression()
poly_reg_model.fit(poly_features, y)
y_predicted = poly_reg_model.predict(poly_features)
plt.figure(figsize=(10, 6))
plt.title("Your first polynomial regression - congrats! :)", size=16)
plt.scatter(x, y)
plt.plot(x, y_predicted, c="red")
plt.show()
```

OUTPUT:



Check Your Progress - 2

2. Make Suitable assumptions and modify the python code of following Regression algorithms:
   a. Linear regression
   b. Polynomial egression

16.4 FEATURE SELECTION AND EXTRACTION

Feature selection and extraction are one of the most important steps that must be performed in order for machine learning to be successful. While we covered the theoretical aspects of this process in the earlier units of this course, it is now time to understand the implementation part of the mechanisms that we have learned for Feature selection and extraction. Let's begin with dimensionality reduction, which is the process of lowering the number of random variables that are being considered by generating a set of primary variables. Dimensionality reduction may be seen of as a way to streamline the analysis process.

16.4.1 Principal Component Analysis (PCA)

You have already discussed this classifier in detail in Block 4 Unit 13 of this course, you may refer to Block 4 Unit 13 to understand the concept.

Among the various techniques the Principal Component Analysis (PCA) is most frequently used, and the implementation of PCA is given below:

Implementation code in Python

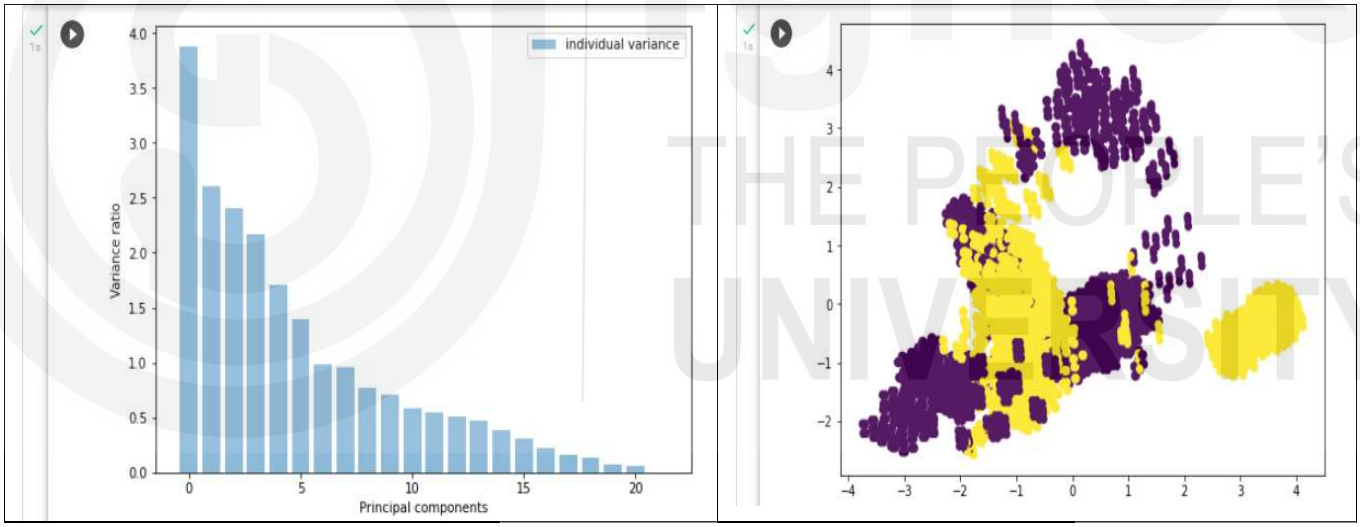
The screenshot of the executed code is given below

```
# PRINCIPAL COMPONENT ANALYSISI(PCA)
# Importing the libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
m_data = pd.read_csv('mushrooms.csv')

# Machine learning systems work with integers, we need to encode these
# string characters into ints
encoder = LabelEncoder()
# Now apply the transformation to all the columns:
for col in m_data.columns:
 m_data[col] = encoder.fit_transform(m_data[col])
X_features = m_data.iloc[:,1:23]
y_label = m_data.iloc[:, 0]
# Scale the features
scaler = StandardScaler()
X_features = scaler.fit_transform(X_features)
# Visualize
```

```
# Visualize
pca = PCA()
pca.fit_transform(X_features)
pca_variance = pca.explained_variance_
plt.figure(figsize=(8, 6))
plt.bar(range(22), pca_variance, alpha=0.5, align='center',  label='individual variance')
plt.legend()
plt.ylabel('Variance ratio')
plt.xlabel('Principal components')
plt.show()
pca2 = PCA(n_components=17)
pca2.fit(X_features)
x_3d = pca2.transform(X_features)
plt.figure(figsize=(8,6))
plt.scatter(x_3d[:,0], x_3d[:,5], c=m_data['class'])
plt.show()
```

OUTPUT :



Check Your Progress - 3

3. Make Suitable assumptions and modify the python code of Principal Component Analysis, for dimensionality reduction.

16.5 ASSOCIATION RULES

We discussed Apriori algorithm and FP Growth algorithm, while studying the topic of Association Rules. These algorithms are frequently used in pattern matching. Since FP Growth is a step ahead to Apriori Algorithm, we are discussing the implementation of Apriori algorithm only

16.5.1 Apriori Algorithm

The Apriori algorithm is a data mining technique that is used for mining frequent item sets and appropriate association rules. It does this by using a mathematical formula. We focused on the definitions of association rule mining and Apriori algorithms, as well as the application of an Apriori algorithm, in the area of this class that was most pertinent to the topic. In this section, we will construct one Apriori model utilising the Python programming language and a hypothetical situation involving a small firm. However, it does have some limits, the effects of which can be mitigated using a variety of different approaches. Data mining and pattern recognition are two of the many applications that see widespread use of the method.

The candidate set is produced by the model that is described further down below by merging the set of frequent items from the step before it.

Conduct testing on subsets, and if the candidate set contains infrequent item sets, remove them. And then calculate the final frequent itemset by obtaining the items that meet the minimal support requirement.

You have already discussed this classifier in detail in Block 4 Unit 14 of this course, you may refer to Block 4 Unit 14 to understand the concept.

Implementation code in Python

The screenshot of the executed code is given below

```
# APRIORI ALGORITHM
##Importing the required libraries
import numpy as np
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
## Loading and exploring the data
# Changing the working location to the location of the file
# Loading the Data
data = pd.read_excel('Online Retail.xlsx')
data.head()
# Exploring the columns of the data
data.columns
# Exploring the different regions of transactions
data.Country.unique()
# Stripping extra spaces in the description
data['Description'] = data['Description'].str.strip()
# Dropping the rows without any invoice number
data.dropna(axis = 0, subset =['InvoiceNo'], inplace = True)
data['InvoiceNo'] = data['InvoiceNo'].astype('str')
# Dropping all transactions which were done on credit
data = data[~data['InvoiceNo'].str.contains('C')]

##Splitting the data according to the region of transaction
```

```
##Splitting the data according to the region of transaction
# Transactions done in France
basket_France = (data[data['Country'] =="France"]
.groupby(['InvoiceNo', 'Description'])['Quantity']
.sum().unstack().reset_index().fillna(0)
.set_index('InvoiceNo'))
# Transactions done in the United Kingdom
basket_UK = (data[data['Country'] =="United Kingdom"] .groupby(['InvoiceNo', 'Description'])['Quantity']
.sum().unstack().reset_index().fillna(0)
.set_index('InvoiceNo'))
# Transactions done in Portugal
basket_Por = (data[data['Country'] =="Portugal"]
.groupby(['InvoiceNo', 'Description'])['Quantity']
.sum().unstack().reset_index().fillna(0)
.set_index('InvoiceNo'))
# Transactions done in Sweden
basket_Sweden = (data[data['Country'] =="Sweden"]
.groupby(['InvoiceNo', 'Description'])['Quantity']
.sum().unstack().reset_index().fillna(0)
.set_index('InvoiceNo'))
## Hot encoding the Data
```

```
# Defining the hot encoding function to make the data suitable
 # for the concerned libraries
def hot_encode(x):
 if(x<= 0):
  return 0
 if(x>= 1):
  return 1
# Encoding the datasets
basket_encoded = basket_France.applymap(hot_encode)
basket_France = basket_encoded
basket_encoded = basket_UK.applymap(hot_encode)
basket_UK = basket_encoded
basket_encoded = basket_Por.applymap(hot_encode)
basket_Por = basket_encoded
basket_encoded = basket_Sweden.applymap(hot_encode)
basket_Sweden = basket_encoded
## Building the models and analyzing the results
# Building the model
###France
frq_items = apriori(basket_France, min_support = 0.05, use_colnames = True)
# Collecting the inferred rules in a dataframe
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False,  False])
```

```
# Collecting the inferred rules in a dataframe
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False,  False])
print(rules.head())
###United Kingdom
frq_items = apriori(basket_UK, min_support = 0.01, use_colnames = True)
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False,  False])
print(rules.head())
###Portugal
frq_items = apriori(basket_Por, min_support = 0.05, use_colnames =
True)
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False,  False])
print(rules.head())
### Sweden
frq_items = apriori(basket_Sweden, min_support = 0.05, use_colnames = True)
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending =[False,  False])
print(rules.head())
```

```
2019  (SUKI  SHOULDER BAG, JAM MAKING SET PRINTED)     (DOTCOM POSTAGE)
2295        (HERB MARKER MINT, HERB MARKER THYME)  (HERB MARKER ROSEMARY)
2300   (HERB MARKER PARSLEY, HERB MARKER ROSEMARY)     (HERB MARKER THYME)
2301      (HERB MARKER PARSLEY, HERB MARKER THYME)  (HERB MARKER ROSEMARY)

      antecedent support  consequent support   support  confidence      lift  \
116             0.011036             0.037928  0.010768    0.975728  25.725872
2019            0.011625             0.037928  0.011196    0.963134  25.393807
2295            0.010714             0.012375  0.010232    0.955000  77.173095
2300            0.011089             0.012321  0.010553    0.951691  77.240055
2301            0.011089             0.012375  0.010553    0.951691  76.905682

      leverage  conviction
116   0.010349   39.637371
2019  0.010755   26.096206
2295  0.010099   21.947227
2300  0.010417   20.444951
2301  0.010416   20.443842
                           antecedents                          consequents  \
1170  (SET 12 COLOUR PENCILS DOLLY GIRL)     (SET 12 COLOUR PENCILS SPACEBOY)
1171    (SET 12 COLOUR PENCILS SPACEBOY)   (SET 12 COLOUR PENCILS DOLLY GIRL)
1172  (SET 12 COLOUR PENCILS DOLLY GIRL)   (SET OF 4 KNICK KNACK TINS LONDON)
1173  (SET OF 4 KNICK KNACK TINS LONDON)   (SET 12 COLOUR PENCILS DOLLY GIRL)
1174  (SET 12 COLOUR PENCILS DOLLY GIRL)  (SET OF 4 KNICK KNACK TINS POPPIES)
```

```
      antecedent support  consequent support   support  confidence       lift  \
1170            0.051724            0.051724  0.051724         1.0  19.333333
1171            0.051724            0.051724  0.051724         1.0  19.333333
1172            0.051724            0.051724  0.051724         1.0  19.333333
1173            0.051724            0.051724  0.051724         1.0  19.333333
1174            0.051724            0.051724  0.051724         1.0  19.333333

      leverage  conviction
1170  0.049049         inf
1171  0.049049         inf
1172  0.049049         inf
1173  0.049049         inf
1174  0.049049         inf
                        antecedents                       consequents  \
0     (12 PENCILS SMALL TUBE SKULL)      (PACK OF 72 SKULL CAKE CASES)
1     (PACK OF 72 SKULL CAKE CASES)      (12 PENCILS SMALL TUBE SKULL)
4           (36 DOILIES DOLLY GIRL)     (ASSORTED BOTTLE TOP  MAGNETS)
5    (ASSORTED BOTTLE TOP  MAGNETS)           (36 DOILIES DOLLY GIRL)
180  (CHILDRENS CUTLERY DOLLY GIRL)  (CHILDRENS CUTLERY CIRCUS PARADE)

     antecedent support  consequent support   support  confidence  lift  \
0              0.055556            0.055556  0.055556         1.0  18.0
1              0.055556            0.055556  0.055556         1.0  18.0
4              0.055556            0.055556  0.055556         1.0  18.0
5              0.055556            0.055556  0.055556         1.0  18.0
```

```
                        antecedents                       consequents  \
0     (12 PENCILS SMALL TUBE SKULL)      (PACK OF 72 SKULL CAKE CASES)
1     (PACK OF 72 SKULL CAKE CASES)      (12 PENCILS SMALL TUBE SKULL)
4           (36 DOILIES DOLLY GIRL)     (ASSORTED BOTTLE TOP  MAGNETS)
5    (ASSORTED BOTTLE TOP  MAGNETS)           (36 DOILIES DOLLY GIRL)
180  (CHILDRENS CUTLERY DOLLY GIRL)  (CHILDRENS CUTLERY CIRCUS PARADE)

     antecedent support  consequent support   support  confidence  lift  \
0              0.055556            0.055556  0.055556         1.0  18.0
1              0.055556            0.055556  0.055556         1.0  18.0
4              0.055556            0.055556  0.055556         1.0  18.0
5              0.055556            0.055556  0.055556         1.0  18.0
180            0.055556            0.055556  0.055556         1.0  18.0

     leverage  conviction
0    0.052469         inf
1    0.052469         inf
4    0.052469         inf
5    0.052469         inf
180  0.052469         inf
```

16.6 CLUSTERING ALGORITHMS

We learned about the theoretical aspects of various clustering algorithms like K-Means, DBSCAN etc. in the respective unit of this course. The K-Means algorithm was quite simple and hence its implementation is given below:

16.6.1 K-Means - Implementation code in Python

You have already discussed this classifier in detail in Block 4 Unit 15 of this course, you may refer to Block 4 Unit 15 to understand the concept.

Implementation code in Python

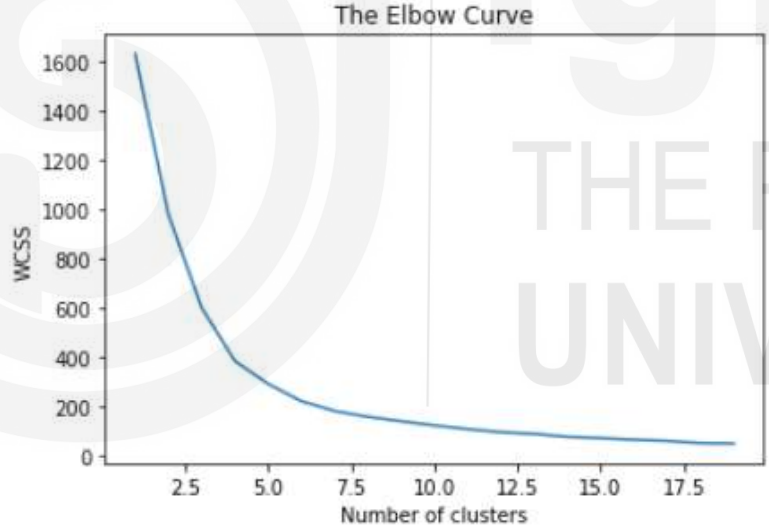
The screenshot of the executed code is given below

```
# K-MEANS CLUSTERING
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
x1=10*np.random.rand(100,2)
x1.shape
kmean=KMeans(n_clusters=3)
kmean.fit(x1)
kmean.cluster_centers_
kmean.labels_
wcss = []
for i in range(1,20):
 kmeans = KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0)
 kmeans.fit(x1)
 wcss.append(kmeans.inertia_)
 print('Cluster', i, 'Inertia', kmeans.inertia_)
plt.plot(range(1,20),wcss)
plt.title('The Elbow Curve')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') ##WCSS stands for total within-cluster sum of square
plt.show()
```

```
plt.ylabel('WCSS') ##WCSS stands for total within-cluster sum of square
plt.show()
```

```
Cluster 1 Inertia 1633.4729386366648
Cluster 2 Inertia 980.7251659226347
Cluster 3 Inertia 602.6606235675433
Cluster 4 Inertia 386.8266954126674
Cluster 5 Inertia 293.52522127511526
Cluster 6 Inertia 223.57829467348864
Cluster 7 Inertia 183.55842327586788
Cluster 8 Inertia 160.52571627742793
Cluster 9 Inertia 142.12347605903437
Cluster 10 Inertia 125.1405682359277
Cluster 11 Inertia 110.07801216378505
Cluster 12 Inertia 97.71840448709966
Cluster 13 Inertia 91.0468655863335
Cluster 14 Inertia 78.97168184469689
Cluster 15 Inertia 73.78690617997606
Cluster 16 Inertia 67.33331767461436
Cluster 17 Inertia 62.26772230658193
Cluster 18 Inertia 53.44137427335066
Cluster 19 Inertia 51.55574132217379
```

```
Cluster 14 Inertia 78.97168184469689
Cluster 15 Inertia 73.78690617997606
Cluster 16 Inertia 67.33331767461436
Cluster 17 Inertia 62.26772230658193
Cluster 18 Inertia 53.44137427335066
Cluster 19 Inertia 51.55574132217379
```



Check Your Progress - 4

4. Make Suitable assumptions and modify the python code of K-Means algorithm

16.8 SUMMARY

In this unit we understood the implementation of various machine learning algorithms for Classification, Regression, Dimension Reductionality and clustering. The theoretical aspects of the respective algorithm were already discussed in the respective units of this course.

16.9 SOLUTIONS/ANSWERS

Check Your Progress - 1

1. Make Suitable assumptions and modify the python code of following Classification algorithms:
    a. K-NN
    b. Decision Tree
    c. Logistic Regression
    d. Support Vector Machines

Solution : Refer to section 16.2

Check Your Progress - 2

2. Make Suitable assumptions and modify the python code of following Regression algorithms:
    a. Linear regression
    b. Polynomial egression

Solution : Refer to section 16.3

Check Your Progress - 3

3. Make Suitable assumptions and modify the python code of Principal Component Analysis, for dimensionality reduction.

Solution : Refer to section 16.4

Check Your Progress - 4

4. Make Suitable assumptions and modify the python code of K-Means algorithm

Solution : Refer to section 16.6

16.10 FURTHER READINGS

- https://www.kaggle.com/
- https://www.github.com/
- https://towardsdatascience.com
- https://machinelearningmastery.com