Lab Code: 20ECL501/SOC3
MACHINE LEARNING
Lab Manual

Department of Electronics & Communication Engineering

Bapatla Engineering College :: Bapatla

(Autonomous)

G.B.C. Road, Mahatmajipuram, Bapatla-522102, Guntur (Dist.)
Andhra Pradesh, India.
E-Mail:bec.principal@becbapatla.ac.in
Web:www.becbapatla.ac.in

Contents

12.	Write a program demonstrating how the Principal Component Analysis (PCA) works. Select the appropriate data set for your experiment and draw graphs.
13.	Write a program demonstrating how the Kernal Principal Component Analysis (K-PCA) works. Select the appropriate data set for your experiment and draw graphs.
14.	Write a program demonstrating how the Q-Learning Algorithm works. Select the appropriate data set for your experiment and draw graphs.

Bapatla Engineering College :: Bapatla
(Autonomous)

Vision

- To build centers of excellence, impart high quality education and instill high standards of ethics and professionalism through strategic efforts of our dedicated staff, which allows the college to effectively adapt to the ever changing aspects of education.

- To empower the faculty and students with the knowledge, skills and innovative thinking to facilitate discovery in numerous existing and yet to be discovered fields of engineering, technology and interdisciplinary endeavors.

Mission

- Our Mission is to impart the quality education at par with global standards to the students from all over India and in particular those from the local and rural areas.

- We continuously try to maintain high standards so as to make them technologically competent and ethically strong individuals who shall be able to improve the quality of life and economy of our country.

Bapatla Engineering College :: Bapatla

(Autonomous)

Department of Electronics and Communication Engineering

Vision

To produce globally competitive and socially responsible Electronics and Communication Engineering graduates to cater the ever changing needs of the society.

Mission

- To provide quality education in the domain of Electronics and Communication Engineering with advanced pedagogical methods.

- To provide self learning capabilities to enhance employability and entrepreneurial skills and to inculcate human values and ethics to make learners sensitive towards societal issues.

- To excel in the research and development activities related to Electronics and Communication Engineering.

Bapatla Engineering College :: Bapatla

(Autonomous)

Department of Electronics and Communication Engineering

Program Educational Objectives (PEO's)

PEO-I: Equip Graduates with a robust foundation in mathematics, science and Engineering Principles, enabling them to excel in research and higher education in Electronics and Communication Engineering and related fields.

PEO-II: Impart analytic and thinking skills in students to develop initiatives and innovative ideas for Start-ups, Industry and societal requirements.

PEO-III: Instill interpersonal skills, teamwork ability, communication skills, leadership, and a sense of social, ethical, and legal duties in order to promote lifelong learning and Professional growth of the students.

Program Outcomes (PO's)

Engineering Graduates will be able to:

PO1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7.Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and Teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-long learning: Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Bapatla Engineering College :: Bapatla

(Autonomous)

Department of Electronics and Communication Engineering

Program Specific Outcomes (PSO's)

PSO1: Develop and implement modern Electronic Technologies using analytical methods to meet current as well as future industrial and societal needs.

PSO2: Analyze and develop VLSI, IoT and Embedded Systems for desired specifications to solve real world complex problems.

PSO3: Apply machine learning and deep learning techniques in communication and signal processing.

MACHINE LEARNING
III B.Tech. V Semester (Code:20ECL501/SOC3)

Lectures	:	1	Tutorial	:	0	Practica	:	2
CIE	:	30	SEE	:	70	Credits	:	2

Pre-Requisite: None

Course Objectives: Students will learn how to

- Understand how a machine learns and various applications of machine
- Distinguish between classification and regression
- Fundamentals of Artificial neural networks
- Gain knowledge in Support Vector Machine and Baye's classifier

Course Outcomes: After studying this course, the students will be able to

CO1	Analyze the mathematical and statistical prospective of machine learning algorithms through python programming
CO2	Evaluate the machine learning models pre-processed through various
CO3	Design and develop the code for recommender system using Natural Language
CO4	Apply various Baye's techniques for data clustering.

Mapping of Course Outcomes with Program Outcomes & Program

	PO's												PSO's		
CO	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**1**	**2**	**3**
CO1	2	2	3		3										3
CO2	3	3			3										3
CO3	3	3			3										3
CO4	2	2			3										3
AVG	2.5	2.5	3		3										3

Syllabus

UNIT-1 (12 Hours)

INTRODUCTION - Towards Intelligent Machines, Well-Posed Machine Learning Problems, Examples of Applications in Diverse Fields, Data Representation, Domain Knowledge for Productive use of Machine Learning,

UNIT-2 (12 Hours)

DECISION TREE LEARNING – Introduction, Decision tree representation, Appropriate problems for decision tree learning. Linear Regression with Least Square Error Criterion, Logistic Regression for Classification Tasks, Fisher's Linear Discriminant and Thresholding for Classification Minimum Description

UNIT-3 (12 Hours)

ARTIFICIAL NEURAL NETWORKS– Neural network representation, Appropriate problems for neural network learning, Perceptrons - Gradient descent and the Deltarule, Multilayer networks and The back propagation	
UNIT-4	(12 Hours)
BAYESIAN LEARNING – Bayes theorem, Learning with Support Vector Machines (SVM), Variants of Basic SVM Techniques.	

PRACTICAL EXERCISES

S.No.	Title of the Experiment
1.	Create an array using Numpy Library and perform basic operations.
2.	Import a .CSV file in PANDAS Library and perform basic operations.
3.	Plot the different plots in MATPLOT Library
4.	Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.
5.	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.
6.	Implement the Simple Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs
7.	Implement the Logistic Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs
8.	Write a program to demonstrate the working of the decision tree algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
9.	Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering.

10. Write a program to demonstrate the working of the Support Vector Machine (SVM). Select appropriate data set for your experiment and draw graphs.

11. Write a program demonstratinghow the Hierarchical Cluster Analysis (HCA) works. Select the appropriate data set for your experiment and draw graphs.

12. Write a program demonstratinghow the Principal Component Analysis (PCA) works. Select the appropriate data set for your experiment and draw graphs.

13. Write a program demonstrating how the KernalPrincipal Component Analysis (K-PCA) works. Select the appropriate data set for your experiment and draw graphs.

14. Write a program demonstrating how the Q-Learning Algorithm works. Select the appropriate data set for your experiment and draw graphs.

1. Numpy Library

Aim: To write a Python to implement of operations on matrices using numpy library.

Software Required:

Google Colab

Theory:

Numpy is a Python package which means 'Numerical Python'. It is the library for logical computing, which contains a powerful n-dimensional array object, gives tools to integrate C, C++ and so on. It is likewise helpful in linear based math, arbitrary number capacity and so on. NumPy exhibits can likewise be utilized as an effective multi-dimensional compartment for generic data. NumPy Array: Numpy array is a powerful N-dimensional array object which is in the form of rows and columns. We can initialize NumPy arrays from nested Python lists and access it elements. A Numpy array on a structural level is made up of a combination of:

- The Data pointer indicates the memory address of the first byte in the array.
- The Data type or dtype pointer describes the kind of elements that are contained within the array.
- The shape indicates the shape of the array.
- The strides are the number of bytes that should be skipped in memory to go to the next element.

Program:

i.Basic Data Structures in Python List, tuple, set and Dictionary

```
import numpy as np
a=[1,2,3] #creating a list, list is mutable
print(a)
a.append(4)
```

```
print(a)
b=(3,4) #creating a tuple, tuple is immutable, i.e.,

#elements can neither be added nor deleted.
c={1,3,5,7,7} #creating a set, set is mutable and duplicate elementsare
removed.
c.add(11)
print(c)

#Dictionary in Python is a collection of key-value pairs
#used to store data values like a map,

dict={1:'study',2:'play',3:'sleep'}
print(dict)
print(dict.keys())
print(dict.values())
```

ii. Creating Arrays

```
import numpy as np
a=np.array([1,2,3])
b=np.array([(1,2,3),(4,5,6)])
print(np.zeros(3))
print(np.ones((3,4)))
print(np.eye(5))
print(np.full((2,3),8))
print(np.random.random(5))
print(np.random.rand(2,3))
print(np.random.randint(1,10))
print(np.arange(0,10,1))
```

iii. Inspecting Properties

```
import numpy as np
data1=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(data1)
print(np.size(data1)) #Retuns total number of elements in the array
print(np.ndim(data1))#Returns number of dimensions of array
print(np.shape(data1)) #Returns tuple of integers representing
#the size of the array in each dimension
data2=np.array([9,7,1,2])
print(data2.dtype)
```

iv.Copying/Sorting/Reshaping

```
import numpy as np
a=np.array([1,2,3,4])
b=np.array([[1,2,3],[4,5,6],[7,8,9]])
s=np.copy(a) #Copies array to new memory
print(s)
print(b.flatten()) #Flattens 2D array to 1D array
print(b.reshape(9,1))
print(np.resize(b,(2,2)))
```

v.Adding/Removing Elements

```
b1=np.array([1,2,3,4,5])
print(np.append(b1,3)) #appends values to end of the array.
print(np.insert(b1,3,6)) #Inserts value into the array before index 3.
b2=np.array([[4,-2,1],[1,-3,0],[2,0,-1]])
print(b2)
b3=np.insert(b2,1,2,axis=1)#inserts a column of all 2's at index 1 of the
array
print(b3)
print(np.delete(b2,1,axis=0)) #Deletes row at index 1 of the array
print(np.delete(b2,0,axis=1)) #Deletes column at index 0 the array
```

vi.Combining/Splitting

```
import numpy as np
a1=np.array([[1,2,3],[3,4,5],[6,7,8]])
print(a1)
b1=np.array([[5,6,7],[7,8,9],[1,2,3]])
print(b1)
c1=np.concatenate((a1,b1),axis=0)
d1=np.concatenate((a1,b1),axis=1)
print(c1)
print(d1)
print(np.hsplit(a1,1))
print(np.vsplit(a1,1))
```

vii.Indexing/Slicing/Subsetting

```
import numpy as np
a=np.array([1,2,3,4,5,6,7])
a[3]=0 #Assigns the array element on index 3 the value of 0
print(a[2:5]) #Returns the elements at indices 2,3,4,5
b=np.array([[1,2,3],[4,5,6],[7,8,9]])
b[1,2]=-12 #Assigning the value -12 to element at index [1][2]
print(b)
print(b[1,:])
print(b[:,2])
print(b[0:2])
print(b[:,1:2])
print(b[:,[1,2]]) #selecting multiple columns at a time
print(b[[0,2],:]) #selecting multiple rows at a time
print(b<5) #Returns array with boolean values
print(b[b<5]) #Returns array elements smaller than 5
print(b.T) #Returns transpose of the array
```

viii. Scalar Math

```
data1=np.array([3,1,2,-4,5])
print(data1)
# Performs scalar arithmetic on the array
print((np.add(data1,1)),(np.subtract(data1,2)),
(np.multiply(data1,-1)))
```

ix. Vector Math

```
a1=np.array([2.7,3.1,-4.3,-5.8])
a2=np.array([1,0,9,7])
print((np.add(a1,a2)),(np.subtract(a1,a2)),
(np.multiply(a1,a2)))
print(np.array_equal(a1,a2))
print(np.log(a1)) #Natural log of each element in the array
print(np.abs(a1)) #Absolute value of each element in the array
print(np.ceil(a1)) #Rounds up to the nearest int
a3=[1.7,2.1,3.6,5.3,6.2,9.5]
print(np.floor(a3)) #Rounds down to the nearest int
print(np.round(a3)) #Rounds to the nearest integer
```

x. Statistics

```
a1=np.array([1,2,3,7,8]) #creates a numpy array
print(np.min(a1),np.max(a1),np.sum(a1))
#Returns mean, variance and standard deviation of the array.
print(np.mean(a1),np.var(a1),np.std(a1))
a2=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(np.var(a2,axis=1)) #Returns variance of the array.
print(np.corrcoef(a2[1:],a2[2:])) #Returns correlation coefficient of the array
```

Result: Hence, successfully implemented the basic operations on matrices using numpy library.

2. Pandas Library

Aim: To write a Python program for the implementation of data framing and some perform basic operations using pandas library.

Software Required:

Google Colab

Theory:

Pandas is a powerful data manipulation and analysis library for Python. It provides versatile data structures like series and dataframes, making it easy to work with numeric values. In this article, we will explore five different methods for performing numeric value operations in Pandas, along with code examples to demonstrate their usage. Numeric value operations in Pandas Python form the backbone of efficient data analysis, offering a streamlined approach to handling numerical data. With specialized data structures like Series and Data Frame, Pandas simplifies arithmetic operations, statistical calculations, and data aggregation.

Program:

i. Importing.csv file into Colab Notebook as A DataFrame

```
from google.colab import
files uploaded=files.upload()

import pandas as pd
 import io
df=pd.read_csv(io.BytesIO(uploaded['enjoysport.csv']) )
print(df)

import numpy as np
import pandas as pd
data1=[1,7,2] data1 = pd.Series(data1, index = ['x','y','z'])
```

```
#Creates a Series
 #type data structure with specified index.Default ind ex is
#integers starting from 0.

print(data1,type(data1))
print(data1['y'])#Returns value at index 'y.' #Creating a Dictionary.
data2={"Age":[25,45,22,36,29,60],"Height(inft)":[5.6,6.1,4.9,5.7,5.1,5.9],
"Qualification":["B.Tech",'B.Tech','M.Phil','Ph.D','B.Sc','CA'],
"Salary":[18000,90000,20000,50000,40000,100000],
"Married":[False,True,True,False,True,True]}

#Converts Dictionary into a DataFrame with specified index.
data2=pd.DataFrame(data2,index=['Ram','Krishna','Sita','Prasad','Gayatri','S
hankar']) print(data2,type(data2))
```

ii.Finding Summary of the DataFrame

```
data2.info()
data2.describe()
```

iii.Displaying Entries of the DataFrame print(data2.columns)

```
print(data2.index) print(data2.values) print(data2.head())
print(data2.head(2))
```

iv.Displaying Entries of the DataFrame print(data2.columns)

```
print(data2.index)
print(data2.values)
print(data2.head())
print(data2.head(2))
```

v.Slicing and Indexing of DataFrame

```
print(data2['Salary'])
print(data2['Krishna':'Gayatri'])
print(data2[0:2])
```

```
print(data2[-3:])
print(data2['Qualification'][1:3]) print(data2.loc['Ram':'Krishna','Height(in
ft)':'Qual ification'])
print(data2.iloc[0:2,1:3])
print(data2['Age']<40])
```

vi.Removing a Column or a Row from a DataFrame

```
a=data2.drop('Age',axis=1)
print(a)
b=data2.drop('Sita',axis=0)
print(b)
```

vii.Adding a Column/Row to a DataFrame

```
address=['Kkd','Rjy','Bpt','Slo','Ong','Bza',] data2['Address']=address
print(data2)
data2.loc[len(data2.index)]=[18,5.2,'MCA',10000,'False','vskp']
print(data2)
```

viii.Shuffling, Sorting and Grouping #Shuffling a Data Set

```
c=data2.reindex(np.random.permutation(data2.index))
print(c)
#Sorting
d=data2.sort_values(by='Salary',ascending=True)
print(d)
e=data2.groupby('Qualification').count()
print(e)
```

Result: Hence, successfully implemented the basic operations and data framing using panda’s library.

3. Matplot Library

Aim: To write a Python program for the plotting of basic mathematical plots using matplotlib.pyplot library.

Software Required:

Google Colab

Theory:

Matplotlib is a powerful plotting library in Python used for creating static, animated, and interactive visualizations. Matplotlib's primary purpose is to provide users with the tools and functionality to represent data graphically, making it easier to analyze and understand. It was originally developed by John D. Hunter in 2003 and is now maintained by a large community of developers. Matplotlib is easy to use and an amazing visualizing library in Python. It is built on NumPy arrays and designed to work with the broader SciPy stack and consists of several plots like line, bar, scatter, histogram, etc.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(0,10,100)
y=x*x
plt.figure(figsize=(4,2))
plt.plot(x,y)
plt.title('Square function')
plt.xlabel("x")
plt.ylabel("$x^2$")
plt.figure(figsize=(5,5))
plt.plot(x,np.sin(x))
plt.title('sin(x)')
```

```
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.figure(figsize=(6,3))
plt.plot(x,np.tan(x))
plt.title('Tangent function')
plt.xlabel("x")
plt.ylabel("tan(x)")
plt.figure(figsize=(3,3))
plt.plot(x,np.exp(x))
plt.title('Exponential function')
plt.xlabel("x")
plt.ylabel("e^x")
```

i. Scatter Plot

```
x=(np.random.random(10)*10).round(1)
y=(np.random.random(10)*10).round(2)
print(x,y,sep="\n")
plt.figure(figsize=(5,5))
plt.scatter(x,y)
plt.xlabel('x')
plt.ylabel('y')
```

ii. Bar Plot

```
items=np.array(['Coke','Pepsi','Fanta','Maaza','Mirin
da'])
qty=np.array([100,85,20,30,45])
plt.bar(items,qty)
plt.title('Sales')
plt.xlabel('Beverages')
plt.ylabel('Qty Sold')
```

iii. Pie Plot

```
plt.pie(qty,labels=items,autopct='%0.1f')#autopct is
used to la
#wedge with their numerical value.
plt.title("% of Sales")
```

iv. Histogram

```
import numpy as np
from matplotlib import pyplot as plt
marks=np.random.randint(0,100,60)
grade_intervals=[0,30,50,80,100]
#print(marks)
plt.hist(marks,grade_intervals)
plt.title('Student Grades')
plt.xlabel('Percentage')
plt.ylabel('No.of Students')
```

v. Box Plot

```
math_marks=np.random.randint(10,100,180)
phy_marks=np.random.randint(0,100,180)
chem_marks=np.random.randint(30,100,180)
marks=[math_marks,phy_marks,chem_marks]
plt.boxplot(marks,labels=['Maths','Physics','Chemistry'])
```

Result: Hence, successfully implemented the basic mathematical plots using matplot library.

4. FIND-S Algorithm

Aim: To write a Python program for the implementation of the Find-S Algorithm for the given data set.

Software Required:

Google Colab

Theory:

The find-S algorithm is a basic concept learning algorithm in machine learning. The find-S algorithm finds the most specific hypothesis that fits all the positive examples. We have to note here that the algorithm considers only those positive training example. The find-S algorithm starts with the most specific hypothesis and generalizes this hypothesis each time it fails to classify an observed positive training data. Hence, the Find-S algorithm moves from the most specific hypothesis to the most general hypothesis.

Program:

```
from google.colab import files
uploaded=files.upload()

from google.colab.output import enable_custom_widget_manager
import pandas as pd
import numpy as np
import io
#to read the data in the csv file
df=pd.read_csv(io.BytesIO(uploaded['walkinghyp.csv']))
print(df)

#making an array of all the attributes
d=np.array(df)[:,:-1]
print(d)

#segragating the target that has positive and negative examples
```

```
target = np.array(df)[:,-1]
print("The target is: ",target)

#training function to implement find-s algorithm
def train(c,t):
      for i, val in enumerate(t):
          if val == "Yes":
             specific_hypothesis= c[i].copy()
             break
for i, val in enumerate(c):
    if t[i] == "Yes":
       for x in range(len(specific_hypothesis)):
           if val[x] != specific_hypothesis[x]:
              specific_hypothesis[x] ='?'
else:
             pass
       return specific_hypothesis
print("The final hypothesis is:",train(d,target))
```

Data Set:

	A	B	C	D	E	F	G
1	**Time**	**Weather**	**Temperatu**	**Company**	**Humidity**	**Wind**	**Goes**
2	Morning	Sunny	Warm	Yes	Mild	Strong	Yes
3	Evening	Rainy	Cold	No	Mild	Normal	No
4	Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
5	Evening	Sunny	Cold	Yes	High	Strong	Yes

Expected Output:

The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']

Result: Hence, the Final Specific hypothesis is calculated for a given data set by using Find-S algorithm

5. Candidate-Elimination Algorithm

Aim: To write a Python program for the implementation of the Candidate-Elimination Algorithm for the given data set.

Software Required:

Google Colab

Theory:

The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

Step1: Load Data set

Step2: Initialize General Hypothesis and Specific Hypothesis.

Step3: For each training example

Step4: If example is positive example

```
    if attribute_value == hypothesis_value:
        Do nothing
    else:
        replace attribute value with '?' (Basically generalizing it)
```

Step5: If example is Negative example

    Make generalize hypothesis more specific.

Program:

```
from google.colab import files
uploaded=files.upload()
from google.colab.output import enable_custom_widget_manager

import numpy as np
```

```
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('cedatanew.csv'))
print(data)

concepts = np.array(data.iloc[:,0:-1])
print(concepts)

target = np.array(data.iloc[:,-1])
print(target)

def learn(concepts, target):
  specific_h = concepts[0].copy()
  print("initialization of specific_h and general_h")
  print(specific_h)
  general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
  print(general_h)
  for i, h in enumerate(concepts):
    if target[i] == "yes":
      for x in range(len(specific_h)):
        if h[x]!= specific_h[x]:
          specific_h[x] ='?'
          general_h[x][x] ='?'
        print(specific_h)
           print(general_h)
    if target[i] == "no":
      for x in range(len(specific_h)):
        if h[x]!= specific_h[x]:
          general_h[x][x] = specific_h[x]
        else:
          general_h[x][x] = '?'
    print(" steps of Candidate Elimination Algorithm",i+1)
```

```
    print(specific_h)
    print(general_h)
  indices = [i for i, val in enumerate(general_h) if val ==['?', '?', '?', '?', '?', '?']]
  for i in indices:
    general_h.remove(['?', '?', '?', '?', '?', '?'])
  return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

Data Set:

	A	B	C	D	E	F	G
1	**Time**	**Weather**	**Temperatu**	**Company**	**Humidity**	**Wind**	**Goes**
2	Morning	Sunny	Warm	Yes	Mild	Strong	Yes
3	Evening	Rainy	Cold	No	Mild	Normal	No
4	Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
5	Evening	Sunny	Cold	Yes	High	Strong	Yes

Expected Output:

Final Specific_h: ['sunny' 'warm' '?' 'strong' '?' '?']

Final General_h: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

Result: Hence, the Final Specific hypothesis and Final General are calculated for a given data set by using Candidate elimination algorithm.

6. Simple Linear Regression Algorithm

Aim: To write a Python program for the implementation of the Simple Linear Regression Algorithm for the separation of the given data set.

Software Required:

Google Colab

Theory:

Regression: It predicts the continuous output variables based on the independent input variable. like the prediction of house prices based on different parameters like house age, distance from the main road, location, area, etc. Linear regression is also a type of machine-learning algorithm more specifically a supervised machine-learning algorithm that learns from the labelled datasets and maps the data points to the most optimized linear functions. which can be used for prediction on new datasets. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc. Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable changes according to the value of the independent variable.

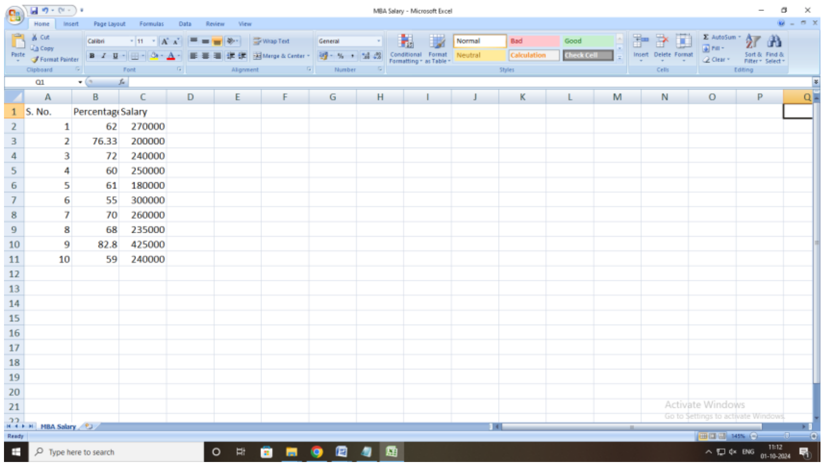
Program:

```
from google.colab import files
uploaded=files.upload()

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import io
df=pd.read_csv(io.BytesIO(uploaded['MBA Salary.csv']))
```

```
print(df)

x=df.iloc[:,-2]
x=x.values
x=x.reshape(-1,1)
print(x)
y=df.iloc[:,-1]
y=y.values
y=y.reshape(-1,1)
print(y)

plt.scatter(x,y)

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.05)
x_train=x_train/max(x_train)
y_train=y_train/max(y_train)
print(y_train)
x_test=x_test/max(x_test)
y_test=y_test/max(y_test)
print(y_test)

from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
print('model intercept:',model.intercept_)
print('model coefficients',model.coef_)
plt.scatter(x_train, y_train)
plt.plot(x_train, model.predict(x_train))
y_pred=model.predict(x_test)
print(y_pred)
```

```
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test,y_pred))
plt.scatter(x_test,y_test)
plt.plot(x_test,y_pred)
```

Data Set:

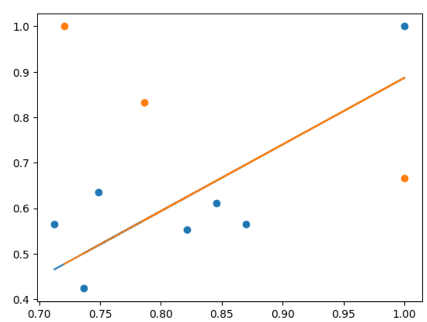


S. No.	Percentag	Salary
1	62	270000
2	76.33	200000
3	72	240000
4	60	250000
5	61	180000
6	55	300000
7	70	260000
8	68	235000
9	82.8	425000
10	59	240000

Expected Output:

model intercept: [-0.57941749]

model coefficients [[1.46641514]]



Result: Hence, the model intercept and model coefficients are calculated for a given data set by using linear regression algorithm

7. Logistic Regression Algorithm

Aim: To write a Python program for the implementation of the Logistic Regression Algorithm for the given data set and to find its accuracy.

Software Required:

Google Colab

Theory:

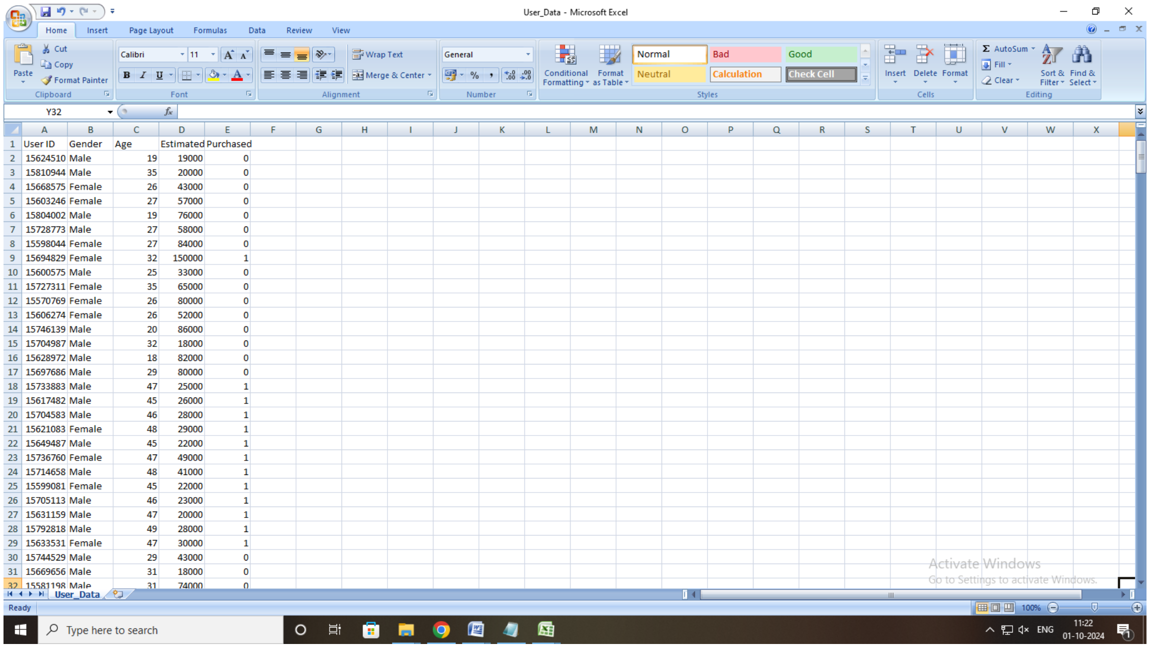
Logistic regression is a supervised machine learning algorithm used for classification tasks where the goal is to predict the probability that an instance belongs to a given class or not. Logistic regression is a statistical algorithm which analyze the relationship between two data factors.For example, we have two classes Class 0 and Class 1 if the value of the logistic function for an input is greater than 0.5 (threshold value) then it belongs to Class 1 otherwise it belongs to Class 0. It's referred to as regression because it is the extension of linear regression but is mainly used for classification problems.

Program:

```
from google.colab import files
uploaded=files.upload()

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import io
import seaborn as sns
df=pd.re

ad_csv(io.BytesIO(uploaded['User_Data.csv']))
print(df)
```

```
X = df.iloc[:, [2,3]].values
Y = df.iloc[:, 4].values
X
Y

# 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, rando
m_state = None)

# 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)

Y_Pred = classifier.predict(X_Test)
Y_Pred

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_Test, Y_Pred)
cm

# Heatmap of Confusion matrix
sns.heatmap(pd.DataFrame(cm), annot=True)
from sklearn.metrics import accuracy_score
accuracy =accuracy_score(Y_Test, Y_Pred)
accuracy
```

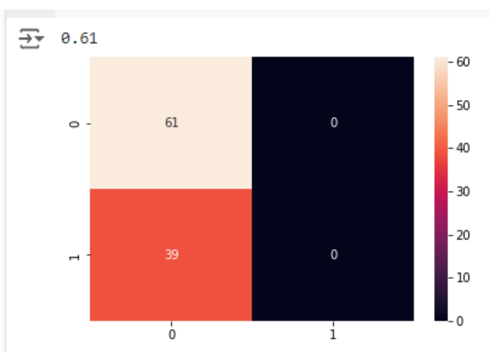
Data Set:



User ID	Gender	Age	Estimated	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1
15714658	Male	48	41000	1
15599081	Female	45	22000	1
15705113	Male	46	23000	1
15631159	Male	47	20000	1
15792818	Male	49	28000	1
15633531	Female	47	30000	1
15744529	Male	29	43000	0
15669656	Male	31	18000	0
15581198	Male	31	74000	0

Expected Output:

Accuracy: 0.61



Result: Hence, the Accuracy and Confusion matrix is calculated for a given data set using logistic regression algorithm

8. Decision Tree Algorithm

Aim: To write a Python program for the implementation of the Decision Tree Algorithm for the given data set and to find its accuracy and confusion matrix.

Software Required:

Google Colab

Theory:

A decision tree is a flowchart-like structure used to make decisions or predictions. It consists of nodes representing decisions or tests on attributes, branches representing the outcome of these decisions, and leaf nodes representing final outcomes or predictions. Each internal node corresponds to a test on an attribute, each branch corresponds to the result of the test, and each leaf node corresponds to a class label or a continuous value.

The process of creating a decision tree involves:

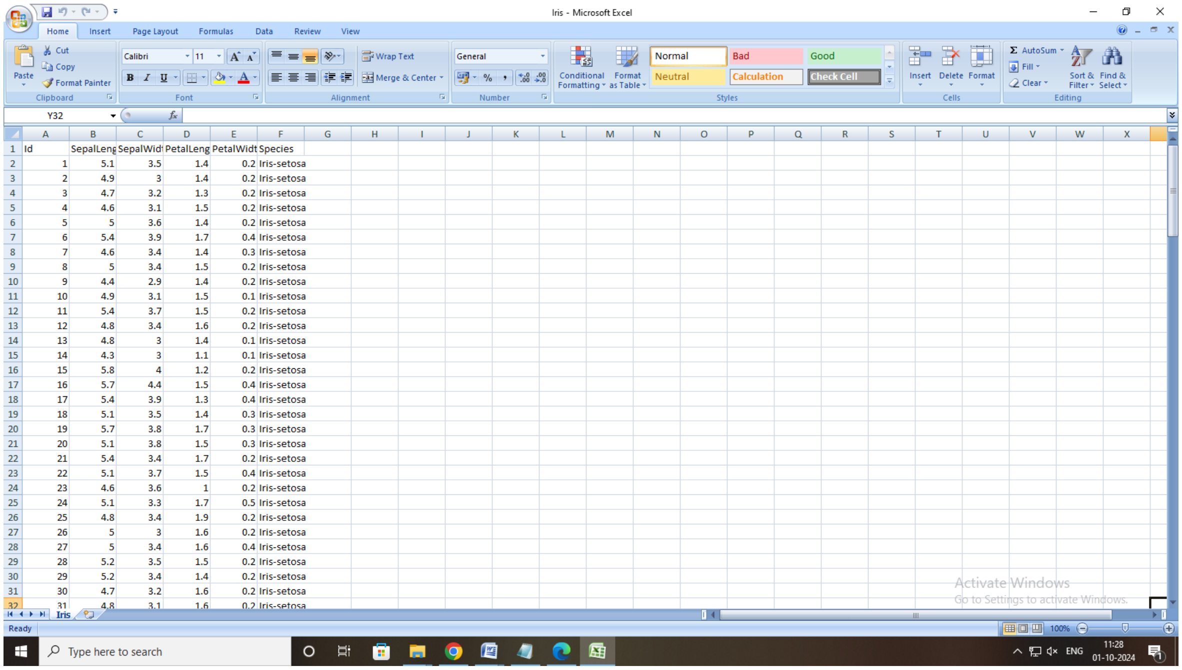
1. Selecting the Best Attribute: Using a metric like Gini impurity, entropy, or information gain, the best attribute to split the data is selected.
2. Splitting the Dataset: The dataset is split into subsets based on the selected attribute.
3. Repeating the Process: The process is repeated recursively for each subset, creating a new internal node or leaf node until a stopping criterion is met (e.g., all instances in a node belong to the same class or a predefined depth is reached).

   Unsupervised Machine Learning is the process of teaching a computer to use unlabeled, unclassified data and enabling the algorithm to operate on that data without supervision. Without any previous data training, the machine's job in this case is to organize unsorted data according to parallels, patterns, and variations.

Program:

```
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
import io
data=pd.read_csv(io.BytesIO(uploaded['Iris.csv']))
print(data)

x=data.values[:,1:5]
y=data.values[:,-1]
y=y.reshape(-1,1)
#print(y)
#print(x)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
#Perform training with GiniIndex
clf_gini=DecisionTreeClassifier(criterion='gini',rand
om_state=100,max_depth=3)
clf_gini.fit(x_train,y_train)

#Perform training with Entropy
clf_entropy=Decision Tree Classifier (criterion='entropy', random_state=100,
max_depth=3)
clf_entropy.fit(x_train,y_train)
y_pred=clf_entropy.predict(x_test)
print("Confusion Matrix:",confusion_matrix(y_test, y_pred))
print ("Accuracy :",accuracy_score(y_test,y_pred)*100)
```

Data Set:



Id	SepalLeng	SepalWidt	PetalLeng	PetalWidt	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa
13	4.8	3	1.4	0.1	Iris-setosa
14	4.3	3	1.1	0.1	Iris-setosa
15	5.8	4	1.2	0.2	Iris-setosa
16	5.7	4.4	1.5	0.4	Iris-setosa
17	5.4	3.9	1.3	0.4	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa
19	5.7	3.8	1.7	0.3	Iris-setosa
20	5.1	3.8	1.5	0.3	Iris-setosa
21	5.4	3.4	1.7	0.2	Iris-setosa
22	5.1	3.7	1.5	0.4	Iris-setosa
23	4.6	3.6	1	0.2	Iris-setosa
24	5.1	3.3	1.7	0.5	Iris-setosa
25	4.8	3.4	1.9	0.2	Iris-setosa
26	5	3	1.6	0.2	Iris-setosa
27	5	3.4	1.6	0.4	Iris-setosa
28	5.2	3.5	1.5	0.2	Iris-setosa
29	5.2	3.4	1.4	0.2	Iris-setosa
30	4.7	3.2	1.6	0.2	Iris-setosa
31	4.8	3.1	1.6	0.2	Iris-setosa

Expected Output:

Confusion Matrix: [[20 0 0]
[0 8 0]
[0 1 16]]

Accuracy: 97.77777777777777

Result: Hence, the Accuracy and Confusion matrix is calculated for a given data set using decision tree algorithm

9. K-Means Algorithm

Aim: To write a Python program for the implementation of the K-Means Algorithm to classify the given data set.

Software Required:

Google Colab

Theory:

Kmeans clustering, assigns data points to one of the K clusters depending on their distance from the center of the clusters. It starts by randomly assigning the clusters centroid in the space. Then each data point assign to one of the cluster based on its distance from centroid of the cluster. After assigning each point to one of the cluster, new cluster centroids are assigned. This process runs iteratively until it finds good cluster. In the analysis we assume that number of cluster is given in advanced and we have to put points in one of the group.

In some cases, K is not clearly defined, and we have to think about the optimal number of K. K Means clustering performs best data is well separated. When data points overlapped this clustering is not suitable. K Means is faster as compare to other clustering technique. It provides strong coupling between the data points. K Means cluster do not provide clear information regarding the quality of clusters. Different initial assignment of cluster centroid may lead to different clusters. Also, K Means algorithm is sensitive to noise. It may have stuck in local minima.

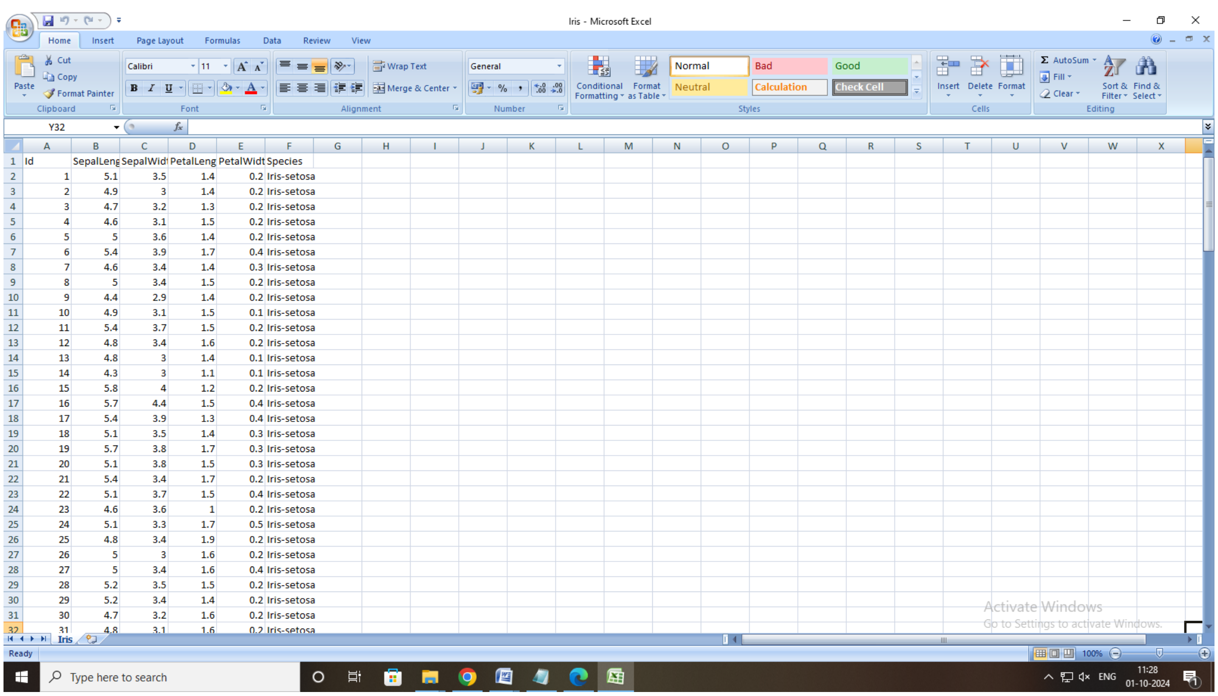
Program:

```
from google.colab import files
uploaded=files.upload()
import numpy as nm
```

```
import matplotlib.pyplot as mtp
import pandas as pd
import io
dataset = pd.read_csv(io.BytesIO (uploaded['Iris.csv']))
print(dataset)

x = dataset.iloc[:, [3, 4]].values

#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= [] #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
 kmeans = KMeans(n_clusters=i, init='k  means++', random_state= 42)
 kmeans.fit(x)
 wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()

#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)

#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c =
'blue', label = 'Cluster 1') #for first cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c =
```

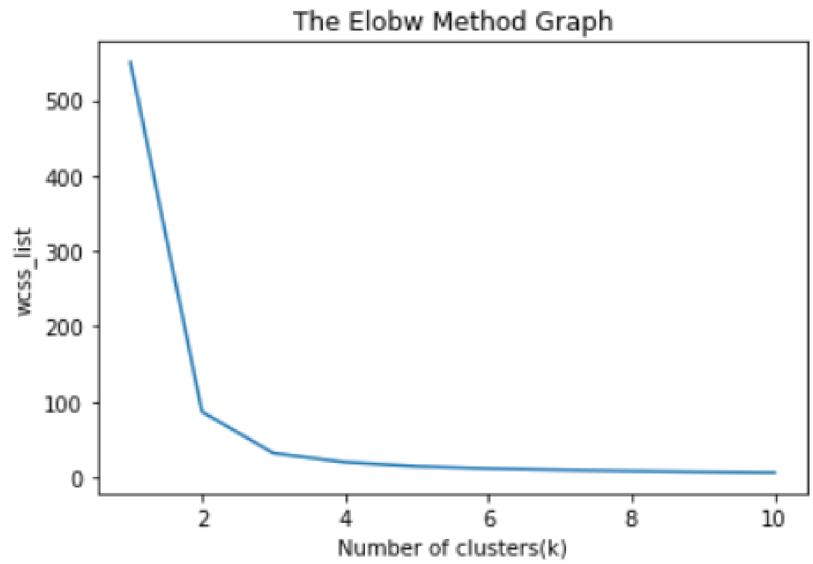
```
'green', label = 'Cluster 2') #for second cluster
mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c =
'red', label = 'Cluster 3') #for third cluster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c =
'cyan', label = 'Cluster 4') #for fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c =
'magenta', label = 'Cluster 5') #for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s =
300, c = 'yellow', label ='Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('PETAL LENGTH')
mtp.ylabel('PETAL WIDTH')
mtp.legend()
mtp.show()
```

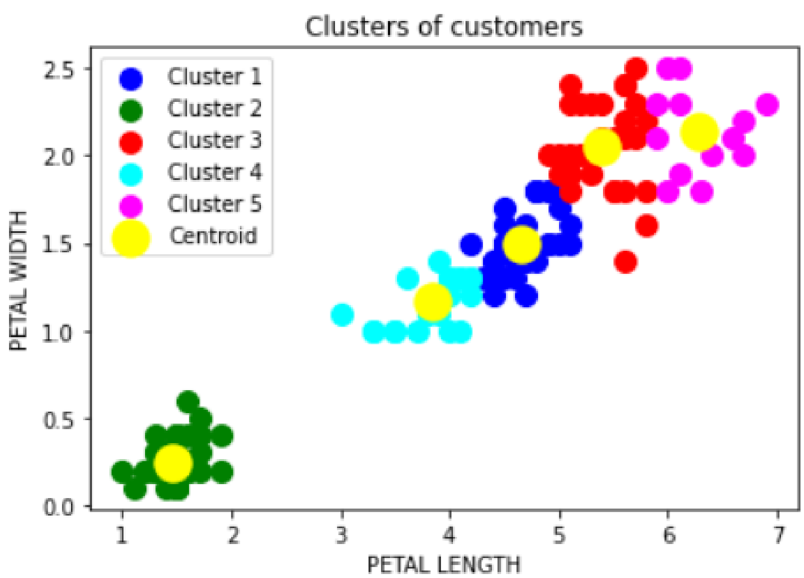
Data Set:



Id	SepalLeng	SepalWidt	PetalLeng	PetalWidt	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa
13	4.8	3	1.4	0.1	Iris-setosa
14	4.3	3	1.1	0.1	Iris-setosa
15	5.8	4	1.2	0.2	Iris-setosa
16	5.7	4.4	1.5	0.4	Iris-setosa
17	5.4	3.9	1.3	0.4	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa
19	5.7	3.8	1.7	0.3	Iris-setosa
20	5.1	3.8	1.5	0.3	Iris-setosa
21	5.4	3.4	1.7	0.2	Iris-setosa
22	5.1	3.7	1.5	0.4	Iris-setosa
23	4.6	3.6	1	0.2	Iris-setosa
24	5.1	3.3	1.7	0.5	Iris-setosa
25	4.8	3.4	1.9	0.2	Iris-setosa
26	5	3	1.6	0.2	Iris-setosa
27	5	3.4	1.6	0.4	Iris-setosa
28	5.2	3.5	1.5	0.2	Iris-setosa
29	5.2	3.4	1.4	0.2	Iris-setosa
30	4.7	3.2	1.6	0.2	Iris-setosa
31	4.8	3.1	1.6	0.2	Iris-setosa

Expected Output:





Result: Hence the data is classified for a given data set by using K-Means Algorithm.

10. Support Vector Machine

Aim: To write a Python program for the implementation of the Support Vector Machine for the given data set and to find its accuracy.

Software Required:

Google Colab

Theory:

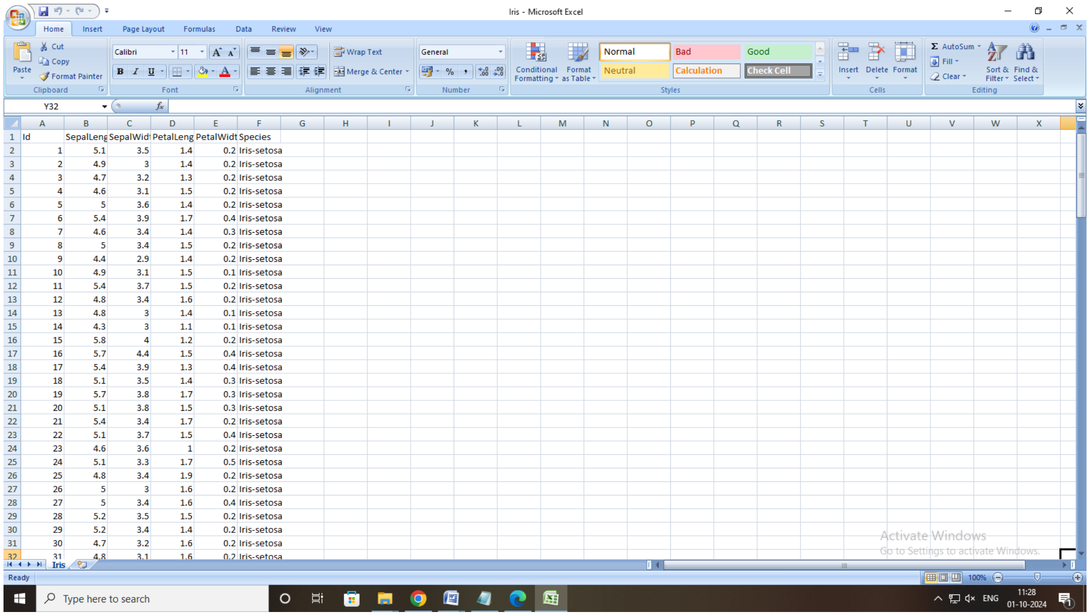
Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well it's best suited for classification. The main objective of the SVM algorithm is to find the optimal hyperplane in an N-dimensional space that can separate the data points in different classes in the feature space. The hyperplane tries that the margin between the closest points of different classes should be as maximum as possible. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from google.colab import files
uploaded=files.upload()
data=pd.read_csv('Iris.csv')
data.head()


#Encoding the categorical column
```

```
data=data.replace({"Species": {"Iris-setosa":1,"Iris  versicolor":2,"Iris-virginica":3}})

#Visualize the new dataset
data.head()

#plt.figure(1)
sns.heatmap(data.corr())
plt.title('Correlation On iris Classes')
x = data.iloc[:,:-1]
y = data.iloc[:, -1].values
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)

#Create the SVM model
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)

#Fit the model for the data
classifier.fit(x_train, y_train)

#Make the prediction
y_pred = classifier.predict(x_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
from sklearn.metrics import accuracy_score
print ("Accuracy:",accuracy_score(y_test,y_pred)*100
```

Data Set:

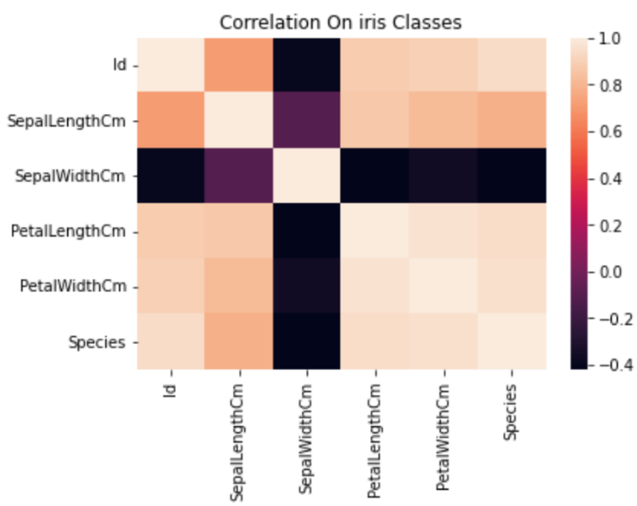


	A	B	C	D	E	F
1	Id	SepalLeng	SepalWidt	PetalLeng	PetalWidt	Species
2	1	5.1	3.5	1.4	0.2	Iris-setosa
3	2	4.9	3	1.4	0.2	Iris-setosa
4	3	4.7	3.2	1.3	0.2	Iris-setosa
5	4	4.6	3.1	1.5	0.2	Iris-setosa
6	5	5	3.6	1.4	0.2	Iris-setosa
7	6	5.4	3.9	1.7	0.4	Iris-setosa
8	7	4.6	3.4	1.4	0.3	Iris-setosa
9	8	5	3.4	1.5	0.2	Iris-setosa
10	9	4.4	2.9	1.4	0.2	Iris-setosa
11	10	4.9	3.1	1.5	0.1	Iris-setosa
12	11	5.4	3.7	1.5	0.2	Iris-setosa
13	12	4.8	3.4	1.6	0.2	Iris-setosa
14	13	4.8	3	1.4	0.1	Iris-setosa
15	14	4.3	3	1.1	0.1	Iris-setosa
16	15	5.8	4	1.2	0.2	Iris-setosa
17	16	5.7	4.4	1.5	0.4	Iris-setosa
18	17	5.4	3.9	1.3	0.4	Iris-setosa
19	18	5.1	3.5	1.4	0.3	Iris-setosa
20	19	5.7	3.8	1.7	0.3	Iris-setosa
21	20	5.1	3.8	1.5	0.3	Iris-setosa
22	21	5.4	3.4	1.7	0.2	Iris-setosa
23	22	5.1	3.7	1.5	0.4	Iris-setosa
24	23	4.6	3.6	1	0.2	Iris-setosa
25	24	5.1	3.3	1.7	0.5	Iris-setosa
26	25	4.8	3.4	1.9	0.2	Iris-setosa
27	26	5	3	1.6	0.2	Iris-setosa
28	27	5	3.4	1.6	0.4	Iris-setosa
29	28	5.2	3.5	1.5	0.2	Iris-setosa
30	29	5.2	3.4	1.4	0.2	Iris-setosa
31	30	4.7	3.2	1.6	0.2	Iris-setosa
32	31	4.8	3.1	1.6	0.2	Iris-setosa

Expected Output:

[[13 0 0]
[0 16 0]
[0 0 9]]

Accuracy : 100.0



Result: Hence, the accuracy is calculated for given data set for support vector machine.

11. Hierarchical Cluster Analysis (HCA)

Aim: To write a Python program for the implementation of the Hierarchical Cluster Analysis (HCA) on a dataset and visualise the resulting dendrogram.

Software Required:

Google Colab

Theory:

Hierarchical cluster analysis is an algorithm that groups similar objects into groups called clusters. The endpoint is a set of clusters, where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other.

Hierarchical clustering is an unsupervised learning method for clustering data points. The algorithm builds clusters by measuring the dissimilarities between data. Unsupervised learning means that a model does not have to be trained, and we do not need a "target" variable. This method can be used on any data to visualize and interpret the relationship between individual data points.

Program:

```
import pandas as pd
import numpy as np
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import files
uploaded = files.upload()  # Use this to upload your CSV file

# Assuming the uploaded file is 'data.csv'
```

```
df = pd.read_csv('data.csv')
df.head()  # Display the first few rows of the dataset
# Drop rows with missing values (if any)
df = df.dropna()

# Select numerical features for clustering
X = df.select_dtypes(include=[np.number])

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Create linkage matrix
Z = linkage(X_scaled, method='ward')
plt.figure(figsize=(10, 7))
dendrogram(Z)
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()
from scipy.cluster.hierarchy import cut_tree
clusters = cut_tree(Z, n_clusters=3)  # Example: 3 clusters
df['Cluster'] = clusters
sns.scatterplot(x=X_scaled[:, 0], y=X_scaled[:, 1], hue=df['Cluster'])
plt.title('Clusters Visualization')
plt.show()
```

Result:

Hence, Summarize the findings of the hierarchical cluster analysis, including the number of clusters, the characteristics of each cluster, and any insights gained from the dendrogram.

12. Principal Component Analysis (PCA)

Aim: To write a Python program for the implementation of the Principal Component Analysis for the given data set and to find its Final Principal component.

Software Required:

Google Colab

Theory:

As the number of features or dimensions in a dataset increases, the amount of data required to obtain a statistically significant result increases exponentially. This can lead to issues such as overfitting, increased computation time, and reduced accuracy of machine learning models this is known as the curse of dimensionality problems that arise while working with high-dimensional data.

As the number of dimensions increases, the number of possible combinations of features increases exponentially, which makes it computationally difficult to obtain a representative sample of the data. It becomes expensive to perform tasks such as clustering or classification because the algorithms need to process a much larger feature space, which increases computation time and complexity. Additionally, some machine learning algorithms can be sensitive to the number of dimensions, requiring more data to achieve the same level of accuracy as lower-dimensional data.

To address the curse of dimensionality, Feature engineering techniques are used which include feature selection and feature extraction. Dimensionality reduction is a type of feature extraction technique that aims to reduce the number of input features while retaining as much of the original information as possible.

In this article, we will discuss one of the most popular dimensionality reduction techniques i.e. Principal Component Analysis(PCA).

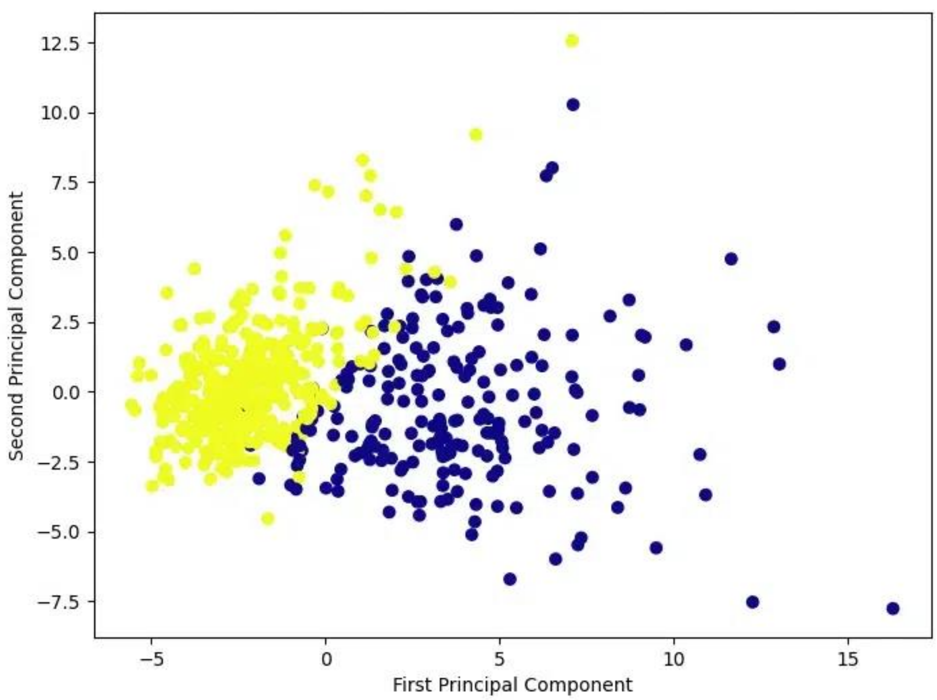
Program:

```
# Importing PCA
from sklearn.decomposition import PCA

# Let's say, components = 2
pca = PCA(n_components=2)
pca.fit(Z)
x_pca = pca.transform(Z)

# Create the dataframe
df_pca1 = pd.DataFrame(x_pca,
                      columns=['PC{}'.
                      format(i+1)
                       for i in range(n_components)])
print(df_pca1)

# giving a larger plot
plt.figure(figsize=(8, 6))

plt.scatter(x_pca[:, 0], x_pca[:, 1],
            c=cancer['target'],
            cmap='plasma')

# labeling x and y axes
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()
```

Data Set:

```
          PC1        PC2
0     9.184755   1.946870
1     2.385703  -3.764859
2     5.728855  -1.074229
3     7.116691  10.266556
4     3.931842  -1.946359
..         ...        ...
564   6.433655  -3.573673
565   3.790048  -3.580897
566   1.255075  -1.900624
567  10.365673   1.670540
568  -5.470430  -0.670047
[569 rows x 2 columns]
```

Expected Output:



Result: Hence, the Final Principal component is calculated for given data set for Principal Component Analysis.

13.Kernal Principal Component Analysis (K-PCA)

Aim: To write a Python program for the implementation of the Kernal Principal Component Analysis for the given data set and apply the kernel PCA to a non-linear dataset using scikit-learn

Software Required:

Google Colab

Theory:

is a tool which is used to reduce the dimension of the data. It allows us to reduce the dimension of the data without much loss of information. PCA reduces the dimension by finding a few orthogonal linear combinations (principal components) of the original variables with the largest variance. The first principal component captures most of the variance in the data. The second principal component is orthogonal to the first principal component and captures the remaining variance, which is left of first principal component and so on. There are as many principal components as the number of original variables. These principal components are uncorrelated and are ordered in such a way that the first several principal components explain most of the variance of the original data. To learn more about PCA you can read the article Principal Component Analysis

PCA is a linear method. That is it can only be applied to datasets which are linearly separable. It does an excellent job for datasets, which are linearly separable. But, if we use it to non-linear datasets, we might get a result which may not be the optimal dimensionality reduction. Kernel PCA uses a kernel function to project dataset into a higher dimensional feature space, where it is linearly separable. It is similar to the idea of Support Vector Machines. There are various kernel methods like linear, polynomial, and gaussian.

Kernel Principal Component Analysis (KPCA) is a technique used in machine learning for nonlinear dimensionality reduction. It is an extension of the classical Principal Component Analysis (PCA) algorithm, which is a linear method that identifies the most significant features or components of a dataset.

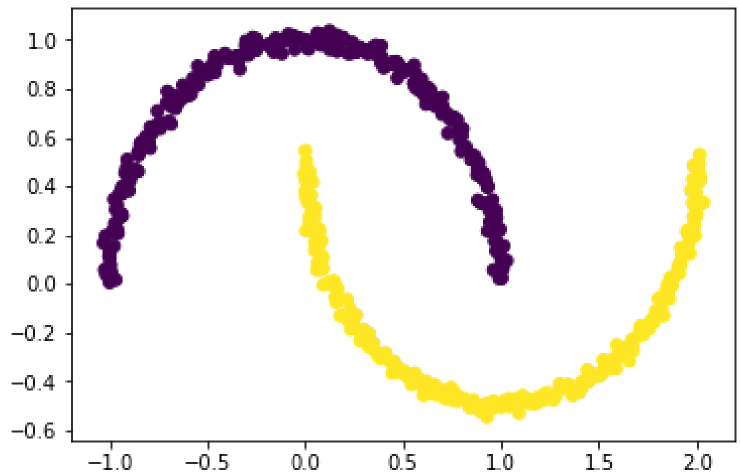
KPCA applies a nonlinear mapping function to the data before applying PCA, allowing it to capture more complex and nonlinear relationships between the data points.

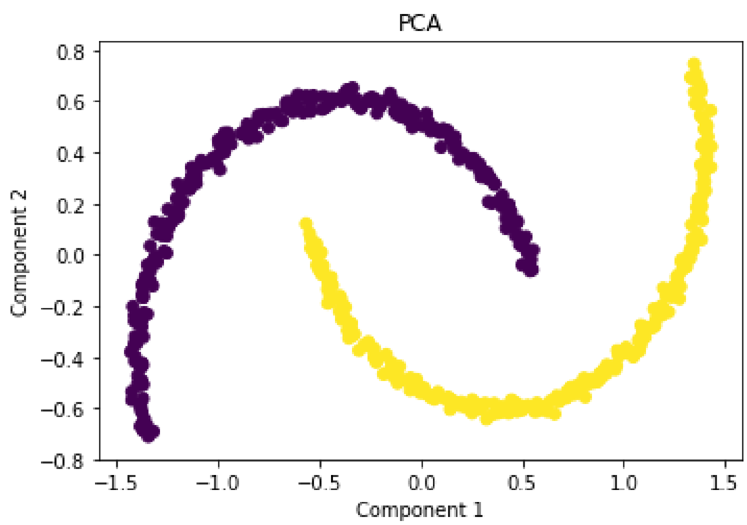
Program:

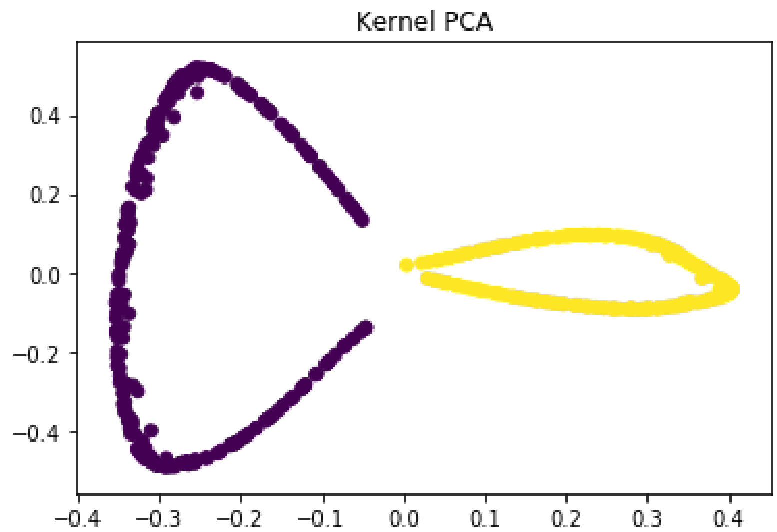
```
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=500, noise=0.02, random_state=417)

plt.scatter(X[:, 0], X[:, 1], c=y)
plt.show()

#taking pca
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.title("PCA")
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y)
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.show()


#As you can see PCA failed to distinguish the two classes.
#Applying kernel PCA on this dataset with RBF kernel with a gamma value of 15.
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel='rbf', gamma=15)
X_kpca = kpca.fit_transform(X)
```

```
plt.title("Kernel PCA")
plt.scatter(X_kpca[:, 0], X_kpca[:, 1], c=y)
plt.show()
```

Expected Output:







Results: Hence, successfully applied the kernel PCA to a non-linear dataset using scikit-learn for the given data set.

14. Q-Learning Algorithm

Aim: To write a Python program for the implementation of the Q-Learning Algorithm for the given data set and calculate the Learned Q-table.

Software Required:

Google Colab

Theory: Q-learning is a machine learning approach that enables a model to iteratively learn and improve over time by taking the correct action. Q-learning is a type of reinforcement learning.

With reinforcement learning, a machine learning model is trained to mimic the way animals or children learn. Good actions are rewarded or reinforced, while bad actions are discouraged and penalized.

With the state-action-reward-state-action form of reinforcement learning, the training regimen follows a model to take the right actions. Q-learning provides a model-free approach to reinforcement learning. There is no model of the environment to guide the reinforcement learning process. The agent -- which is the AI component that acts in the environment -- iteratively learns and makes predictions about the environment on its own.

Q-learning also takes an off-policy approach to reinforcement learning. A Q-learning approach aims to determine the optimal action based on its current state. The Q-learning approach can accomplish this by either developing its own set of rules or deviating from the prescribed policy. Because Q-learning may deviate from the given policy, a defined policy is not needed.

Program:

```
import numpy as np
# Define the environment
n_states = 16  # Number of states in the grid world
n_actions = 4  # Number of possible actions (up, down, left, right)
goal_state = 15  # Goal state
```

```
# Initialize Q-table with zeros
Q_table = np.zeros((n_states, n_actions))

# Define parameters
learning_rate = 0.8
discount_factor = 0.95
exploration_prob = 0.2
epochs = 1000

# Q-learning algorithm
for epoch in range(epochs):
    current_state = np.random.randint(0, n_states)  # Start from a random state

    while current_state != goal_state:
        # Choose action with epsilon-greedy strategy
        if np.random.rand() < exploration_prob:
            action = np.random.randint(0, n_actions)  # Explore
        else:
            action = np.argmax(Q_table[current_state])  # Exploit

        # Simulate the environment (move to the next state)
        # For simplicity, move to the next state
        next_state = (current_state + 1) % n_states

        # Define a simple reward function (1 if the goal state is reached, 0
otherwise)
        reward = 1 if next_state == goal_state else 0
        # Update Q-value using the Q-learning update rule
        Q_table[current_state, action] += learning_rate * \
            (reward + discount_factor *
```

```
            np.max(Q_table[next_state]) - Q_table[current_state, action])
        current_state = next_state  # Move to the next state
# After training, the Q-table represents the learned Q-values
print("Learned Q-table:")
print(Q_table)
```

Expected Output:

```
Learned Q-table:
[[0.48767498 0.48377358 0.48751874 0.48377357]
 [0.51252074 0.51317781 0.51334071 0.51334208]
 [0.54036009 0.5403255  0.54018713 0.54036009]
 [0.56880009 0.56880009 0.56880008 0.56880009]
 [0.59873694 0.59873694 0.59873694 0.59873694]
 [0.63024941 0.63024941 0.63024941 0.63024941]
 [0.66342043 0.66342043 0.66342043 0.66342043]
 [0.6983373  0.6983373  0.6983373  0.6983373 ]
 [0.73509189 0.73509189 0.73509189 0.73509189]
 [0.77378094 0.77378094 0.77378094 0.77378094]
 [0.81450625 0.81450625 0.81450625 0.81450625]
 [0.857375   0.857375   0.857375   0.857375  ]
 [0.9025     0.9025     0.9025     0.9025    ]
 [0.95       0.95       0.95       0.95      ]
 [1.         1.         1.         1.        ]
 [0.         0.         0.         0.        ]]
```

Results: Hence, successfully the implemented the Q-Learning Algorithm for the given data set and calculated its Learned Q-table.

REFERENCES

1. Applied Machine Learning, M.Gopal, McGraw Hill Education, 1st Edition, 2018, ISBN-13:978-93-5316-025-8.
2. Machine Learning by Tom Mitchell, McGraw Hill1997, 1st edition
3. Pattern Recognition and Machine Learning by Bishop, 2006 1st Edition, ISBN: 978-0-387-31073-2
4. Machine Learning For Absolute Beginners: A Plain English Introduction (Second Edition) by Oliver Theobald.
5. Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies by John D. Kelleher, Brian Mac Namee, and Aoife D'Arcy
6. Machine Learning For Dummies by John Paul Mueller and Luca Massaron
7. Machine Learning for Hackers by Drew Conway and John Myles White
8. Machine Learning in Action by Peter Harrington
9. https://github.com/
10. https://www.geeksforgeeks.org/
11. https://www.kaggle.com/