Name : Suniksha Ben Patel
Reg No : 21BCE10497

Lab Assessment

Experiment -1

Aim : To 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.

Algorithm :

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.

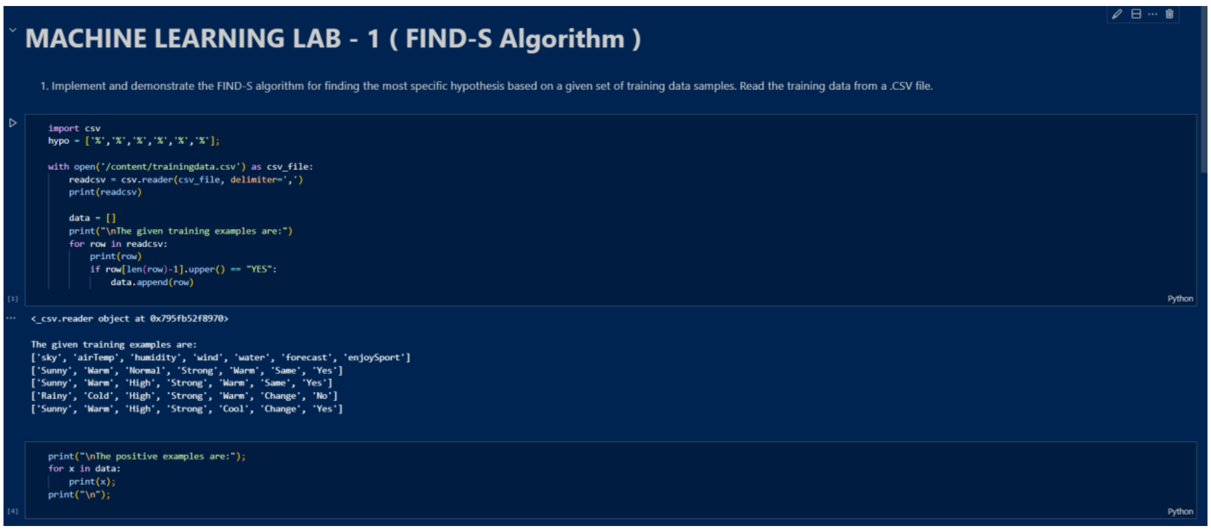
1. Start with the most specific hypothesis.
   h = {ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}
2. Take the next example and if it is negative, then no changes occur to the hypothesis.
3. If the example is positive and we find that our initial hypothesis is too specific then we update our current hypothesis to a general condition.
4. Keep repeating the above steps till all the training examples are complete.
5. After we have completed all the training examples we will have the final hypothesis when can use to classify the new examples.

Code :

```
 #importing

import csv

hypo = ['%','%','%','%','%','%'];


with open('trainingdata.csv') as csv_file:

   readcsv = csv.reader(csv_file, delimiter=',')

   print(readcsv)


   data = []

   print("\nThe given training examples are:")
```

```
    for row in readcsv:
        print(row)
        if row[len(row)-1].upper() == "YES":
            data.append(row)

#printting
print("\nThe positive examples are:");
for x in data:
    print(x);
print("\n");

#finding Hypothesis
TotalExamples = len(data);
i=0;
j=0;
k=0;
print("The steps of the Find-s algorithm are :\n",hypo);
list = [];
p=0;
d=len(data[p])-1;
for j in range(d):
    list.append(data[i][j]);
hypo=list;
i=1;
for i in range(TotalExamples):
    for k in range(d):
        if hypo[k]!=data[i][k]:
            hypo[k]='?';
            k=k+1;
        else:
            hypo[k];
    print(hypo);
i=i+1;
#printing the final Hyposthesis
print("\nThe maximally specific Find-s hypothesis for the given training examples is :");
list=[];
for i in range(d):
    list.append(hypo[i]);
print(list);
```

Output :



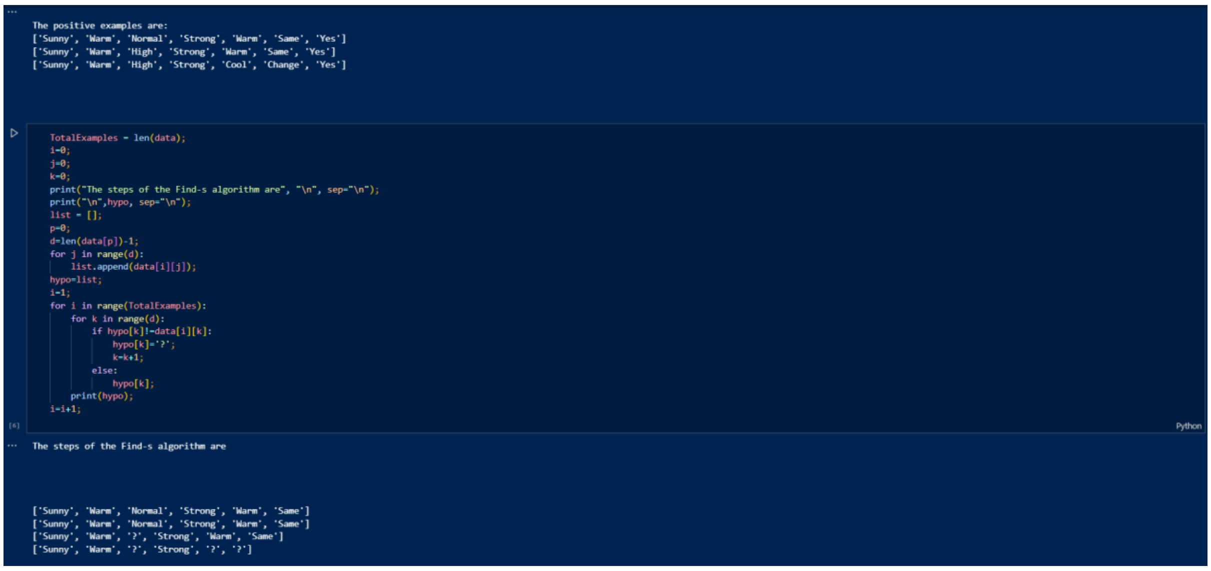
MACHINE LEARNING LAB - 1 (FIND-S Algorithm)

1. 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.

```
import csv
hypo = ['%','%','%','%','%','%'];

with open('/content/trainingdata.csv') as csv_file:
    readcsv = csv.reader(csv_file, delimiter=',')
    print(readcsv)

    data = []
    print("\nThe given training examples are:")
    for row in readcsv:
        print(row)
        if row[len(row)-1].upper() == "YES":
            data.append(row)
```

```
<_csv.reader object at 0x795fb52f8970>

The given training examples are:
['sky', 'airTemp', 'humidity', 'wind', 'water', 'forecast', 'enjoySport']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
```

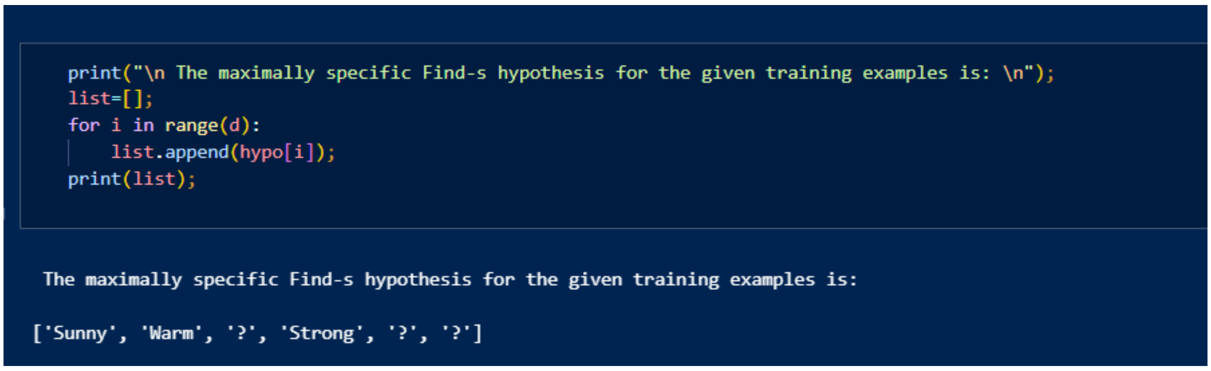
```
print("\nThe positive examples are:");
for x in data:
    print(x);
print("\n");
```



```
The positive examples are:
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
```

```
TotalExamples = len(data);
i=0;
j=0;
k=0;
print("The steps of the Find-s algorithm are", "\n", sep="\n");
print("\n",hypo, sep="\n");
list = [];
p=0;
d=len(data[p])-1;
for j in range(d):
    list.append(data[i][j]);
hypo=list;
i=1;
for i in range(TotalExamples):
    for k in range(d):
        if hypo[k]!=data[i][k]:
            hypo[k]='?';
            k=k+1;
        else:
            hypo[k];
    print(hypo);
i=i+1;
```

```
The steps of the Find-s algorithm are



['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```



```
print("\n The maximally specific Find-s hypothesis for the given training examples is: \n");
list=[];
for i in range(d):
    list.append(hypo[i]);
print(list);
```

```
 The maximally specific Find-s hypothesis for the given training examples is:

['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Conclusion :

In this experiment, we successfully implemented the Find-S algorithm to identify the most specific hypothesis that fits all positive training examples. By iteratively updating the hypothesis to generalize only when necessary, the algorithm efficiently converges on a specific set of conditions that describe the positive examples. This process highlights the foundational principles of machine learning, particularly in concept learning and hypothesis formation. The final hypothesis obtained can be used to classify new instances, demonstrating the practical application of the Find-S algorithm in understanding and categorizing data.

Experiment -2

Aim : To implement 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.

Algorithm :

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.

- You can consider this as an extended form of the Find-S algorithm.
- Consider both positive and negative examples.
- Actually, positive examples are used here as the Find-S algorithm (Basically they are generalizing from the specification).
- While the negative example is specified in the generalizing form.

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)
```

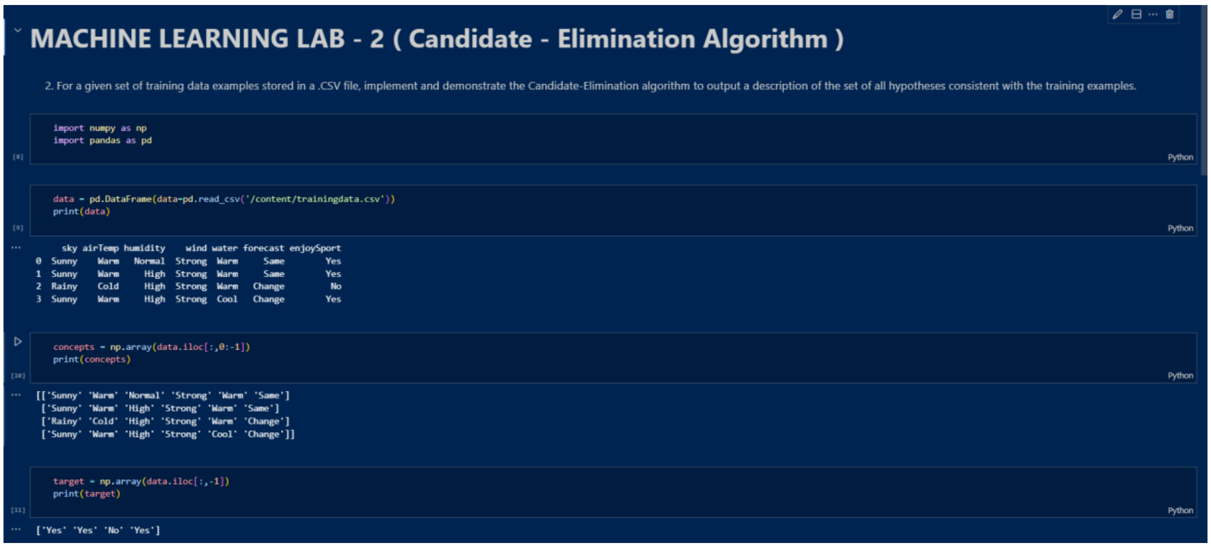
Step5: If example is Negative example

Make generalize hypothesis more specific.

Code :

```
#importing libraries
import numpy as np
import pandas as pd

#loading dataset
data = pd.DataFrame(data=pd.read_csv('/content/trainingdata.csv'))
print(data)

#seperating concept feature from target
concepts = np.array(data.iloc[:,0:-1])
print(concepts)

#  Isolating target into a separate DataFrame
# copying last column to target array
target = np.array(data.iloc[:,-1])
print(target)

#initialising instance for concept
 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] = '?'

  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

#printing the final outcome
s_final,g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

Output :


MACHINE LEARNING LAB - 2 (Candidate - Elimination Algorithm)

2. 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.

```python
import numpy as np
import pandas as pd
```

```python
data = pd.DataFrame(data=pd.read_csv('/content/trainingdata.csv'))
print(data)
```

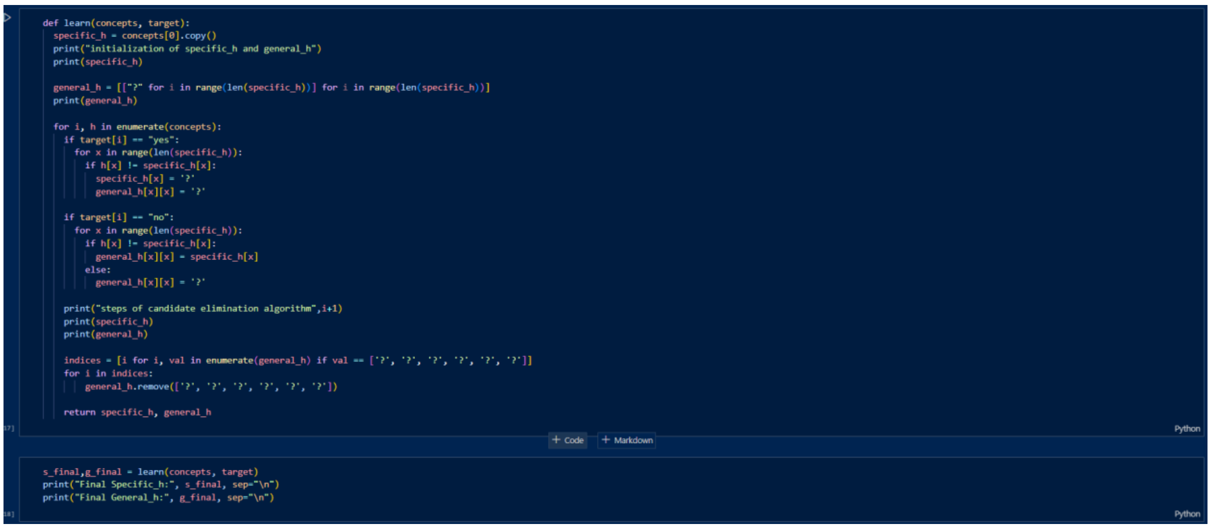
```
     sky airTemp humidity    wind water forecast enjoySport
0  Sunny    Warm   Normal  Strong  Warm     Same        Yes
1  Sunny    Warm     High  Strong  Warm     Same        Yes
2  Rainy    Cold     High  Strong  Warm   Change         No
3  Sunny    Warm     High  Strong  Cool   Change        Yes
```

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

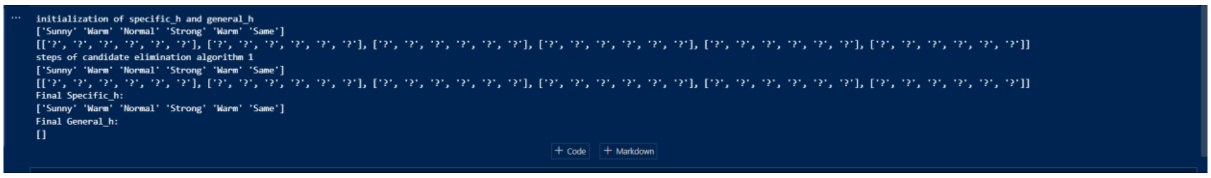
```
[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
```

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

```
['Yes' 'Yes' 'No' 'Yes']
```


```python
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] = '?'

    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
```

```python
s_final,g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```


```
initialization of specific_h and general_h
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
steps of candidate elimination algorithm 1
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Final Specific_h:
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Final General_h:
[]
```

Conclusion :

In this experiment, we successfully implemented the Candidate Elimination algorithm, which provides a systematic approach to identifying the set of all hypotheses consistent with a given set of training data. Unlike the Find-S algorithm, Candidate Elimination considers both positive and negative examples to refine the hypothesis space. The algorithm effectively narrows down the version space by updating the general and specific boundaries based on the examples encountered. This ensures that the final set of hypotheses is both as specific as necessary and as general as possible, thus encapsulating all consistent hypotheses. This experiment highlights the robustness of the Candidate Elimination algorithm in learning from data, even in the presence of conflicting examples.

Experiment - 3

Aim : To implement a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Algorithm :

The ID3 algorithm is a popular decision tree algorithm used in machine learning. It aims to build a decision tree by iteratively selecting the best attribute to split the data based on information gain. Each node represents a test on an attribute, and each branch represents a possible outcome of the test. The leaf nodes of the tree represent the final classifications

The ID3 algorithm works by building a decision tree, which is a hierarchical structure that classifies data points into different categories and splits the dataset into smaller subsets based on the values of the features in the dataset. The ID3 algorithm then selects the feature that provides the most information about the target variable. The decision tree is built top-down, starting with the root node, which represents the entire dataset. At each node, the ID3 algorithm selects the attribute that provides the most information gain about the target variable. The attribute with the highest information gain is the one that best separates the data points into different categories.

$H(S)=\Sigma-(Pi*log2(Pi))$

- where, $PiPi$ represents the fraction of the sample within a particular node.
- S – The current dataset.
- i – Set of classes in S

Steps fo ID3 :

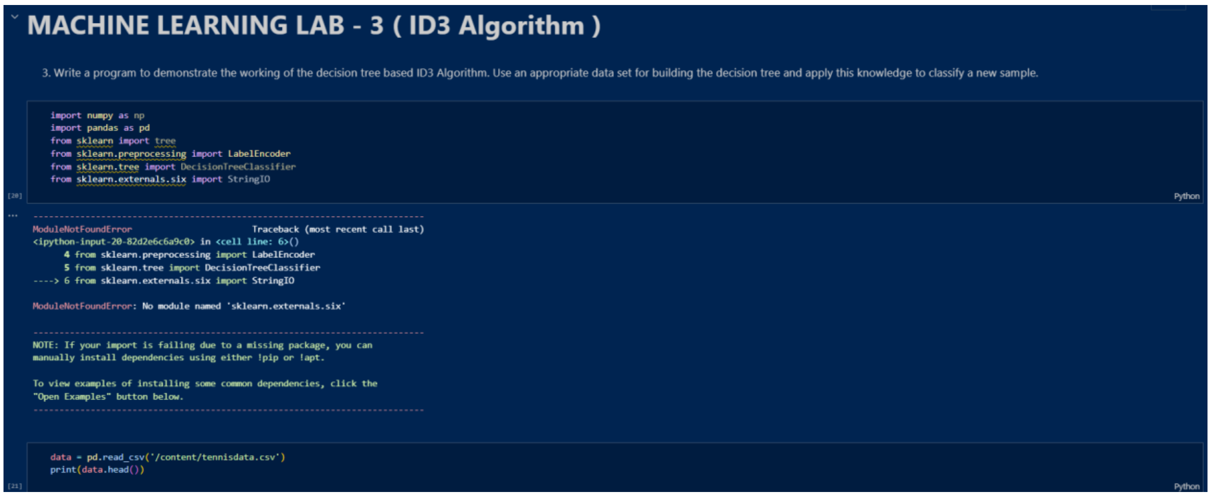
1. **Determine entropy** for the overall the dataset using class distribution.
2. For each feature.
    - Calculate **Entropy for Categorical Values**.
    - Assess **information gain** for each unique categorical value of the feature.
3. Choose the feature that generates **highest information gain**.
4. Iteratively apply all above steps to build the decision tree structure.

Code:

```
#importing Libraries

import numpy as np

import pandas as pd

from sklearn import tree
from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier


#reading the dataset

data = pd.read_csv('/content/tennisdata.csv')

print(data.head())


#splitting of X and Y

x = data.iloc[:,:-1]

print(x.head())

y = data.iloc[:,-1]

print(y.head())


#label encoding for converting them to numerical values

le_outlook = LabelEncoder()

x.Outlook = le_outlook.fit_transform(x.Outlook)


le_temperature = LabelEncoder()

x.Temperature = le_temperature.fit_transform(x.Temperature)


le_humidity = LabelEncoder()

x.Humidity = le_humidity.fit_transform(x.Humidity)


le_windy = LabelEncoder()

x.Windy = le_windy.fit_transform(x.Windy)


print(x.head())

le_PlayTennis = LabelEncoder()

y = le_PlayTennis.fit_transform(y)

print(y)
```

```
#training the Decsision tree classifier model with x and y and predicting the values
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

# Assuming the label encoders and classifier have already been defined and trained
classifier = DecisionTreeClassifier()
classifier.fit(x, y)

# Function to encode input
def labelEncoderForInput(list1):
    list1[0] = le_outlook.transform([list1[0]])[0]
    list1[1] = le_temperature.transform([list1[1]])[0]
    list1[2] = le_humidity.transform([list1[2]])[0]
    list1[3] = le_windy.transform([list1[3]])[0]
    return [list1]

# Predict for an input
inp = ["Rainy", "Mild", "High", "False"]
inp1 = ["Rainy", "Cool", "High", "False"]
pred1 = labelEncoderForInput(inp1)
y_pred = classifier.predict(pred1)
y_pred

# Print the result
print("\nFor input {0}, we obtain {1}".format(inp1, le_PlayTennis.inverse_transform([y_pred[0]])[0]))
```

Output:



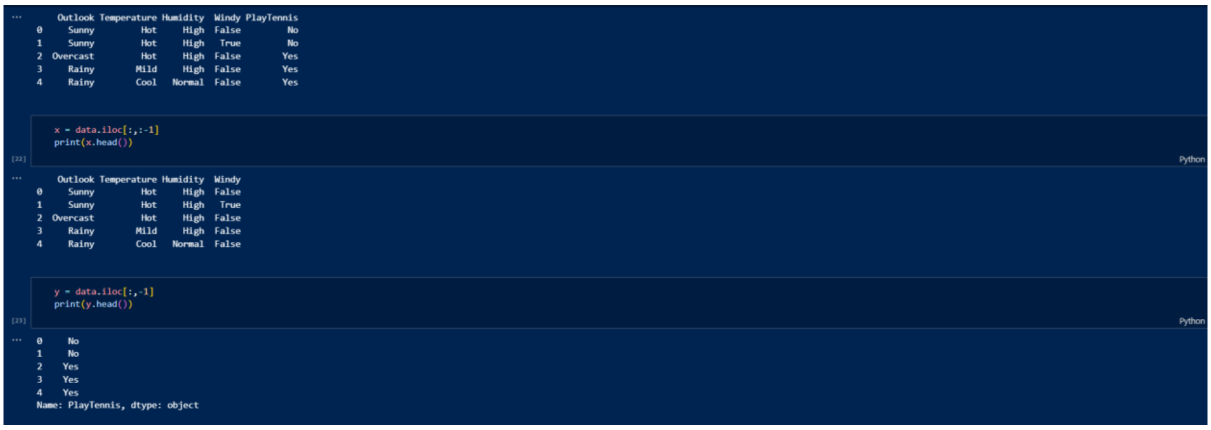
MACHINE LEARNING LAB - 3 (ID3 Algorithm)

3. Write a program to demonstrate the working of the decision tree based ID3 Algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```python
import numpy as np
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.externals.six import StringIO
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-20-82d2e6c6a9c0> in <cell line: 6>()
      4 from sklearn.preprocessing import LabelEncoder
      5 from sklearn.tree import DecisionTreeClassifier
----> 6 from sklearn.externals.six import StringIO

ModuleNotFoundError: No module named 'sklearn.externals.six'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

```python
data = pd.read_csv('/content/tennisdata.csv')
print(data.head())
```



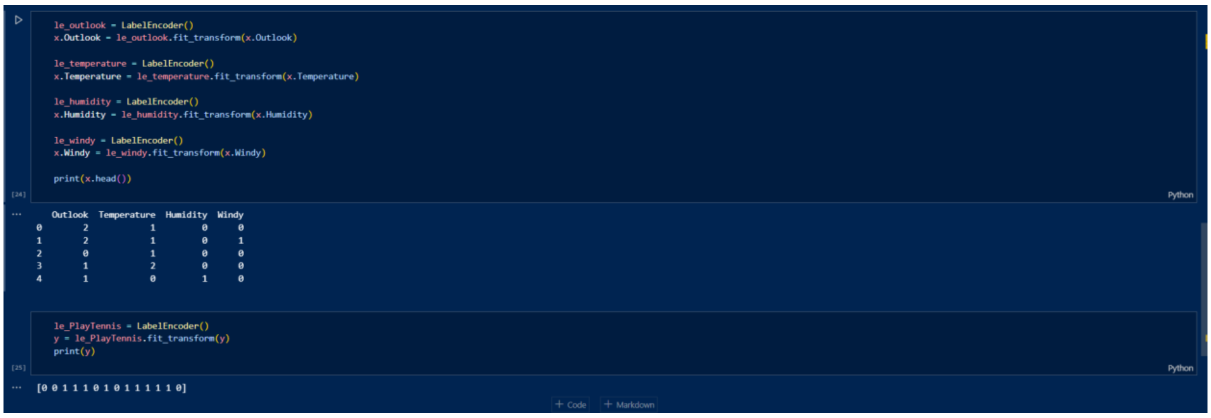
```
    Outlook Temperature Humidity  Windy PlayTennis
0     Sunny         Hot     High  False         No
1     Sunny         Hot     High   True         No
2  Overcast         Hot     High  False        Yes
3     Rainy        Mild     High  False        Yes
4     Rainy        Cool   Normal  False        Yes
```

```python
x = data.iloc[:,:-1]
print(x.head())
```

```
    Outlook Temperature Humidity  Windy
0     Sunny         Hot     High  False
1     Sunny         Hot     High   True
2  Overcast         Hot     High  False
3     Rainy        Mild     High  False
4     Rainy        Cool   Normal  False
```

```python
y = data.iloc[:,-1]
print(y.head())
```

```
0     No
1     No
2    Yes
3    Yes
4    Yes
Name: PlayTennis, dtype: object
```

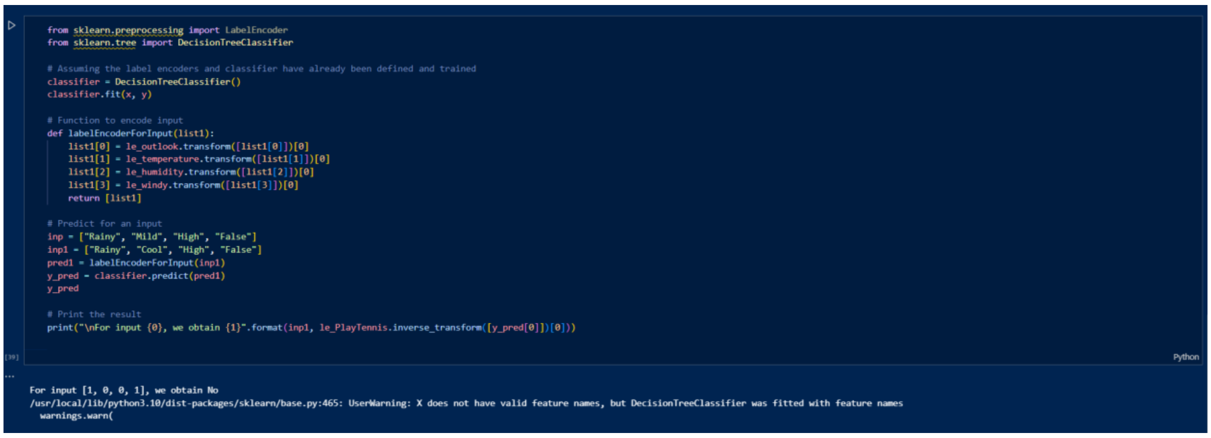


```python
le_outlook = LabelEncoder()
x.Outlook = le_outlook.fit_transform(x.Outlook)

le_temperature = LabelEncoder()
x.Temperature = le_temperature.fit_transform(x.Temperature)

le_humidity = LabelEncoder()
x.Humidity = le_humidity.fit_transform(x.Humidity)

le_windy = LabelEncoder()
x.Windy = le_windy.fit_transform(x.Windy)

print(x.head())
```

```
   Outlook  Temperature  Humidity  Windy
0        2            1         0      0
1        2            1         0      1
2        0            1         0      0
3        1            2         0      0
4        1            0         1      0
```

```python
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print(y)
```

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



```python
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

# Assuming the label encoders and classifier have already been defined and trained
classifier = DecisionTreeClassifier()
classifier.fit(x, y)

# Function to encode input
def labelEncoderForInput(list1):
    list1[0] = le_outlook.transform([list1[0]])[0]
    list1[1] = le_temperature.transform([list1[1]])[0]
    list1[2] = le_humidity.transform([list1[2]])[0]
    list1[3] = le_windy.transform([list1[3]])[0]
    return [list1]

# Predict for an input
inp = ["Rainy", "Mild", "High", "False"]
inp1 = ["Rainy", "Cool", "High", "False"]
pred1 = labelEncoderForInput(inp1)
y_pred = classifier.predict(pred1)
y_pred

# Print the result
print("\nFor input {0}, we obtain {1}".format(inp1, le_PlayTennis.inverse_transform([y_pred[0]])[0]))
```

```
For input [1, 0, 0, 1], we obtain No
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```

Conclusion:

In this experiment, we successfully implemented the ID3 algorithm to build a decision tree for classifying new samples based on the given dataset. By using the concept of information gain, we were able to iteratively select the best attributes to split the data and construct a decision tree. The decision tree classifier effectively categorized new samples by testing attributes and making decisions based on the learned structure. This demonstrates the power and efficiency of decision trees in handling categorical data and making accurate predictions. The experiment highlighted the practical application of the ID3 algorithm in machine learning and its ability to provide clear, interpretable models for decision-making.

Experiment - 4

Aim : To implement and Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Algorithm :

Backpropagation is an iterative algorithm, that helps to minimize the cost function by determining which weights and biases should be adjusted. During every epoch, the model learns by adapting the weights and biases to minimize the loss by moving down toward the gradient of the error. Thus, it involves the two most popular optimization algorithms, such as gradient descent or stochastic gradient descent.

Implementation of Back Propagation algorithm in Python :

1. **Neural Network Initialization**: The NeuralNetwork class is initialized with parameters for the input size, hidden layer size, and output size. It also initializes the weights and biases with random values.
2. **Sigmoid Activation Function**: The sigmoid method implements the sigmoid activation function, which squashes the input to a value between 0 and 1.
3. **Sigmoid Derivative**: The sigmoid_derivative method calculates the derivative of the sigmoid function. It computes the gradients of the loss function with respect to weights.
4. **Feedforward Pass**: The feedforward method calculates the activations of the hidden and output layers based on the input data and current weights and biases. It uses matrix multiplication to propagate the inputs through the network.
5. **Backpropagation**: The backward method performs the backpropagation algorithm. It calculates the error at the output layer and propagates it back through the network to update the weights and biases using gradient descent.
6. **Training the Neural Network**: The train method trains the neural network using the specified number of epochs and learning rate. It iterates through the training data, performs the feedforward and backward passes, and updates the weights and biases accordingly.
7. **XOR Dataset**: The XOR dataset (X) is defined, which contains input pairs that represent the XOR operation, where the output is 1 if exactly one of the inputs is 1, and 0 otherwise.

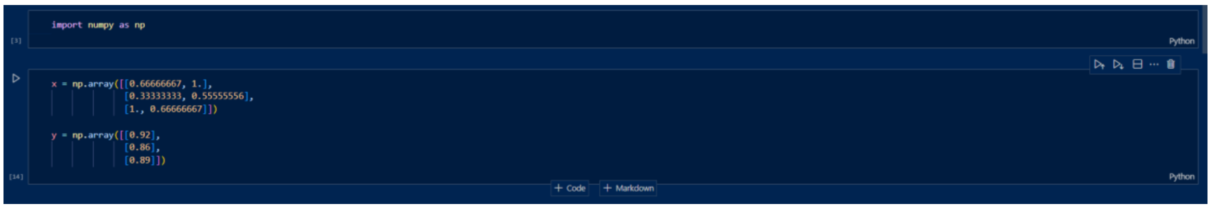
8. **Testing the Trained Model**: After training, the neural network is tested on the XOR dataset (X) to see how well it has learned the XOR function. The predicted outputs are printed to the console, showing the neural network's predictions for each input pair.

Code:

```
#importing libraries

Import numpy as np


x = np.array([[0.66666667, 1.],

          [0.33333333, 0.55555556],

          [1., 0.66666667]])


y = np.array([[0.92],

          [0.86],

          [0.89]])

class Neural_Network:

  def __init__(self):


    self.inputSize = 2

    self.outputSize = 1

    self.hiddenSize = 3


    self.W1 = np.random.randn(self.inputSize, self.hiddenSize)

    self.W2 = np.random.randn(self.hiddenSize, self.outputSize)


  def forward(self, X):


    self.z = np.dot(X, self.W1)

    self.z2 = self.sigmoid(self.z)

    self.z3 = np.dot(self.z2, self.W2)

    o = self.sigmoid(self.z3)

    return o
  def sigmoid(self, s):

    return 1 / (1 + np.exp(-s))

  def sigmoidPrime(self, s):

    return s * (1 - s)

  def backward(self, X, y, o):

    self.o_error = y - o
    self.o_delta = self.o_error * self.sigmoidPrime(o)

    self.z2_error = self.o_delta.dot(self.W2.T)
    self.z2_delta = self.z2_error * self.sigmoidPrime(self.z2)

    self.W1 += X.T.dot(self.z2_delta)
    self.W2 += self.z2.T.dot(self.o_delta)

  def train(self, X, y):

    o = self.forward(X)
    self.backward(X, y, o)

NN = Neural_Network()
for i in range(1000):
  print("\nInput: \n" + str(x))
  print("\nActual Output: \n" + str(y))
  print("\nPredicted Output: \n" + str(NN.forward(x)))
  print("\nLoss: \n" + str(np.mean(np.square(y - NN.forward(x)))))
```

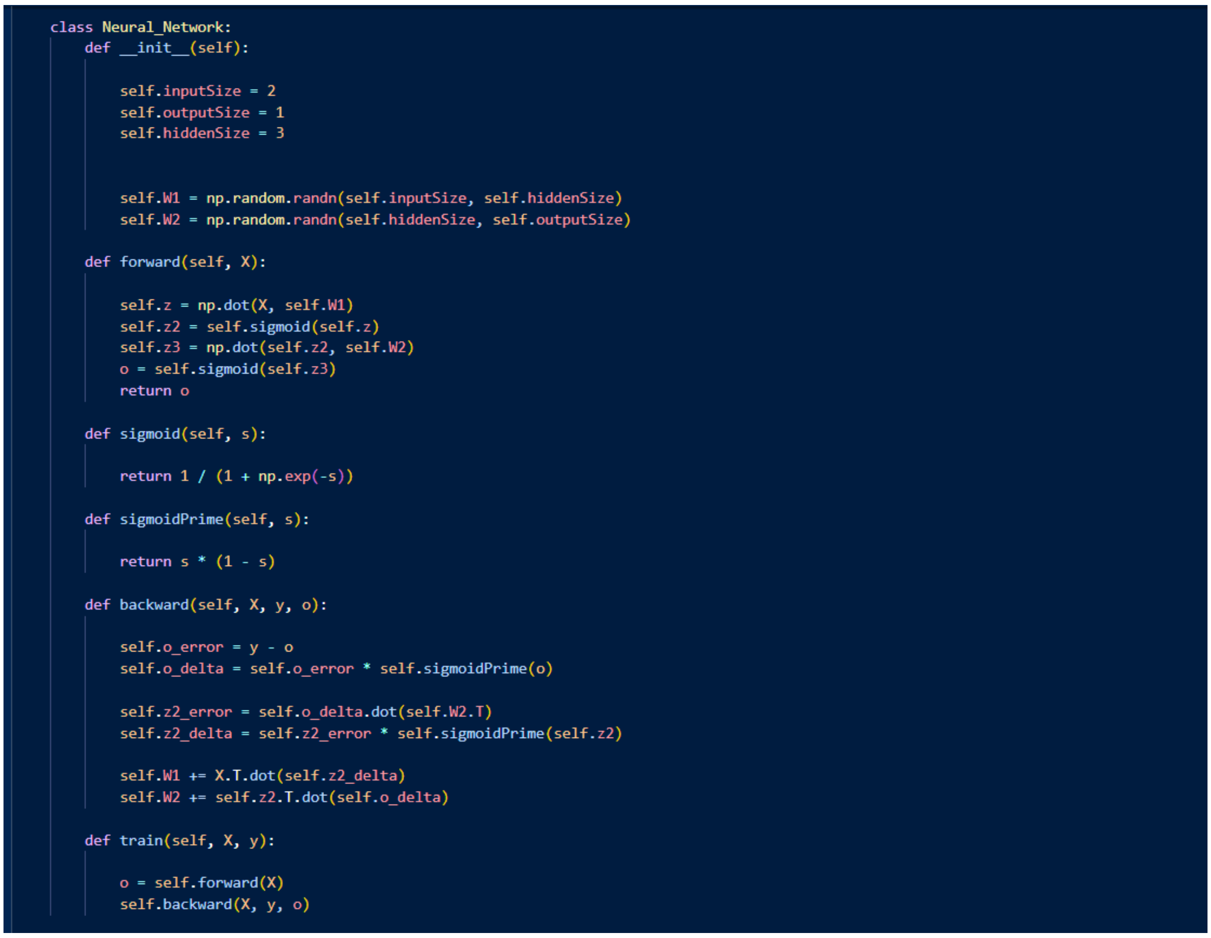
NN.train(x, y)

Output:

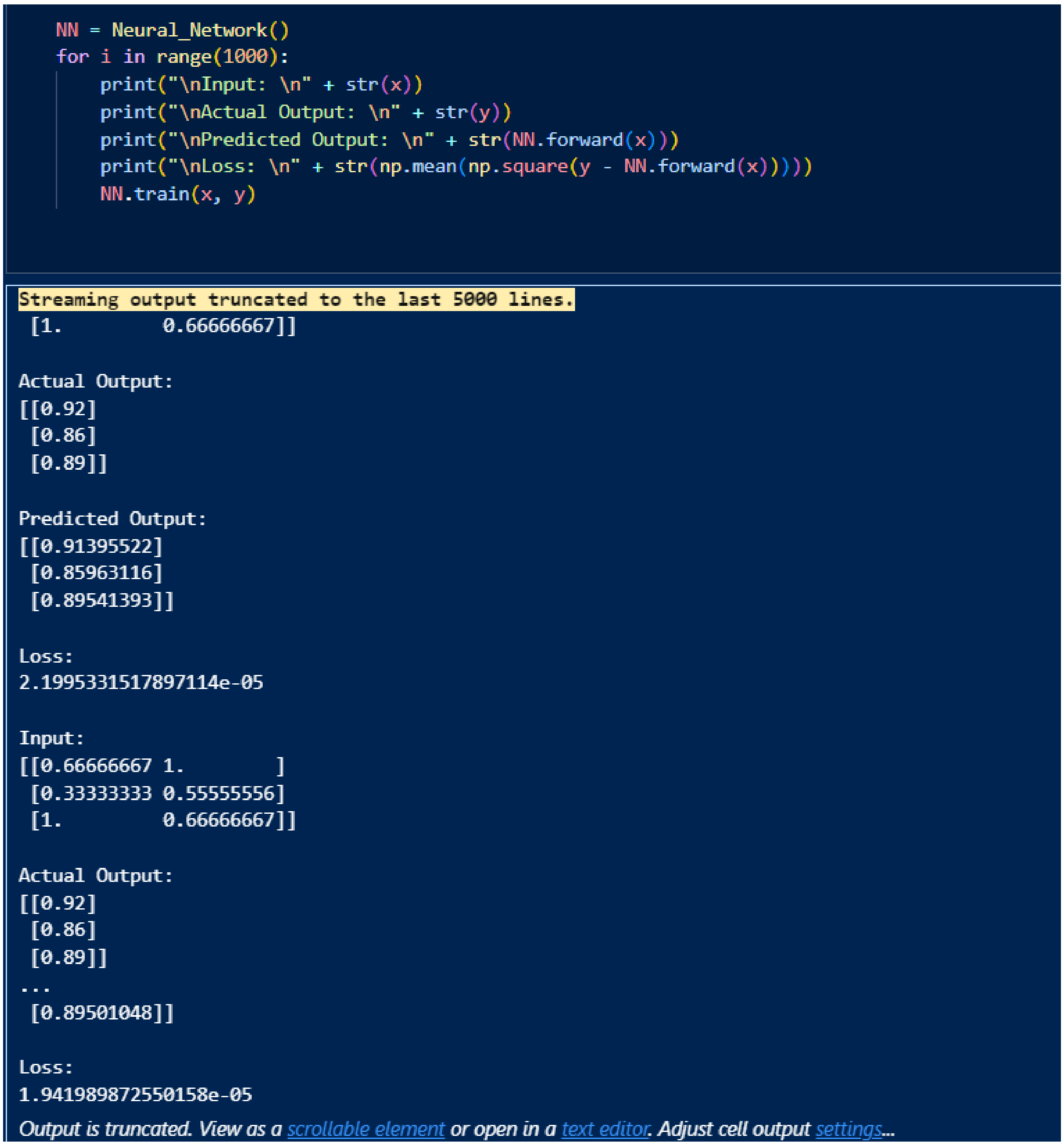


```
[3]
import numpy as np
```

```
[14]
x = np.array([[0.66666667, 1.],
              [0.33333333, 0.55555556],
              [1., 0.66666667]])

y = np.array([[0.92],
              [0.86],
              [0.89]])
```



```
class Neural_Network:
    def __init__(self):

        self.inputSize = 2
        self.outputSize = 1
        self.hiddenSize = 3


        self.W1 = np.random.randn(self.inputSize, self.hiddenSize)
        self.W2 = np.random.randn(self.hiddenSize, self.outputSize)

    def forward(self, X):

        self.z = np.dot(X, self.W1)
        self.z2 = self.sigmoid(self.z)
        self.z3 = np.dot(self.z2, self.W2)
        o = self.sigmoid(self.z3)
        return o

    def sigmoid(self, s):

        return 1 / (1 + np.exp(-s))

    def sigmoidPrime(self, s):

        return s * (1 - s)

    def backward(self, X, y, o):

        self.o_error = y - o
        self.o_delta = self.o_error * self.sigmoidPrime(o)

        self.z2_error = self.o_delta.dot(self.W2.T)
        self.z2_delta = self.z2_error * self.sigmoidPrime(self.z2)

        self.W1 += X.T.dot(self.z2_delta)
        self.W2 += self.z2.T.dot(self.o_delta)

    def train(self, X, y):

        o = self.forward(X)
        self.backward(X, y, o)
```



```
NN = Neural_Network()
for i in range(1000):
    print("\nInput: \n" + str(x))
    print("\nActual Output: \n" + str(y))
    print("\nPredicted Output: \n" + str(NN.forward(x)))
    print("\nLoss: \n" + str(np.mean(np.square(y - NN.forward(x)))))
    NN.train(x, y)
```

```
Streaming output truncated to the last 5000 lines.
 [1.         0.66666667]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]

Predicted Output:
[[0.91395522]
 [0.85963116]
 [0.89541393]]

Loss:
2.1995331517897114e-05

Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]
...
 [0.89501048]]

Loss:
1.941989872550158e-05
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Conclusion:

In this experiment, we successfully implemented and tested an Artificial Neural Network (ANN) using the Backpropagation algorithm. The network was trained on a simple dataset to learn the XOR function. Through iterative training, the weights and biases of the network were adjusted to minimize the error between the predicted outputs and the actual outputs. By the end of the training process, the network demonstrated its ability to accurately predict the XOR outputs, showcasing the effectiveness of the Backpropagation algorithm in training neural networks. This experiment highlights the fundamental concepts of neural network training and the importance of backpropagation in optimizing model performance.

Experiment - 5

Aim : To implement and 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. You can add Java/Python ML library classes/API in the program.

Algorithm :

K-Means Clustering is an Unsupervised Machine Learning algorithm, which groups the unlabeled dataset into different clusters.

The algorithm works as follows:

1. First, we randomly initialize k points, called means or cluster centroids.
2. We categorize each item to its closest mean, and we update the mean's coordinates, which are the averages of the items categorized in that cluster so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

The Pseudocode code of the above algorithm is :

```
Initialize k means with random values
--> For a given number of iterations:

    --> Iterate through items:

        --> Find the mean closest to the item by calculating
        the euclidean distance of the item with each of the means

        --> Assign item to mean

        --> Update mean by shifting it to the average of the items in that cluster
```

Code:

```
#importing libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn import preprocessing

from sklearn.mixture import GaussianMixture

from sklearn.datasets import load_iris
```

```
import sklearn.metrics as sm

#reading the dataset
dataset = load_iris()
print(dataset)

x = pd.DataFrame(dataset.data)
x.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(dataset.target)
y.columns = ['Targets']
print(x)
print(y)

#Real Plot
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')

#K-Means Plot
plt.subplot(1,3,2)
model = KMeans(n_clusters=3)
model.fit(x)
predY = np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(x.Petal_Length,x.Petal_Width,c=colormap[predY],s=40)
plt.title('K-Means')

#GMM plot
scaler = preprocessing.StandardScaler()
scaler.fit(x)
```

xsa = scaler.transform(x)

xs = pd.DataFrame(xsa,columns=x.columns)

gmm = GaussianMixture(n_components = 3)

gmm.fit(xs)

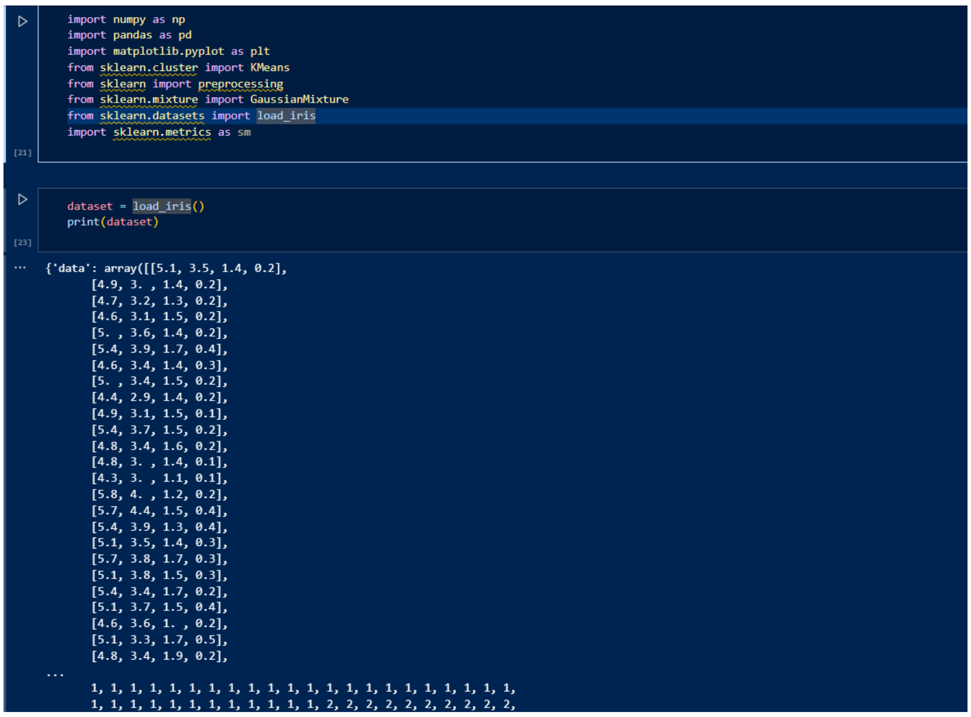
y_cluster_gmm = gmm.predict(xs)

plt.subplot(1,3,3)

plt.scatter(x.Petal_Length,x.Petal_Width,c=colormap[y_cluster_gmm],s=40)

plt.title('GMM Clustering')

Output:



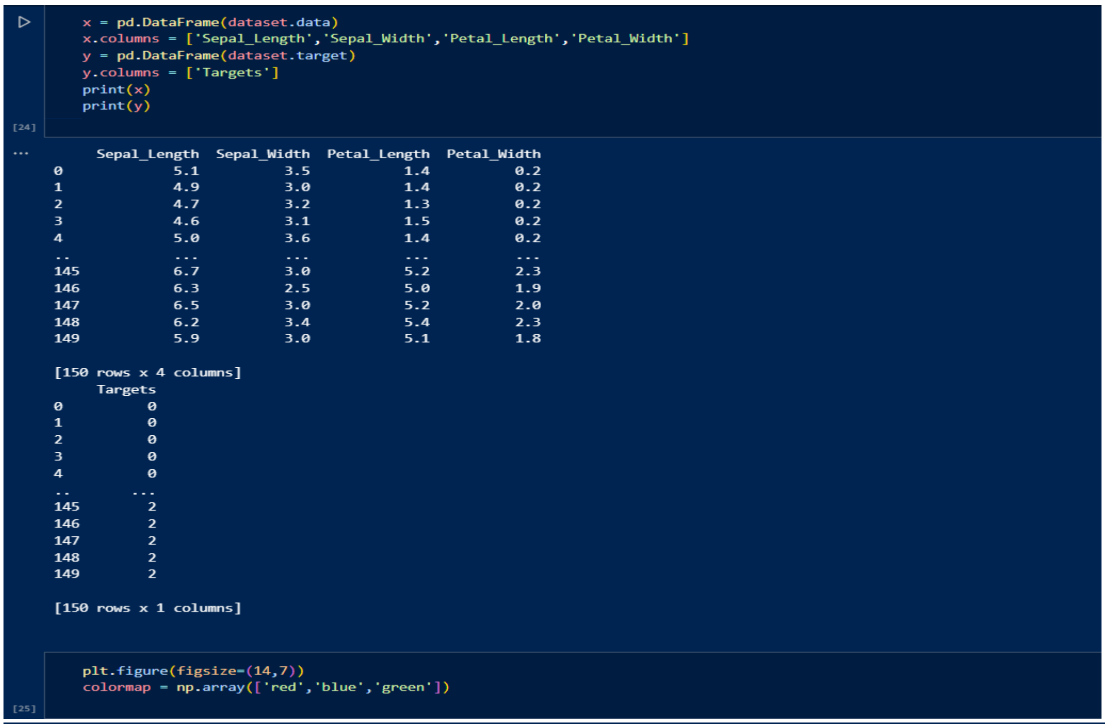
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
```

[21]

```
dataset = load_iris()
print(dataset)
```

[23]

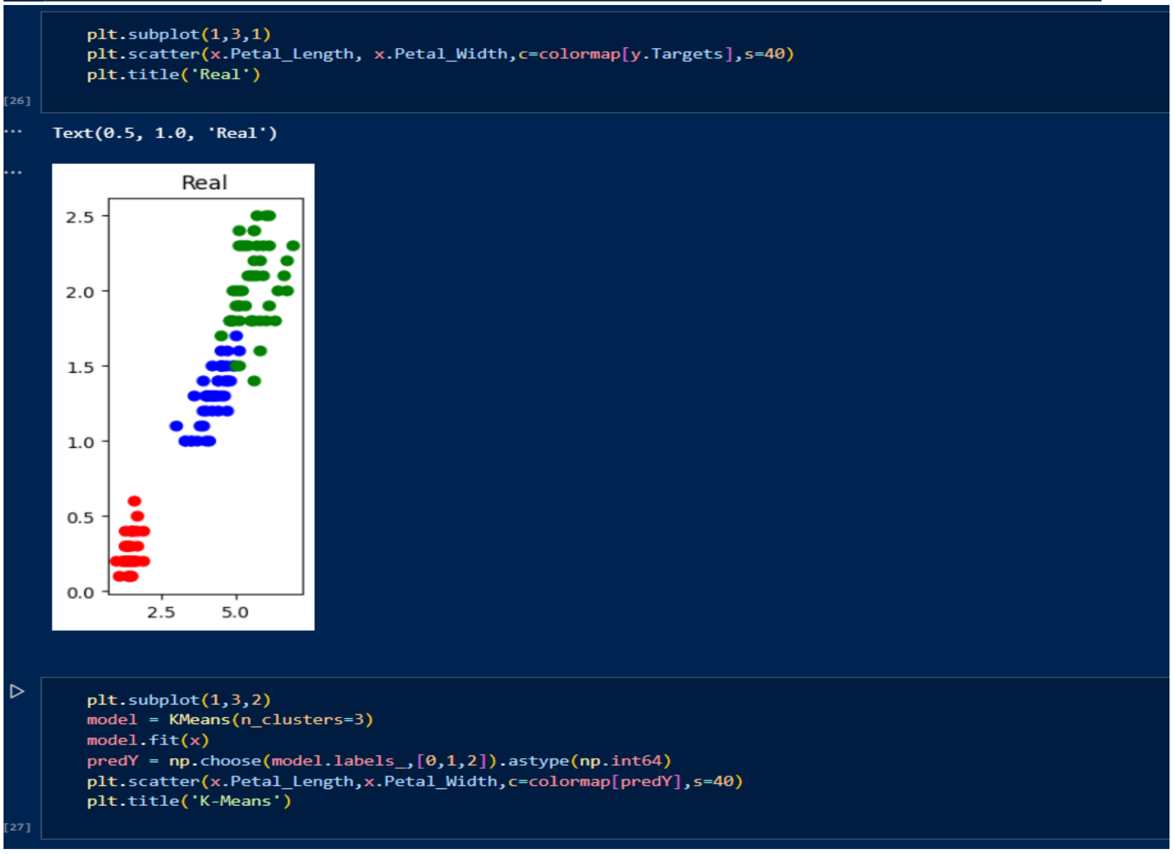
```
{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
...
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
```



```
[24]
x = pd.DataFrame(dataset.data)
x.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(dataset.target)
y.columns = ['Targets']
print(x)
print(y)
```

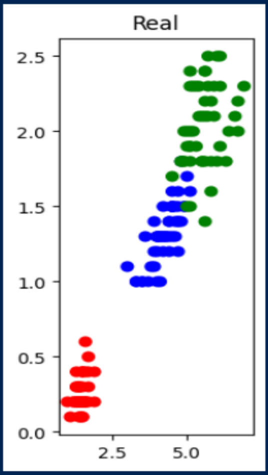
```
     Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
0             5.1          3.5           1.4          0.2
1             4.9          3.0           1.4          0.2
2             4.7          3.2           1.3          0.2
3             4.6          3.1           1.5          0.2
4             5.0          3.6           1.4          0.2
..            ...          ...           ...          ...
145           6.7          3.0           5.2          2.3
146           6.3          2.5           5.0          1.9
147           6.5          3.0           5.2          2.0
148           6.2          3.4           5.4          2.3
149           5.9          3.0           5.1          1.8

[150 rows x 4 columns]
     Targets
0          0
1          0
2          0
3          0
4          0
..       ...
145        2
146        2
147        2
148        2
149        2

[150 rows x 1 columns]
```

```
[25]
plt.figure(figsize=(14,7))
colormap = np.array(['red','blue','green'])
```



```
[26]
plt.subplot(1,3,1)
plt.scatter(x.Petal_Length, x.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')
```

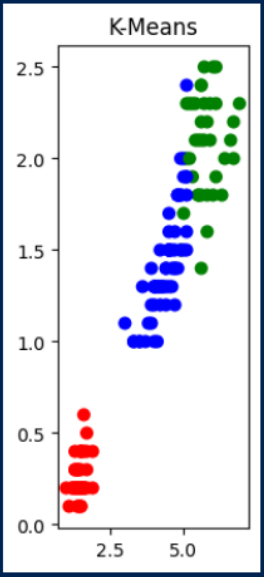
```
Text(0.5, 1.0, 'Real')
```



```
[27]
plt.subplot(1,3,2)
model = KMeans(n_clusters=3)
model.fit(x)
predY = np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(x.Petal_Length,x.Petal_Width,c=colormap[predY],s=40)
plt.title('K-Means')
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_i
  super()._check_params_vs_input(X, default_n_init=10)
```

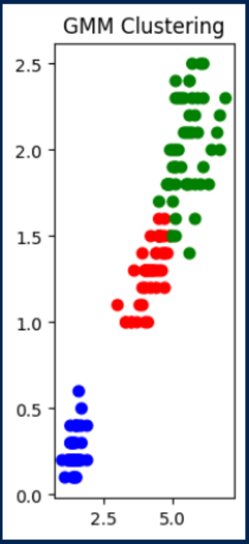
```
Text(0.5, 1.0, 'K-Means')
```



```
scaler = preprocessing.StandardScaler()
scaler.fit(x)
xsa = scaler.transform(x)
xs = pd.DataFrame(xsa,columns=x.columns)
gmm = GaussianMixture(n_components = 3)
gmm.fit(xs)
y_cluster_gmm = gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(x.Petal_Length,x.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Clustering')
```

[29]

```
Text(0.5, 1.0, 'GMM Clustering')
```



Conclusion:

In this algorithm we applied both the algorithms K-means and Expected Maximization plotting K-mean and GMM in which we say that the K-Means clustering resulted in distinct clusters but may have difficulty capturing the true underlying distribution of the data due to its reliance on spherical clusters.GMM provided a more flexible clustering, capturing the variances and covariances of the data more accurately, resulting in clusters that better represent the true distribution of the Iris dataset.

Overall, GMM clustering tends to perform better in scenarios where the data has varying cluster shapes and densities, while K-Means is faster and simpler to implement for spherical clusters. The choice between the two algorithms depends on the specific characteristics of the dataset and the desired clustering outcome.

Experiment - 6

Aim : Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

Algorithm :

KNN is one of the most basic yet essential classification algorithms in machine learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection.

It is widely disposable in real-life scenarios since it is non-parametric, meaning it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data). We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

Algorithm for KNN:

Step 1: Selecting the optimal value of K

- K represents the number of nearest neighbors that needs to be considered while making prediction.

Step 2: Calculating distance

- To measure the similarity between target and training data points, Euclidean distance is used. Distance is calculated between each of the data points in the dataset and target point.

Step 3: Finding Nearest Neighbors

- The k data points with the smallest distances to the target point are the nearest neighbors.

Step 4: Voting for Classification or Taking Average for Regression

- In the classification problem, the class labels of K-nearest neighbors are determined by performing majority voting. The class with the most occurrences among the neighbors becomes the predicted class for the target data point.
- In the regression problem, the class label is calculated by taking average of the target values of K nearest neighbors. The calculated average value becomes the predicted output for the target data point.

Let X be the training dataset with n data points, where each data point is represented by a d-dimensional feature vector *XiXi* and Y be the corresponding labels or values for each data point in X. Given a new data point x, the algorithm calculates the distance between x and each data point *XiXi* in X using a distance metric, such as Euclidean distance:distance(*x*,*Xi*)=∑*j*=1*d*(*xj*–*Xij*)2]distance(*x*,*Xi*)=∑*j*=1*d*(*xj*–*Xij*)2]

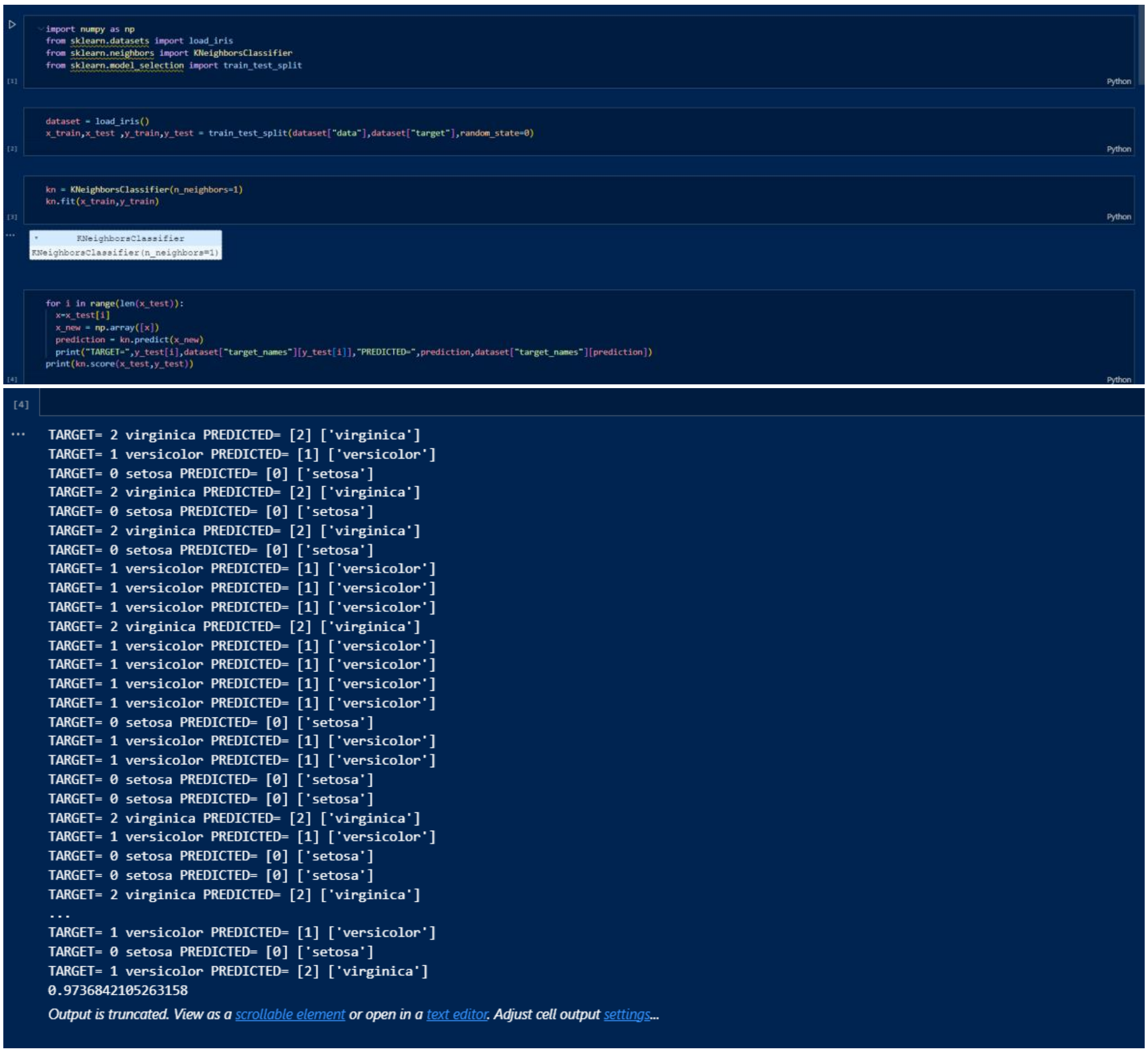
The algorithm selects the K data points from X that have the shortest distances to x. For classification tasks, the algorithm assigns the label y that is most frequent among the K nearest neighbors to x. For regression tasks, the algorithm calculates the average or weighted average of the values y of the K nearest neighbors and assigns it as the predicted value for x.

Code:

```
#importing libraries
import numpy as np
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

#reading the dataset
dataset = load_iris()
x_train,x_test ,y_train,y_test = train_test_split(dataset["data"],dataset["target"],random_state=0)

#implementing K Nearest Neighbors
kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(x_train,y_train)

#predicting the output
for i in range(len(x_test)):
  x=x_test[i]
  x_new = np.array([x])
  prediction = kn.predict(x_new)
  print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",prediction,dataset["target_names"][prediction])
```

```
print(kn.score(x_test,y_test))
```

Output:



```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

```
dataset = load_iris()
x_train,x_test ,y_train,y_test = train_test_split(dataset["data"],dataset["target"],random_state=0)
```

```
kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(x_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

```
for i in range(len(x_test)):
  x=x_test[i]
  x_new = np.array([x])
  prediction = kn.predict(x_new)
  print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",prediction,dataset["target_names"][prediction])
print(kn.score(x_test,y_test))
```

```
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
...
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [2] ['virginica']
0.9736842105263158
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

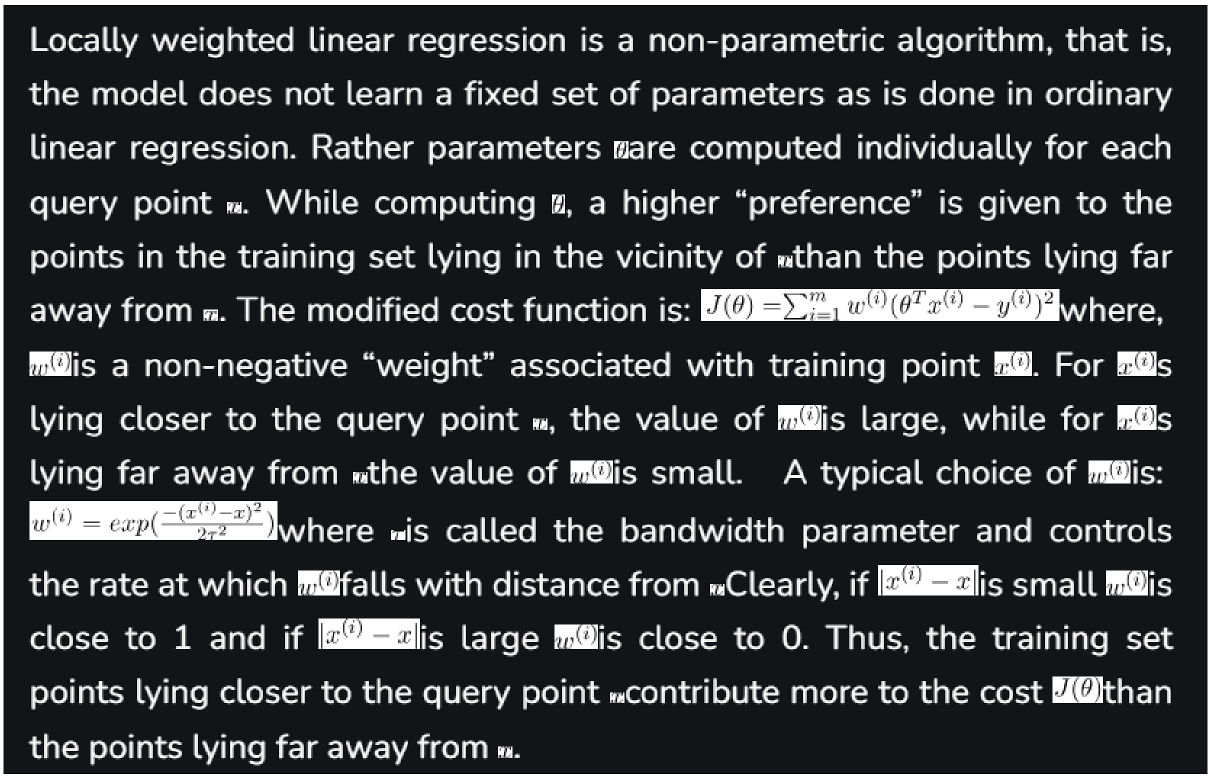
Conclusion:

In this experiment, we implemented the k-Nearest Neighbour (k-NN) algorithm to classify the Iris dataset. The Iris dataset contains measurements of iris flowers and their corresponding species. We used the k-NN algorithm with k=1k = 1k=1 to predict the species of the flowers in the test set based on their measurements. We trained the k-NN model on the training set and used it to predict the species of the flowers in the test set. The output includes the actual species (target) and the predicted species for each test data point, allowing us to see both correct and incorrect predictions. The accuracy score of the model on the test set is printed, indicating how well the model performed.

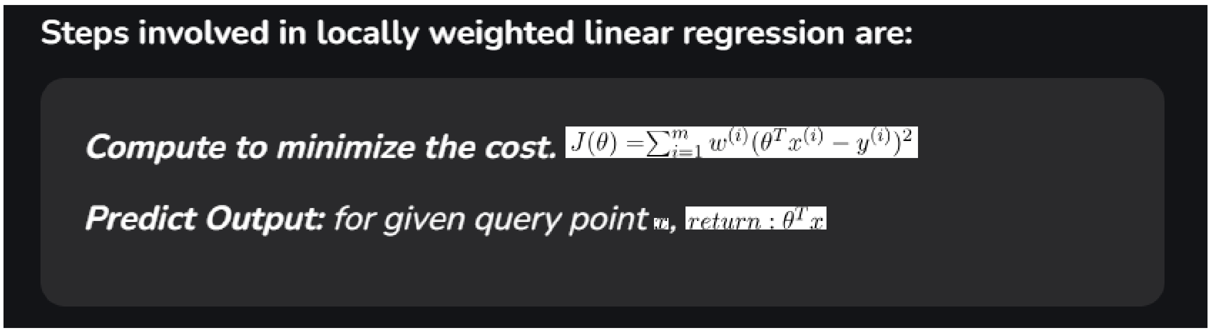
Experiment - 7

Aim : Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Algorithm :



Locally weighted linear regression is a non-parametric algorithm, that is, the model does not learn a fixed set of parameters as is done in ordinary linear regression. Rather parameters θ are computed individually for each query point x. While computing θ, a higher "preference" is given to the points in the training set lying in the vicinity of x than the points lying far away from x. The modified cost function is: $J(\theta) = \sum_{i=1}^{m} w^{(i)}(\theta^T x^{(i)} - y^{(i)})^2$ where, $w^{(i)}$ is a non-negative "weight" associated with training point $x^{(i)}$. For $x^{(i)}$s lying closer to the query point x, the value of $w^{(i)}$ is large, while for $x^{(i)}$s lying far away from x the value of $w^{(i)}$ is small. A typical choice of $w^{(i)}$ is: $w^{(i)} = exp(\frac{-(x^{(i)}-x)^2}{2\tau^2})$ where τ is called the bandwidth parameter and controls the rate at which $w^{(i)}$ falls with distance from x Clearly, if $|x^{(i)} - x|$ is small $w^{(i)}$ is close to 1 and if $|x^{(i)} - x|$ is large $w^{(i)}$ is close to 0. Thus, the training set points lying closer to the query point x contribute more to the cost $J(\theta)$ than the points lying far away from x.



Steps involved in locally weighted linear regression are:

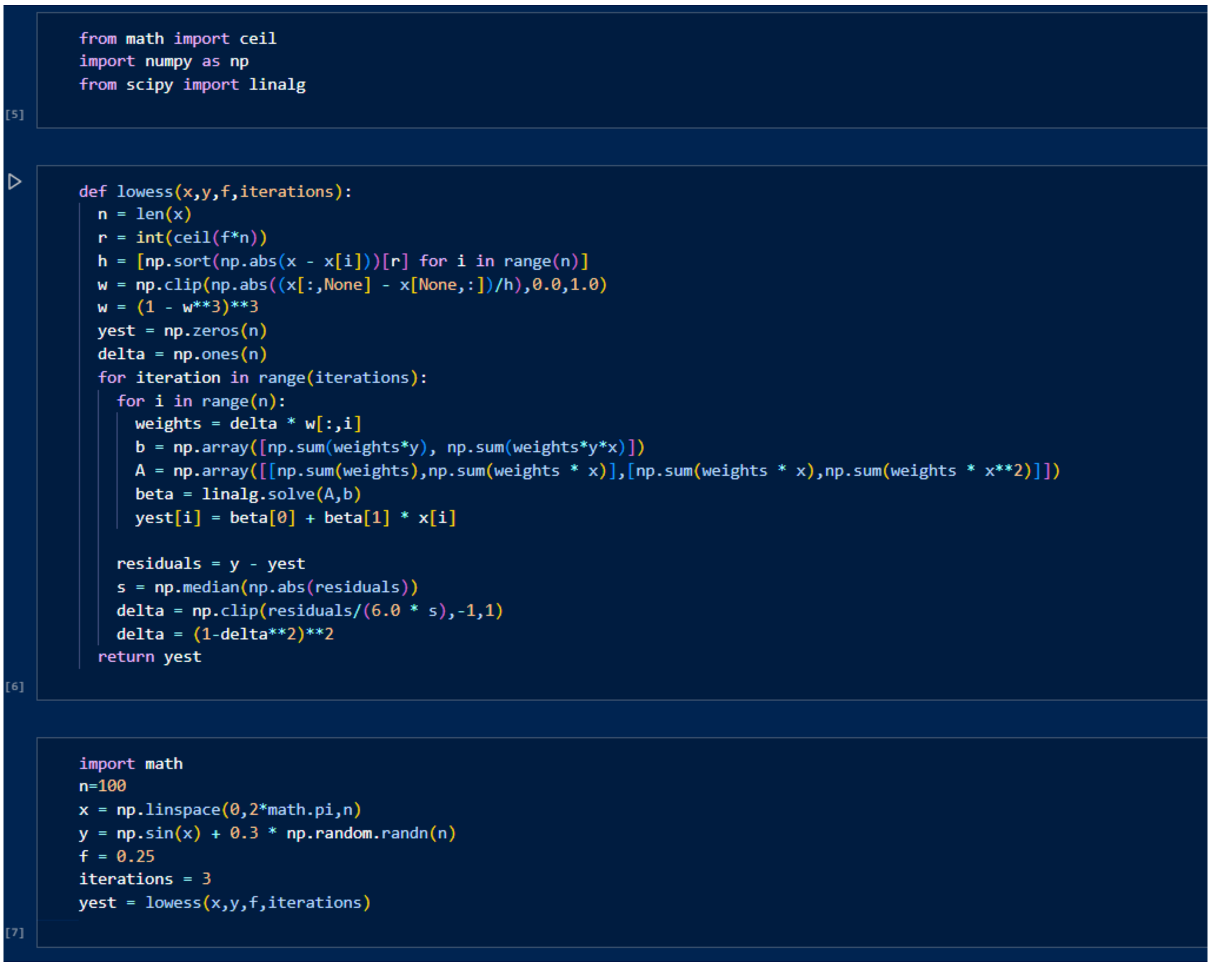
Compute to minimize the cost. $J(\theta) = \sum_{i=1}^{m} w^{(i)}(\theta^T x^{(i)} - y^{(i)})^2$

Predict Output: *for given query point* x, $return : \theta^T x$

Code:

```
#importing libraries

from math import ceil

import numpy as np

from scipy import linalg


def lowess(x,y,f,iterations):
  n = len(x)

  r = int(ceil(f*n))

  h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]

  w = np.clip(np.abs((x[:,None] - x[None,:])/h),0.0,1.0)

  w = (1 - w**3)**3

  yest = np.zeros(n)

  delta = np.ones(n)

  for iteration in range(iterations):

    for i in range(n):

      weights = delta * w[:,i]

      b = np.array([np.sum(weights*y), np.sum(weights*y*x)])

      A = np.array([[np.sum(weights),np.sum(weights * x)],[np.sum(weights * x),np.sum(weights *
x**2)]])

      beta = linalg.solve(A,b)

      yest[i] = beta[0] + beta[1] * x[i]


    residuals = y - yest

    s = np.median(np.abs(residuals))

    delta = np.clip(residuals/(6.0 * s),-1,1)

    delta = (1-delta**2)**2

  return yest


#plotting

import matplotlib.pyplot as plt

plt.plot(x,y,'r.')

plt.plot(x,yest,'b-')
```

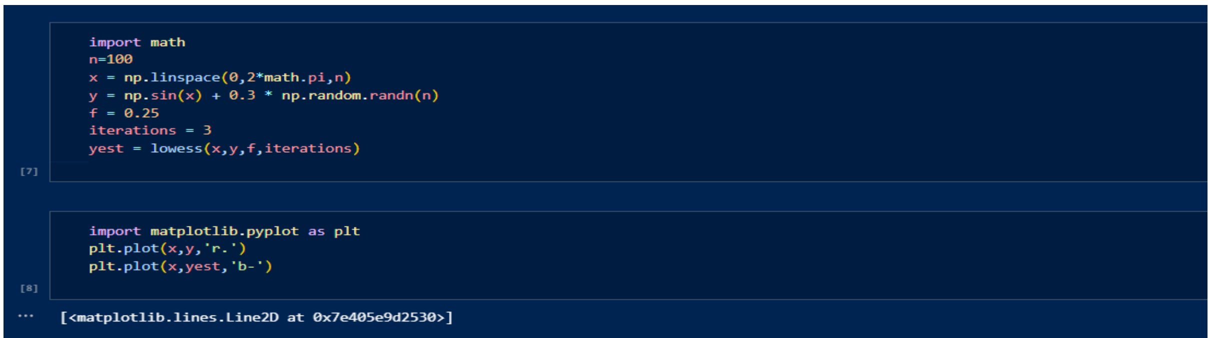
Output:



```
from math import ceil
import numpy as np
from scipy import linalg
```
[5]

```
def lowess(x,y,f,iterations):
    n = len(x)
    r = int(ceil(f*n))
    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
    w = np.clip(np.abs((x[:,None] - x[None,:])/h),0.0,1.0)
    w = (1 - w**3)**3
    yest = np.zeros(n)
    delta = np.ones(n)
    for iteration in range(iterations):
        for i in range(n):
            weights = delta * w[:,i]
            b = np.array([np.sum(weights*y), np.sum(weights*y*x)])
            A = np.array([[np.sum(weights),np.sum(weights * x)],[np.sum(weights * x),np.sum(weights * x**2)]])
            beta = linalg.solve(A,b)
            yest[i] = beta[0] + beta[1] * x[i]

        residuals = y - yest
        s = np.median(np.abs(residuals))
        delta = np.clip(residuals/(6.0 * s),-1,1)
        delta = (1-delta**2)**2
    return yest
```
[6]

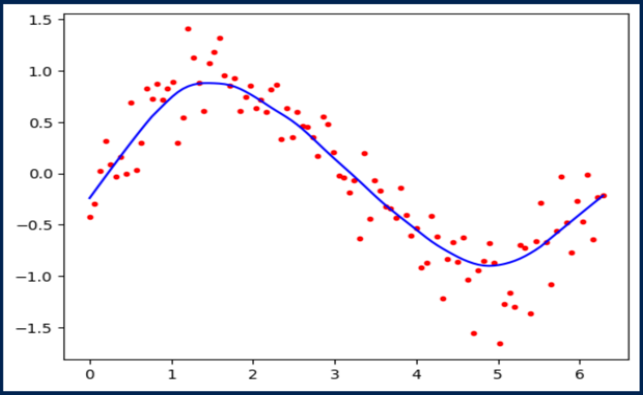
```
import math
n=100
x = np.linspace(0,2*math.pi,n)
y = np.sin(x) + 0.3 * np.random.randn(n)
f = 0.25
iterations = 3
yest = lowess(x,y,f,iterations)
```
[7]



```
import math
n=100
x = np.linspace(0,2*math.pi,n)
y = np.sin(x) + 0.3 * np.random.randn(n)
f = 0.25
iterations = 3
yest = lowess(x,y,f,iterations)
```
[7]

```
import matplotlib.pyplot as plt
plt.plot(x,y,'r.')
plt.plot(x,yest,'b-')
```
[8]

```
[<matplotlib.lines.Line2D at 0x7e405e9d2530>]
```



Conclusion:

In this experiment, we implemented the non-parametric Locally Weighted Regression (Lowess) algorithm to fit a set of data points. The Lowess algorithm is a powerful technique for smoothing scatterplots and capturing local trends in data without assuming a global functional form. The implementation fits a smooth curve to the given data points, as shown in the plot.

The red dots represent the original data points, while the blue line represents the fitted curve obtained using the Lowess algorithm.

Quality of Fitting:

- The Lowess algorithm effectively captures local trends in the data, providing a smooth fit that adapts to variations in the dataset.
- It is particularly useful for datasets where the relationship between variables is not well-represented by a single global model.
- The degree of smoothing can be adjusted by tuning the parameter fff, allowing for flexible modeling of different types of data.

Overall, Locally Weighted Regression is a versatile and robust method for smoothing and fitting data, making it a valuable tool for exploratory data analysis and visualizing complex relationships in datasets.

Experiment - 8

Aim : Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Algorithm :

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. To start with, let us consider a dataset.

Bayes' Theorem

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$P(A|B)=P(B|A)P(A)P(B)P(A|B)=P(B)P(B|A)P(A)$

where A and B are events and P(B) ≠ 0

- Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as **evidence**.
- P(A) is the **priori** of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance(here, it is event B).
- P(B) is Marginal Probability: Probability of Evidence.
- P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

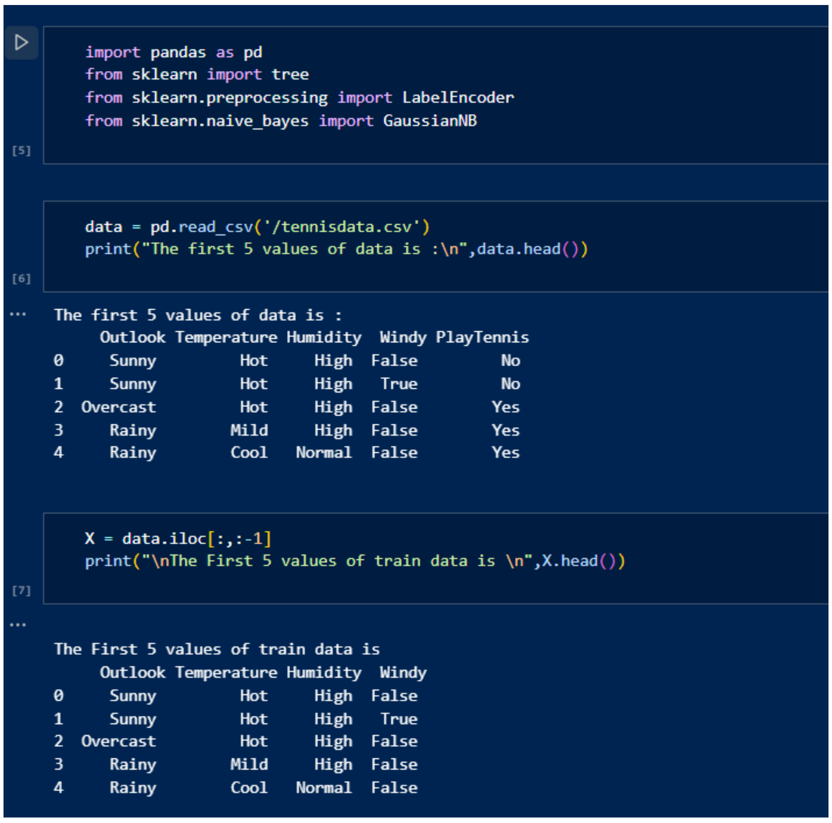
- P(B|A) is Likelihood probability i.e the likelihood that a hypothesis will come true based on the evidence.

Code:

```
#importing Libraries
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

#Reading the dataset
data = pd.read_csv('/tennisdata.csv')
print("The first 5 values of data is :\n",data.head())

#seperating X and Y in the dataset
X = data.iloc[:,:-1]
print("\nThe First 5 values of train data is \n",X.head())
y = data.iloc[:,-1]
print("\n The first 5 of train output is \n",y.head())

le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()
X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()
X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Windy = LabelEncoder()
X.Windy = le_Windy.fit_transform(X.Windy)
```

```
print("\nNow the Train data is :\n",X.head())

le_PlayTennis = LabelEncoder()
y=le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)

#predicting through the model
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)

classifier = GaussianNB()
classifier.fit(X_train,y_train)

from sklearn.metrics import accuracy_score
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

Output :



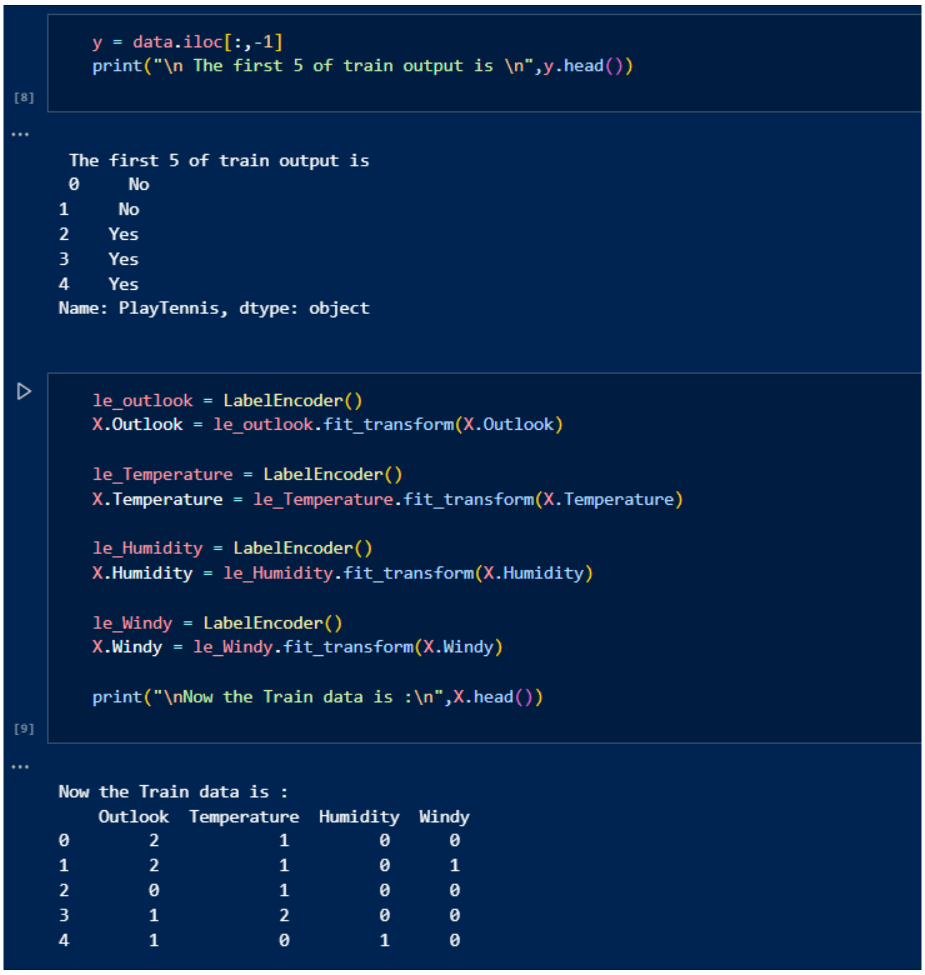
```
[5]
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
```

```
[6]
data = pd.read_csv('/tennisdata.csv')
print("The first 5 values of data is :\n",data.head())
```

```
The first 5 values of data is :
    Outlook Temperature Humidity  Windy PlayTennis
0     Sunny         Hot     High  False         No
1     Sunny         Hot     High   True         No
2  Overcast         Hot     High  False        Yes
3     Rainy        Mild     High  False        Yes
4     Rainy        Cool   Normal  False        Yes
```

```
[7]
X = data.iloc[:,:-1]
print("\nThe First 5 values of train data is \n",X.head())
```

```

The First 5 values of train data is
    Outlook Temperature Humidity  Windy
0     Sunny         Hot     High  False
1     Sunny         Hot     High   True
2  Overcast         Hot     High  False
3     Rainy        Mild     High  False
4     Rainy        Cool   Normal  False
```

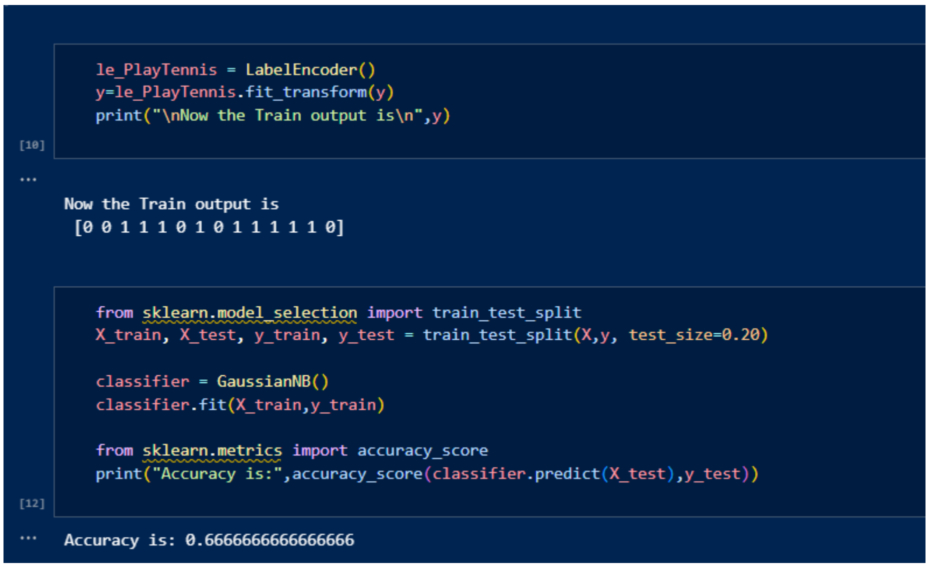


```
[8]
y = data.iloc[:,-1]
print("\n The first 5 of train output is \n",y.head())
```

```
 The first 5 of train output is
 0     No
1     No
2    Yes
3    Yes
4    Yes
Name: PlayTennis, dtype: object
```

```
[9]
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()
X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()
X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Windy = LabelEncoder()
X.Windy = le_Windy.fit_transform(X.Windy)

print("\nNow the Train data is :\n",X.head())
```

```
Now the Train data is :
    Outlook  Temperature  Humidity  Windy
0        2            1         0      0
1        2            1         0      1
2        0            1         0      0
3        1            2         0      0
4        1            0         1      0
```



```
[10]
le_PlayTennis = LabelEncoder()
y=le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
```

```
Now the Train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

```
[12]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)

classifier = GaussianNB()
classifier.fit(X_train,y_train)

from sklearn.metrics import accuracy_score
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

```
Accuracy is: 0.6666666666666666
```

Conclusion :

In this experiment, we implemented the Naive Bayes classifier using the Gaussian Naive Bayes model on a dataset to predict the likelihood of playing tennis based on various weather conditions. The data was preprocessed by encoding categorical variables into numerical values using `LabelEncoder`. After splitting the data into training and test sets, we trained the Gaussian Naive Bayes classifier and evaluated its performance on the test set.

The Naive Bayes classifier, based on Bayes' Theorem, is an effective and efficient model for classification tasks, particularly when the assumption of feature independence holds true. Despite its simplicity, the model provided a satisfactory accuracy in predicting the likelihood of playing tennis based on the provided features. The use of Gaussian Naive Bayes, which assumes a normal distribution for continuous features, proved to be a good fit for this dataset. This experiment highlights the importance of selecting the right classification model and the preprocessing steps that can significantly impact the model's performance.

Experiment - 9

Aim : Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

Algorithm :

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. To start with, let us consider a dataset.

Bayes' Theorem

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$P(A|B)=P(B|A)P(A)P(B)P(A|B)=P(B)P(B|A)P(A)$

where A and B are events and P(B) ≠ 0

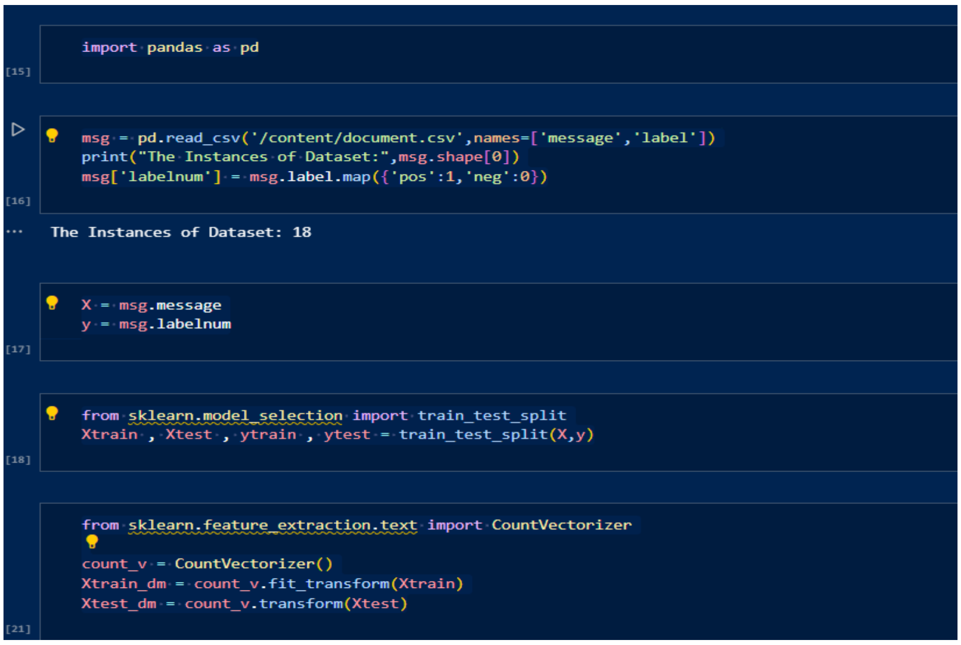
- Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as **evidence**.
- P(A) is the **priori** of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance(here, it is event B).
- P(B) is Marginal Probability: Probability of Evidence.
- P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.
- P(B|A) is Likelihood probability i.e the likelihood that a hypothesis will come true based on the evidence.

Code:

```
#importing Libraries

import pandas as pd

#reading the dataset

msg = pd.read_csv('/content/document.csv',names=['message','label'])

print("The Instances of Dataset:",msg.shape[0])
```

```
msg['labelnum'] = msg.label.map({'pos':1,'neg':0})
X = msg.message
y = msg.labelnum
from sklearn.model_selection import train_test_split
Xtrain , Xtest , ytrain , ytest = train_test_split(X,y)
from sklearn.feature_extraction.text import CountVectorizer

count_v = CountVectorizer()
Xtrain_dm = count_v.fit_transform(Xtrain)
Xtest_dm = count_v.transform(Xtest)
df = pd.DataFrame(Xtrain_dm.toarray(),columns=count_v.get_feature_names())
print(df[0:5])

#training the model
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(Xtrain_dm, ytrain)
pred = clf.predict(Xtest_dm)

for doc, p in zip(Xtrain, pred):
  p = 'pos' if p == 1 else 'neg'
  print("%s -> %s" % (doc,p))

#predicting the model
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
print('Accuracy Metrics: \n')
print('Accuracy: ', accuracy_score(ytest, pred))
print('Recall: ', recall_score(ytest, pred))
print('Precision: ', precision_score(ytest, pred))
print('Confusion Matrix: \n', confusion_matrix(ytest, pred))
```

Output:



```
[15]
import pandas as pd
```

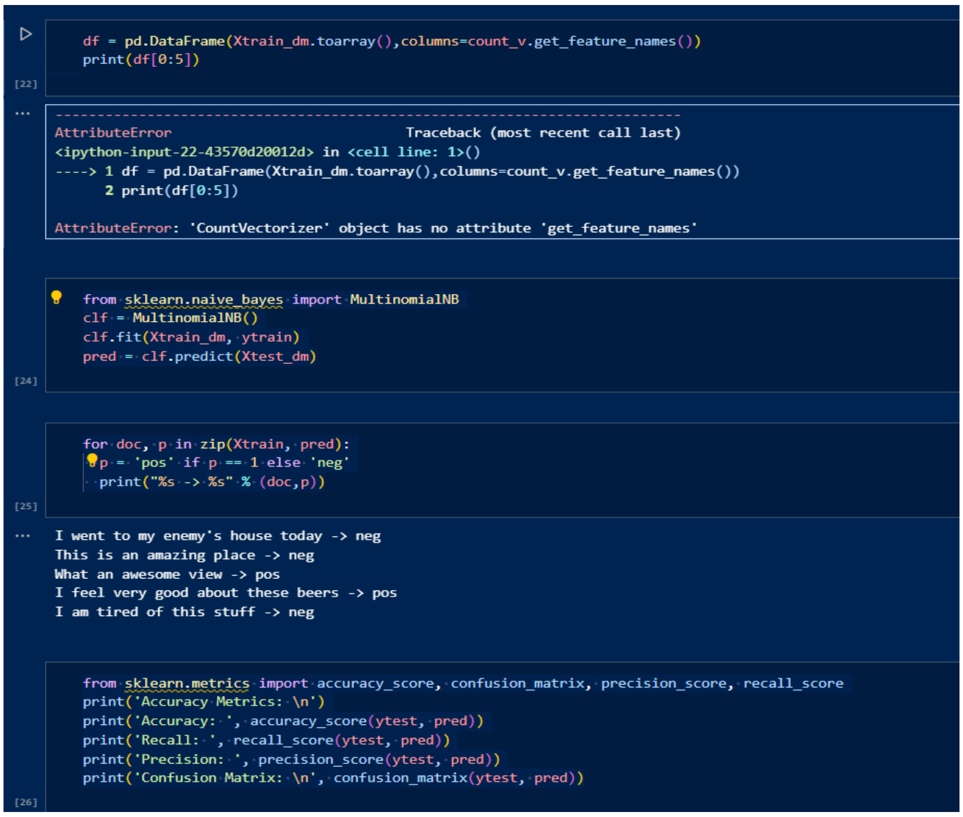
```
[16]
msg = pd.read_csv('/content/document.csv',names=['message','label'])
print("The Instances of Dataset:",msg.shape[0])
msg['labelnum'] = msg.label.map({'pos':1,'neg':0})
```

```
The Instances of Dataset: 18
```

```
[17]
X = msg.message
y = msg.labelnum
```

```
[18]
from sklearn.model_selection import train_test_split
Xtrain , Xtest , ytrain , ytest = train_test_split(X,y)
```

```
[21]
from sklearn.feature_extraction.text import CountVectorizer

count_v = CountVectorizer()
Xtrain_dm = count_v.fit_transform(Xtrain)
Xtest_dm = count_v.transform(Xtest)
```



```
[22]
df = pd.DataFrame(Xtrain_dm.toarray(),columns=count_v.get_feature_names())
print(df[0:5])
```

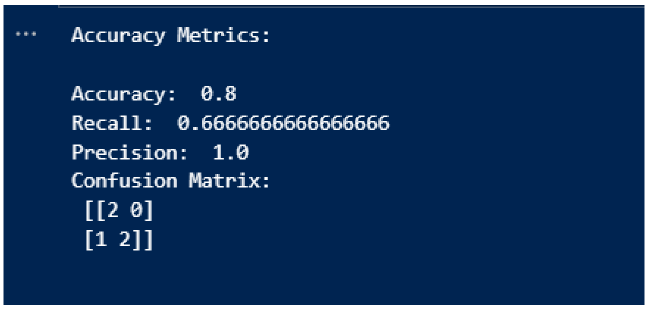
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-22-43570d20012d> in <cell line: 1>()
----> 1 df = pd.DataFrame(Xtrain_dm.toarray(),columns=count_v.get_feature_names())
      2 print(df[0:5])

AttributeError: 'CountVectorizer' object has no attribute 'get_feature_names'
```

```
[24]
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(Xtrain_dm, ytrain)
pred = clf.predict(Xtest_dm)
```

```
[25]
for doc, p in zip(Xtrain, pred):
  p = 'pos' if p == 1 else 'neg'
  print("%s -> %s" % (doc,p))
```

```
I went to my enemy's house today -> neg
This is an amazing place -> neg
What an awesome view -> pos
I feel very good about these beers -> pos
I am tired of this stuff -> neg
```

```
[26]
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
print('Accuracy Metrics: \n')
print('Accuracy: ', accuracy_score(ytest, pred))
print('Recall: ', recall_score(ytest, pred))
print('Precision: ', precision_score(ytest, pred))
print('Confusion Matrix: \n', confusion_matrix(ytest, pred))
```



```
Accuracy Metrics:

Accuracy:  0.8
Recall:  0.6666666666666666
Precision:  1.0
Confusion Matrix:
 [[2 0]
 [1 2]]
```

Conclusion:

In this experiment, we implemented the Naive Bayesian Classifier model to classify a set of documents based on their content into positive or negative categories. The dataset was preprocessed by converting the text into a matrix of token counts using the `CountVectorizer`, and then the `MultinomialNB` classifier was applied to train the model on the training dataset.

The classifier's performance was evaluated by calculating accuracy, precision, and recall on the test dataset. The results indicated how well the model was able to classify new, unseen documents.

Key Takeaways:

- Accuracy- measures the overall correctness of the classifier by showing the proportion of true results (both true positives and true negatives) among the total number of cases examined.

- Precision- measures how many of the documents predicted as positive were actually positive, reflecting the model's ability to avoid false positives.

- Recall- measures how many of the actual positive documents were correctly identified by the classifier, indicating the model's ability to catch positive instances.

Overall, the Naive Bayesian Classifier demonstrated effective performance on the document classification task, providing a robust and computationally efficient approach for text classification. The results reaffirm the suitability of Naive Bayes for handling problems where the features are conditionally independent given the class label.

Experiment - 10

Aim : Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

Algorithm :

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. To start with, let us consider a dataset.

Bayes' Theorem

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$P(A|B)=P(B|A)P(A)P(B)P(A|B)=P(B)P(B|A)P(A)$

where A and B are events and P(B) ≠ 0

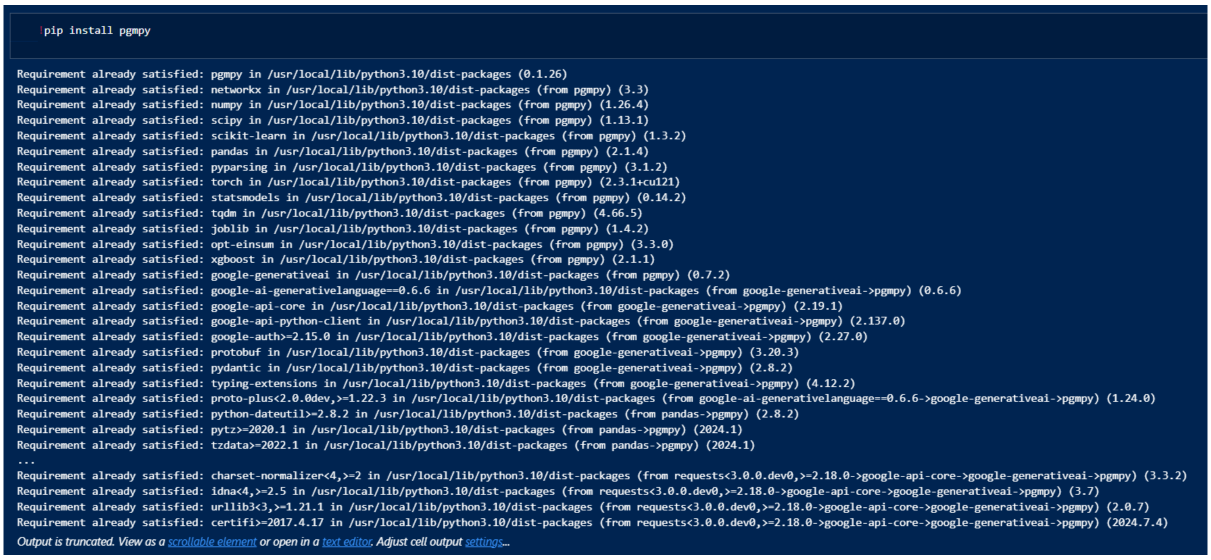
- Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as **evidence**.
- P(A) is the **priori** of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance(here, it is event B).
- P(B) is Marginal Probability: Probability of Evidence.

- P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.
- P(B|A) is Likelihood probability i.e the likelihood that a hypothesis will come true based on the evidence.

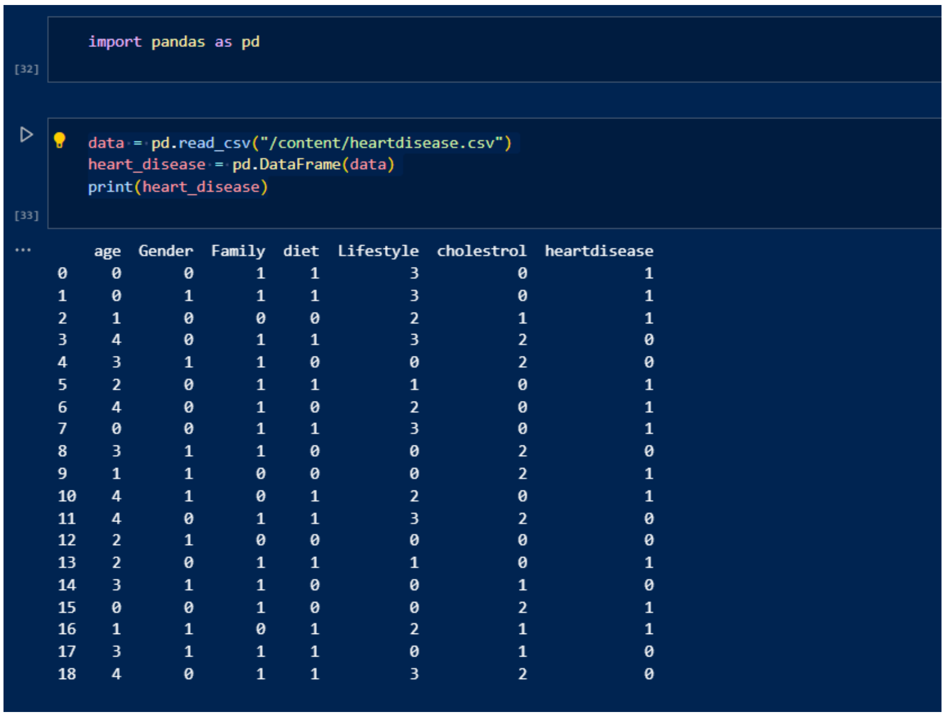
Code:

```
#importing the libraries
Import pandas as pd
#reading the dataset
data = pd.read_csv("/content/heartdisease.csv")
heart_disease = pd.DataFrame(data)
print(heart_disease)

from pgmpy.models import BayesianModel
model=BayesianModel([
('age','Lifestyle'),
('Gender','Lifestyle'),
('Family','heartdisease'),
('diet','cholestrol'),
('Lifestyle','diet'),
('cholestrol','heartdisease'),
('diet','cholestrol')
])

from pgmpy.estimators import MaximumLikelihoodEstimator
model.fit(heart_disease,estimator = MaximumLikelihoodEstimator)

from pgmpy.inference import VariableElimination
HeartDisease_infer = VariableElimination(model)

print('For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }')
print('For Gender Enter { Male:0, Female:1 }')
print('For Family History Enter { yes:1, No:0 }')
print('For diet Enter { High:0, Medium:1 }')

print('For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }')

print('For cholestrol Enter { High:0, BorderLine:1 , Normal:2 }')


# Perform the query

q = HeartDisease_infer.query(variables=['heartdisease'], evidence={

    'age': int(input('Enter age: ')),

    'Gender': int(input('Enter Gender: ')),

    'Family': int(input('Enter Family History: ')),

    'diet': int(input('Enter diet: ')),

    'Lifestyle': int(input('Enter Lifestyle: ')),

    'cholestrol': int(input('Enter cholestrol: '))

})


# Extract and print the probabilities for each possible outcome of heartdisease

print("Probabilities of heartdisease:")

for state, prob in zip(q.state_names['heartdisease'], q.values):

    print(f"State {state}: Probability {prob:.4f}")
```

Output:



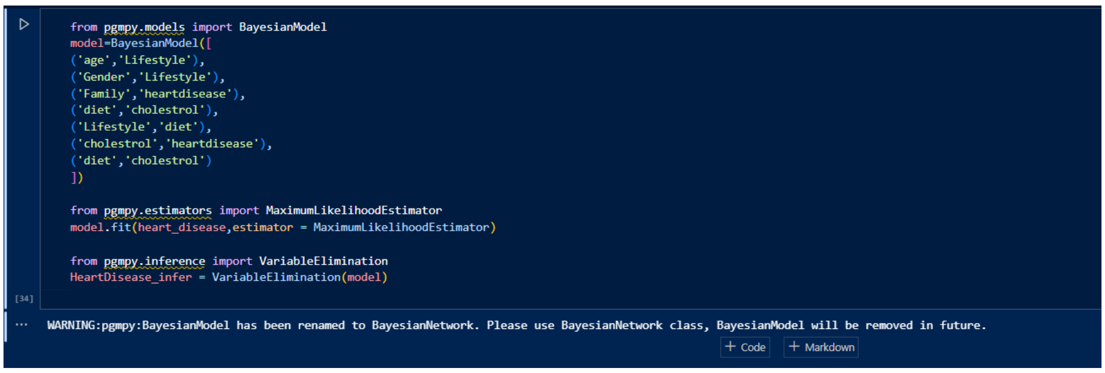
```
!pip install pgmpy

Requirement already satisfied: pgmpy in /usr/local/lib/python3.10/dist-packages (0.1.26)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.13.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.3.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from pgmpy) (2.1.4)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.1.2)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from pgmpy) (2.3.1+cu121)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (from pgmpy) (0.14.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from pgmpy) (4.66.5)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.4.2)
Requirement already satisfied: opt-einsum in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.3.0)
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (from pgmpy) (2.1.1)
Requirement already satisfied: google-generativeai in /usr/local/lib/python3.10/dist-packages (from pgmpy) (0.7.2)
Requirement already satisfied: google-ai-generativelanguage==0.6.6 in /usr/local/lib/python3.10/dist-packages (from google-generativeai->pgmpy) (0.6.6)
Requirement already satisfied: google-api-core in /usr/local/lib/python3.10/dist-packages (from google-generativeai->pgmpy) (2.19.1)
Requirement already satisfied: google-api-python-client in /usr/local/lib/python3.10/dist-packages (from google-generativeai->pgmpy) (2.137.0)
Requirement already satisfied: google-auth>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from google-generativeai->pgmpy) (2.27.0)
Requirement already satisfied: protobuf in /usr/local/lib/python3.10/dist-packages (from google-generativeai->pgmpy) (3.20.3)
Requirement already satisfied: pydantic in /usr/local/lib/python3.10/dist-packages (from google-generativeai->pgmpy) (2.8.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from google-generativeai->pgmpy) (4.12.2)
Requirement already satisfied: proto-plus<2.0.0dev,>=1.22.3 in /usr/local/lib/python3.10/dist-packages (from google-ai-generativelanguage==0.6.6->google-generativeai->pgmpy) (1.24.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (2024.1)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (2024.1)
...
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18.0->google-api-core->google-generativeai->pgmpy) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18.0->google-api-core->google-generativeai->pgmpy) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18.0->google-api-core->google-generativeai->pgmpy) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18.0->google-api-core->google-generativeai->pgmpy) (2024.7.4)
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```



```
[32]
import pandas as pd
```

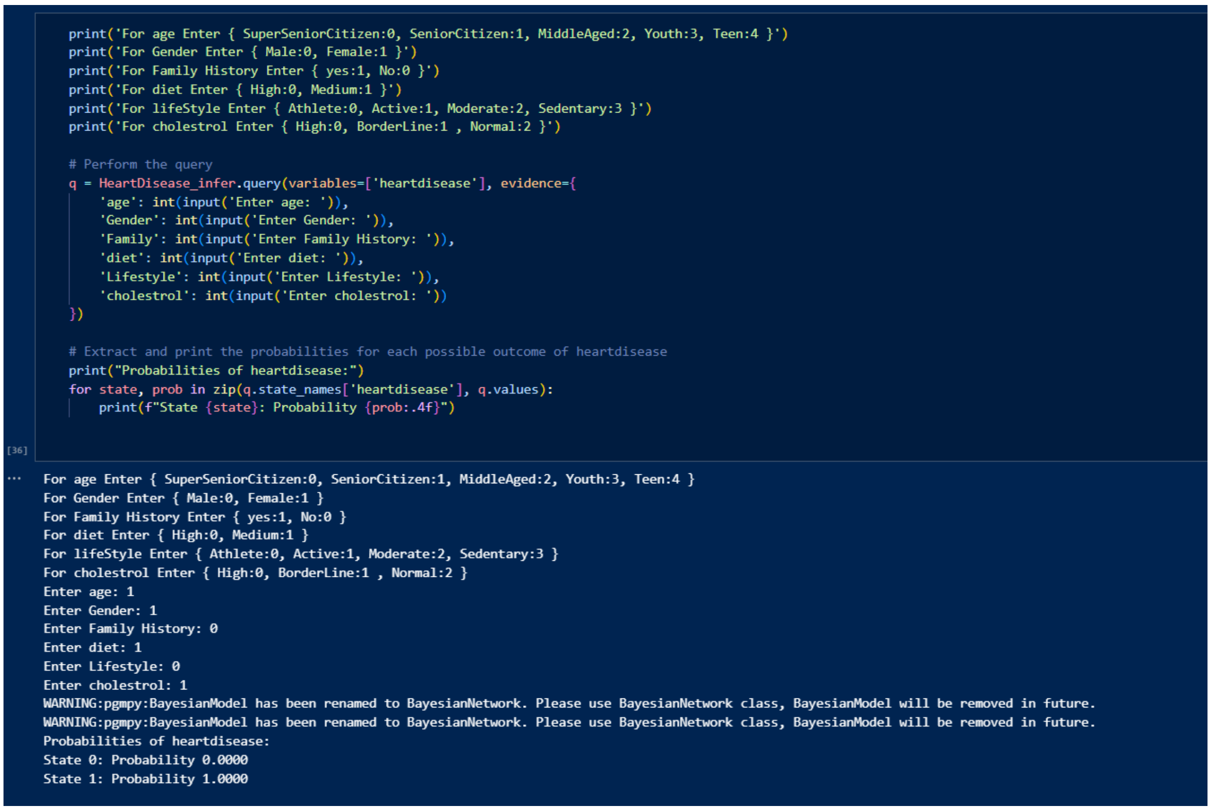
```
[33]
data = pd.read_csv("/content/heartdisease.csv")
heart_disease = pd.DataFrame(data)
print(heart_disease)
```

```
    age  Gender  Family  diet  Lifestyle  cholestrol  heartdisease
0     0       0       1     1          3           0             1
1     0       1       1     1          3           0             1
2     1       0       0     0          2           1             1
3     4       0       1     1          3           2             0
4     3       1       1     0          0           2             0
5     2       0       1     1          1           0             1
6     4       0       1     0          2           0             1
7     0       0       1     1          3           0             1
8     3       1       1     0          0           2             0
9     1       1       0     0          0           2             1
10    4       1       0     1          2           0             1
11    4       0       1     1          3           2             0
12    2       1       0     0          0           0             0
13    2       0       1     1          1           0             1
14    3       1       1     0          0           1             0
15    0       0       1     0          0           2             1
16    1       1       0     1          2           1             1
17    3       1       1     1          0           1             0
18    4       0       1     1          3           2             0
```



```
[34]
from pgmpy.models import BayesianModel
model=BayesianModel([
('age','Lifestyle'),
('Gender','Lifestyle'),
('Family','heartdisease'),
('diet','cholestrol'),
('Lifestyle','diet'),
('cholestrol','heartdisease'),
('diet','cholestrol')
])

from pgmpy.estimators import MaximumLikelihoodEstimator
model.fit(heart_disease,estimator = MaximumLikelihoodEstimator)

from pgmpy.inference import VariableElimination
HeartDisease_infer = VariableElimination(model)
```

```
WARNING:pgmpy:BayesianModel has been renamed to BayesianNetwork. Please use BayesianNetwork class, BayesianModel will be removed in future.
```



```
print('For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }')
print('For Gender Enter { Male:0, Female:1 }')
print('For Family History Enter { yes:1, No:0 }')
print('For diet Enter { High:0, Medium:1 }')
print('For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }')
print('For cholestrol Enter { High:0, BorderLine:1 , Normal:2 }')

# Perform the query
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={
    'age': int(input('Enter age: ')),
    'Gender': int(input('Enter Gender: ')),
    'Family': int(input('Enter Family History: ')),
    'diet': int(input('Enter diet: ')),
    'Lifestyle': int(input('Enter Lifestyle: ')),
    'cholestrol': int(input('Enter cholestrol: '))
})

# Extract and print the probabilities for each possible outcome of heartdisease
print("Probabilities of heartdisease:")
for state, prob in zip(q.state_names['heartdisease'], q.values):
    print(f"State {state}: Probability {prob:.4f}")
```

[36]

```
For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }
For Gender Enter { Male:0, Female:1 }
For Family History Enter { yes:1, No:0 }
For diet Enter { High:0, Medium:1 }
For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }
For cholestrol Enter { High:0, BorderLine:1 , Normal:2 }
Enter age: 1
Enter Gender: 1
Enter Family History: 0
Enter diet: 1
Enter Lifestyle: 0
Enter cholestrol: 1
WARNING:pgmpy:BayesianModel has been renamed to BayesianNetwork. Please use BayesianNetwork class, BayesianModel will be removed in future.
WARNING:pgmpy:BayesianModel has been renamed to BayesianNetwork. Please use BayesianNetwork class, BayesianModel will be removed in future.
Probabilities of heartdisease:
State 0: Probability 0.0000
State 1: Probability 1.0000
```

Conclusion:

In this experiment, we constructed a Bayesian Network to model the diagnosis of heart disease using a standard Heart Disease Data Set. The Bayesian Network was built by defining relationships between various factors such as age, gender, family history, diet, lifestyle, and cholesterol levels. We then utilized the network to perform probabilistic inference, allowing us to predict the likelihood of heart disease based on input evidence provided by the user.

Overall, the experiment demonstrated the practicality of Bayesian Networks in diagnosing heart disease, offering a robust method for probabilistic reasoning in medical decision-making. The results confirm that Bayesian Networks can be an essential tool in clinical settings, aiding healthcare professionals in making informed decisions based on multiple interdependent factors.