MEASI INSTITUTE OF INFORMATION TECHNOLOGY

(Approved by AICTE & Affiliated to University of Madras)

CHENNAI – 600 014

MEASI
Institute *of*
Information
Technology

MASTER OF COMPUTER APPLICATIONS

ACADEMIC YEAR 2023-2024

SEMESTER – III

Practical Record

Machine Learning Lab

REG. NO : _____

NAME : _____

BATCH : _____

MEASI INSTITUTE OF INFORMATION TECHNOLOGY

(Approved by AICTE & Affiliated to University of Madras)

CHENNAI- 600 014

MCA PRACTICAL

Machine Learning Lab

Academic Year 2023-2024

Semester - III

NAME : **CLASS :**

REG-NO : **BATCH :**

This is to certify that this is the bonafide record of work done in the Computer Science Laboratory of **MEASI Institute of Information Technology**, submitted for the **University of Madras** Practical Examination held on ..... at **MEASI Institute of Information Technology**, **Chennai-600 014**.

STAFF IN-CHARGE **HEAD OF THE DEPARTMENT**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

INDEX

EXP NO: 1

DATE:

FIND-S ALGORITHM

AIM:

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:

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x

   For each attribute constraint ai in h

   If the constraint ai is satisfied by x

   Then do nothing

   Else replace ai in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

DATASET:

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

PROGRAM:

```
import csv
num_attribute=6
a=[]
print("\n Given Training Data set\n")
with open('FIND-S.csv','r')as csvfile:
   reader=csv.reader(csvfile)
   for row in reader:
      a.append(row)
      print(row)
print("\n The intial value of hypothesis:")
hypothesis=['0']*num_attribute
print(hypothesis)
for j in range(0,num_attribute):
   hypothesis[j]=a[0][j]
   print(hypothesis)

print("\n Find S: Finding a Maximally Specific Hypothesis\n")
for i in range(0,len(a)):
   if a[i][num_attribute]=="Yes":
      for j in range(0,num_attribute):
         if a[i][j]!=hypothesis[j]:
            hypothesis[j]='?'
         else:
            hypothesis[j]=a[i][j]
      print("For Traning instance No:{0} the hypothesis is\n".format(i),hypothesis)
print("\n The Maximally Specific Hypothesis for a given Traning Examples:\n")
print(hypothesis)
```

OUTPUT:

The Given Training Data Set

['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']

The initial value of hypothesis: ['0', '0', '0', '0', '0', '0']

Find S: Finding a Maximally Specific Hypothesis

For Training Example No:0 the hypothesis is

['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

For Training Example No:1 the hypothesis is

['sunny', 'warm', '?', 'strong', 'warm', 'same']

For Training Example No:2 the hypothesis is

['sunny', 'warm', '?', 'strong', 'warm', 'same']

For Training Example No:3 the hypothesis is

['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally Specific Hypothesis for a given Training Examples:

['sunny', 'warm', '?', 'strong', '?', '?']

RESULT: The program was implemented successfully

EXP NO: 2 DATE:	CANDIDATE ELIMINATE

AIM:

The CANDIDATE-ELIMINTION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

ALGORITHM:

1.Initialize G to the set of maximally general hypotheses in H

2.Initialize S to the set of maximally specific hypotheses in H

3.For each training example d, do

- If d is a positive example
    - Remove from G any hypothesis inconsistent with d
    - For each hypothesis s in S that is not consistent with d
        - Remove s from S
        - Add to S all minimal generalizations h of s such that
            - h is consistent with d, and some member of G is more general than h
        - Remove from S any hypothesis that is more general than another hypothesis in S
- If d is a negative example
    - Remove from S any hypothesis inconsistent with d
    - For each hypothesis g in G that is not consistent with d
        - Remove g from G
        - Add to G all minimal specializations h of g such that
        - h is consistent with d, and some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

DATASET:

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

PROGRAM:

```
import numpy as np
import pandas as pd
data=pd.DataFrame(data=pd.read_csv('FIND-S.csv'))
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(specific_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")
```

OUTPUT:

```
[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
['Yes' 'Yes' 'No' 'Yes']
initialization of specific_h and general_h
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
steps of Candidate Elimination Algorithm 1
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
steps of Candidate Elimination Algorithm 2
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
steps of Candidate Elimination Algorithm 3
```

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' '?' 'Strong' '?' 'Same']

['Sunny' 'Warm' '?' 'Strong' '?' '?']

['Sunny' 'Warm' '?' 'Strong' '?' '?']

steps of Candidate Elimination Algorithm 4

['Sunny' 'Warm' '?' 'Strong' '?' '?']

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:

['Sunny' 'Warm' '?' 'Strong' '?' '?']

Final General_h:

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

RESULT: The program was implemented successfully

EXP NO: 3 DATE:	DECISION TREE BASED ID3 ALGORITHM

AIM:

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.

ALGORITHM:

1.Create a Root node for the tree

2.If all Examples are positive, Return the single-node tree Root, with label = "+"

3.If all Examples are negative, Return the single-node tree Root, with label = "-"

4.If Attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute in Examples

5.Otherwise, Begin

6. A ← the attribute from Attributes that best* classifies Examples

7.The decision attribute for Root ← A

8.For each possible value, vi, of A,

- Add a new tree branch below Root, corresponding to the test A = vi
- Let Examples vi, be the subset of Examples that have value vi for A

- If Examples vi , is empty
  - Then below this new branch add a leaf node with label = most common value of 9.Target_attribute in Examples
- Else below this new branch add the subtree ID3(Examples vi, Target_attribute, 10.Attributes – {A}))
- End

11.Return Root

DATASET:

Training Dataset:

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Test Dataset:

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*
T1	Rain	Cool	Normal	Strong
T2	Sunny	Mild	Normal	Strong

PROGRAM:

```
import math
import csv

def load_csv(filename):
    lines = csv.reader(open(filename, "r"))
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset, headers

class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

def subtables(data, col, delete):
    dic = {}
    coldata = [row[col] for row in data]
    attr = list(set(coldata))
    counts = [0] * len(attr)
    r = len(data)
    c = len(data[0])

    for x in range(len(attr)):
        for y in range(r):
            if data[y][col] == attr[x]:
                counts[x] += 1

    for x in range(len(attr)):
        dic[attr[x]] = [[0 for i in range(c)] for j in range(counts[x])]
        pos = 0
        for y in range(r):
```

```
            if data[y][col] == attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos] = data[y]
                pos += 1
    return attr, dic

def entropy(S):
    attr = list set(S)
    if len(attr) == 1:
        return 0
    counts = [0, 0]

    for i in range(2):
        counts[i] = sum([1 for x in S if attr[i] == x]) / (len(S) * 1.0)

    sums = 0
    for cnt in counts:
        sums += -1 * cnt * math.log(cnt, 2)

    return sums

def compute_gain(data, col):
    attr, dic = subtables(data, col, delete=False)
    total_size = len(data)
    entropies = [0] * len(attr)
    ratio = [0] * len(attr)
    total_entropy = entropy([row[-1] for row in data])

    for x in range(len(attr)):
        ratio[x] = len(dic[attr[x]]) / (total_size * 1.0)
        entropies[x] = entropy([row[-1] for row in dic[attr[x]]])
        total_entropy -= ratio[x] * entropies[x]
```

```
    return total_entropy

def build_tree(data, features):
    lastcol = [row[-1] for row in data]
    if len(set(lastcol)) == 1:
        node = Node("")
        node.answer = lastcol[0]
        return node

    n = len(data[0]) - 1
    gains = [0] * n

    for col in range(n):
        gains[col] = compute_gain(data, col)

    split = gains.index(max(gains))
    node = Node(features[split])
    fea = features[:split] + features[split + 1:]
    attr, dic = subtables(data, split, delete=True)

    for x in range(len(attr)):
        child = build_tree(dic[attr[x]], fea)
        node.children.append((attr[x], child))

    return node

def print_tree(node, level):
    if node.answer != "":
        print(" " * level, node.answer)
        return
    print(" " * level, node.attribute)

    for value, n in node.children:
        print(" " * (level + 1), value)
```

```
        print_tree(n, level + 2)

def classify(node, x_test, features):
    if node.answer != "":
        print(node.answer)
        return

    pos = features.index(node.attribute)

    for value, n in node.children:
        if x_test[pos] == value:
            classify(n, x_test, features)

'''Main program'''
dataset, features = load_csv("data3.csv")
node1 = build_tree(dataset, features)
print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1, 0)

testdata, features = load_csv("data3_test.csv")
for xtest in testdata:
    print("The test instance:", xtest)
    print("The label for the test instance:", end=" ")
    classify(node1, xtest, features)
```

OUTPUT:

The decision tree for the dataset using ID3 algorithm is

The test instance: ['rain', 'cool', 'normal', 'strong']

The label for test instance: no

The test instance: ['sunny', 'mild', 'normal', 'strong']

The label for test instance: yes

RESULT: The program was implemented successfully

EXP NO: 4 DATE:	BACKPROPAGATION

AIM:

Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

ALGORITHM:

1. Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.
2. Initialize all network weights to small random numbers.
3. Until the termination condition is met, Do:
   a) For each ($\vec{}$ →, →) in training examples, Do:
      i. Propagate the input forward through the network:
         1. Input the instance $\vec{}$ →, to the network and compute the output ou of every unit u in the network.
      ii. Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit *h*, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta \delta_j x_{i,j}$$

DATASET:

Training Examples:

Example	Sleep	Study	Expected % in Exams
1	2	9	92
2	1	5	86
3	3	6	89

Normalize the input

Example	Sleep	Study	Expected % in Exams
1	2/3 = 0.66666667	9/9 = 1	0.92
2	1/3 = 0.33333333	5/9 = 0.55555556	0.86
3	3/3 = 1	6/9 = 0.66666667	0.89

PROGRAM:

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X, axis=0)  # maximum of X array longitudinally
y = y/100

# Sigmoid Function
def sigmoid(x):
   return 1/(1 + np.exp(-x))

# Derivative of Sigmoid Function
def derivatives_sigmoid(x):
   return x * (1 - x)

# Variable initialization
epoch = 5000  # Setting training iterations
lr = 0.1  # Setting learning rate
inputlayer_neurons = 2  # number of features in data set
hiddenlayer_neurons = 3  # number of hidden layers neurons
output_neurons = 1  # number of neurons at output layer

# weight and bias initialization
wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))
bh = np.random.uniform(size=(1, hiddenlayer_neurons))
wout = np.random.uniform(size=(hiddenlayer_neurons, output_neurons))
bout = np.random.uniform(size=(1, output_neurons))

# draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
   # Forward Propagation
```

```
    hinp1 = np.dot(X, wh)
    hinp = hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1 = np.dot(hlayer_act, wout)
    outinp = outinp1 + bout
    output = sigmoid(outinp)

    # Backpropagation
    EO = y - output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)

    # how much hidden layer wts contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad

    # dot product of next layer error and current layer op
    wout += hlayer_act.T.dot(d_output) * lr
    wh += X.T.dot(d_hiddenlayer) * lr

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n", output)
```

OUTPUT:

Input:

[[0.66666667 1.]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92] [0.86] [0.89]]

Predicted Output:

[[0.89726759] [0.87196896] [0.9000671]

RESULT: The program was implemented successfully

EXP NO: 5 DATE:	NAIVE BAYESIAN CLASSIFIER

AIM:

Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

ALGORITHM:

Bayes' Theorem is stated as:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

P(h|D): The probability of hypothesis h given the data D. This is called the posterior probability.

P(D|h): The probability of data D given that the hypothesis h was true.

P(h): The probability of hypothesis h being true. This is called the prior probability of h.

P(D): The probability of the data. This is called the prior probability of D.

MAP Hypothesis (hMAP): After calculating the posterior probability for a number of different hypotheses h, one is interested in finding the most probable hypothesis h ∈ H given the observed data D. Any such maximally probable hypothesis is called a maximum a posteriori (MAP) hypothesis.

Bayes' Theorem: Bayes' theorem is used to calculate the posterior probability of each candidate hypothesis. The MAP hypothesis (hMAP) is a MAP hypothesis provided by Bayes' theorem.

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

$$= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)}$$

$$= \arg\max_{h \in H} P(D|h)P(h)$$

PROGRAM:

```
import csv
import random
import math

def loadcsv(filename):
   lines = csv.reader(open(filename, "r"))
   dataset = list(lines)
   for i in range(len(dataset)):
      # Converting strings into numbers for processing
      dataset[i] = [float(x) for x in dataset[i]]
   return dataset

def splitdataset(dataset, splitratio):
   # 67% training size
   trainsize = int(len(dataset) * splitratio)
   trainset = []
   copy = list(dataset)
   while len(trainset) < trainsize:
      index = random.randrange(len(copy))
      trainset.append(copy.pop(index))
   return [trainset, copy]

def separatebyclass(dataset):
   separated = {}  # Dictionary of classes 1 and 0
   for i in range(len(dataset)):
      vector = dataset[i]
      if vector[-1] not in separated:
         separated[vector[-1]] = []
      separated[vector[-1]].append(vector)
   return separated

def mean(numbers):
```

```
    return sum(numbers) / float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)
    return math.sqrt(variance)

def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
    del summaries[-1]  # Excluding labels +ve or -ve
    return summaries

def summarizebyclass(dataset):
    separated = separatebyclass(dataset)
    summaries = {}
    for classvalue, instances in separated.items():
        summaries[classvalue] = summarize(instances)
    return summaries

def calculateprobability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * exponent

def calculateclassprobabilities(summaries, inputvector):
    probabilities = {}
    for classvalue, classsummaries in summaries.items():
        probabilities[classvalue] = 1
        for i in range(len(classsummaries)):
            mean, stdev = classsummaries[i]
            x = inputvector[i]
            probabilities[classvalue] *= calculateprobability(x, mean, stdev)
    return probabilities

def predict(summaries, inputvector):
```

```
    probabilities = calculateclassprobabilities(summaries, inputvector)
    bestLabel, bestProb = None, -1
    for classvalue, probability in probabilities.items():
      if bestLabel is None or probability > bestProb:
        bestProb = probability
        bestLabel = classvalue
    return bestLabel

  def getpredictions(summaries, testset):
    predictions = []
    for i in range(len(testset)):
      result = predict(summaries, testset[i])
      predictions.append(result)
    return predictions

  def getaccuracy(testset, predictions):
    correct = 0
    for i in range(len(testset)):
      if testset[i][-1] == predictions[i]:
        correct += 1
    return (correct / float(len(testset))) * 100.0

  def main():
    filename = 'naivedata.csv'
    splitratio = 0.67
    dataset = loadcsv(filename)
    trainingset, testset = splitdataset(dataset, splitratio)
    print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingset),
len(testset)))

    # Prepare model
    summaries = summarizebyclass(trainingset)

    # Test model
```

```
    predictions = getpredictions(summaries, testset)
    accuracy = getaccuracy(testset, predictions)
    print('Accuracy of the classifier is: {0}%'.format(accuracy))

main()
```

OUTPUT:

Split 768 rows into train=514 and test=254 rows

Accuracy of the classifier is : 71.65354330708661%

RESULT: The program was implemented successfully

EXP NO: 6

DATE:

NAIVE BAYESIAN CLASSIFIER MODEL

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:

1. Collect all words, punctuation, and other tokens that occur in Examples:

Vocabulary ← The set of all distinct words and other tokens occurring in any text document from Examples.

2. Calculate the required P(vj) and P(wk|vj) probability terms:

For each target value vj in V:

docsj ← The subset of documents from Examples for which the target value is vj.

P(vj) ← | docsj | / |Examples|.

Textj ← A single document created by concatenating all members of docsj.

n ← The total number of distinct word positions in Textj.

For each word wk in Vocabulary:

nk ← The number of times word wk occurs in Textj.

P(wk|vj) ← (nk + 1) / (n + | Vocabulary|).

3. positions ← all word positions in Doc that contain tokens found in Vocabulary

4. Return VNB, where

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_{i \in positions} P(a_i|v_j)$$

DATASET:

	Text Documents	Label
1	I love this sandwich	pos
2	This is an amazing place	pos
3	I feel very good about these beers	pos
4	This is my best work	pos
5	What an awesome view	pos
6	I do not like this restaurant	neg
7	I am tired of this stuff	neg
8	I can't deal with this	neg
9	He is my sworn enemy	neg
10	My boss is horrible	neg
11	This is an awesome place	pos
12	I do not like the taste of this juice	neg
13	I love to dance	pos
14	I am sick and tired of this place	neg
15	What a great holiday	pos
16	That is a bad locality to stay	neg
17	We will have good fun tomorrow	pos
18	I went to my enemy's house today	neg

PROGRAM:

```
import pandas as pd

msg = pd.read_csv('naivetext.csv', names=['message', 'label'])
print('The dimensions of the dataset', msg.shape)

msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})
X = msg.message
y = msg.labelnum

print(X)
print(y)

# Splitting the dataset into train and test data
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(X, y)

print('\n The total number of Training Data :', ytrain.shape)
print('\n The total number of Test Data :', ytest.shape)

# Output of count vectorizer is a sparse matrix
from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm = count_vect.transform(xtest)

print('\n The words or Tokens in the text documents \n')
print(count_vect.get_feature_names())

df = pd.DataFrame(xtrain_dtm.toarray(), columns=count_vect.get_feature_names())

# Training Naive Bayes (NB) classifier on training data.
```

```
from sklearn.naive_bayes import MultinomialNB

clf = MultinomialNB().fit(xtrain_dtm, ytrain)
predicted = clf.predict(xtest_dtm)

# Printing accuracy, Confusion matrix, Precision, and Recall
from sklearn import metrics

print('\n Accuracy of the classifier is', metrics.accuracy_score(ytest, predicted))
print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest, predicted))
print('\n The value of Precision', metrics.precision_score(ytest, predicted))
print('\n The value of Recall', metrics.recall_score(ytest, predicted))
```

OUTPUT:

The dimensions of the dataset (18, 2)

0 I love this sandwich

1 This is an amazing place

2 I feel very good about these beers

3 This is my best work

4 What an awesome view

5 I do not like this restaurant

6 I am tired of this stuff

7 I can't deal with this

8 He is my sworn enemy

9 My boss is horrible

10 This is an awesome place

11 I do not like the taste of this juice

12 I love to dance

13 I am sick and tired of this place

14 What a great holiday

15 That is a bad locality to stay

16 We will have good fun tomorrow

17 I went to my enemy's house today

Name: message, dtype: object

0 1

1 1

2 1

3 1

4 1

5 0

6 0

7 0

8 0

9 0

10 1

11 0

12 1

13 0

14 1

15 0

16 1

17 0

Name: labelnum, dtype: int64

The total number of Training Data: (13,)

The total number of Test Data: (5,)

The words or Tokens in the text documents

['about', 'am', 'amazing', 'an', 'and', 'awesome', 'beers', 'best', 'can', 'deal', 'do', 'enemy', 'feel', 'fun', 'good', 'great', 'have', 'he', 'holiday', 'house', 'is', 'like', 'love', 'my', 'not', 'of', 'place', 'restaurant', 'sandwich', 'sick', 'sworn', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'view', 'we', 'went', 'what', 'will', 'with', 'work']

Accuracy of the classifier is 0.8

Confusion matrix

[[2 1] [0 2]]

The value of Precision 0.6666666666666666

The value of Recall 1.0

RESULT: The program was implemented successfully

EXP NO: 7 DATE:	BAYESIAN NETWORK

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:

1.Read the Heart Disease dataset from a CSV file.

2.Replace any missing values (represented as '?') with NaN.

3.Display sample instances and attribute names with data types.

4.Define a Bayesian Network model with the following dependencies:

- age -> heartdisease
- sex -> heartdisease
- exang -> heartdisease
- cp -> heartdisease
- heartdisease -> restecg
- heartdisease -> chol

5.Learn Conditional Probability Distributions (CPDs) using Maximum Likelihood Estimators (MLE) based on the dataset.

6.Initialize the Variable Elimination algorithm for inference.

7.Compute the probability of heart disease given specific evidence:

- restecg=1: Probability of Heart Disease.
- cp=2: Probability of Heart Disease.

8.Print the computed probabilities for heart disease given the provided evidence.

DATASET:

Instance from dataset:

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
60	1	4	130	206	0	2	132	1	2.4	2	2	7	4

PROGRAM:

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination

# Read Cleveland Heart Disease data
heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?', np.nan)

# Display the data
print('Sample instances from the dataset are given below')
print(heartDisease.head())

# Display the Attributes names and datatypes
print('\nAttributes and datatypes')
print(heartDisease.dtypes)

# Create Model - Bayesian Network
model = BayesianModel([
   ('age', 'heartdisease'),
   ('sex', 'heartdisease'),
   ('exang', 'heartdisease'),
   ('cp', 'heartdisease'),
   ('heartdisease', 'restecg'),
   ('heartdisease', 'chol')
])

# Learning CPDs using Maximum Likelihood Estimators
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)
```

```
# Inferencing with Bayesian Network
print('\nInferencing with Bayesian Network:')
HeartDisease_test_infer = VariableElimination(model)

# Computing the Probability of HeartDisease given restecg
print('\n1. Probability of HeartDisease given evidence=restecg:1')
q1 = HeartDisease_test_infer.query(variables=['heartdisease'], evidence={'restecg': 1})
print(q1)

# Computing the Probability of HeartDisease given cp
print('\n2. Probability of HeartDisease given evidence=cp:2')
q2 = HeartDisease_test_infer.query(variables=['heartdisease'], evidence={'cp': 2})
print(q2)
```

OUTPUT:

```
Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg
```

heartdisease	phi(heartdisease)
heartdisease(0)	0.1012
heartdisease(1)	0.0000
heartdisease(2)	0.2392
heartdisease(3)	0.2015
heartdisease(4)	0.4581

```
 2. Probability of HeartDisease given evidence= cp
```

heartdisease	phi(heartdisease)
heartdisease(0)	0.3610
heartdisease(1)	0.2159
heartdisease(2)	0.1373
heartdisease(3)	0.1537
heartdisease(4)	0.1321

```
================ RESTART: E:\ML Lab - 2020-21\MLLab-7\ML7.py ================
Few examples from the dataset are given below
   age  sex  cp  trestbps  chol  ...  oldpeak  slope  ca  thal  heartdisease
0   63    1   1       145   233  ...      2.3      3   0     6             0
1   67    1   4       160   286  ...      1.5      2   3     3             2
2   67    1   4       120   229  ...      2.6      2   2     7             1
3   37    1   3       130   250  ...      3.5      3   0     3             0
4   41    0   2       130   204  ...      1.4      1   0     3             0

[5 rows x 14 columns]

 Attributes and datatypes
age              int64
sex              int64
cp               int64
trestbps         int64
chol             int64
fbs              int64
restecg          int64
thalach          int64
exang            int64
oldpeak        float64
slope            int64
ca              object
thal            object
heartdisease     int64
dtype: object
```

RESULT: The program was implemented successfully

EXP NO: 8 DATE:	K-MEANS

AIM:

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:

1.Load the Iris dataset, which contains features like Sepal Length, Sepal Width, Petal Length, and Petal Width.

2.Create dataframes for features (X) and target labels (y).

3.Initialize a K-Means clustering model with a specified number of clusters (e.g., 3 clusters).

4.Fit the model to the data using the fit method.

5.Visualize the clustering results:

Plot the original classifications based on Petal features.

Plot the K-Means clustering results.

6.Standardize the feature data using the StandardScaler to have a mean of 0 and a standard deviation of 1.

7.Initialize a GMM model with a specified number of components (e.g., 40 components).

8.Fit the GMM model to the standardized feature data.

9.Plot the GMM clustering results based on Petal features.

10.Compare the clustering results of K-Means and GMM.

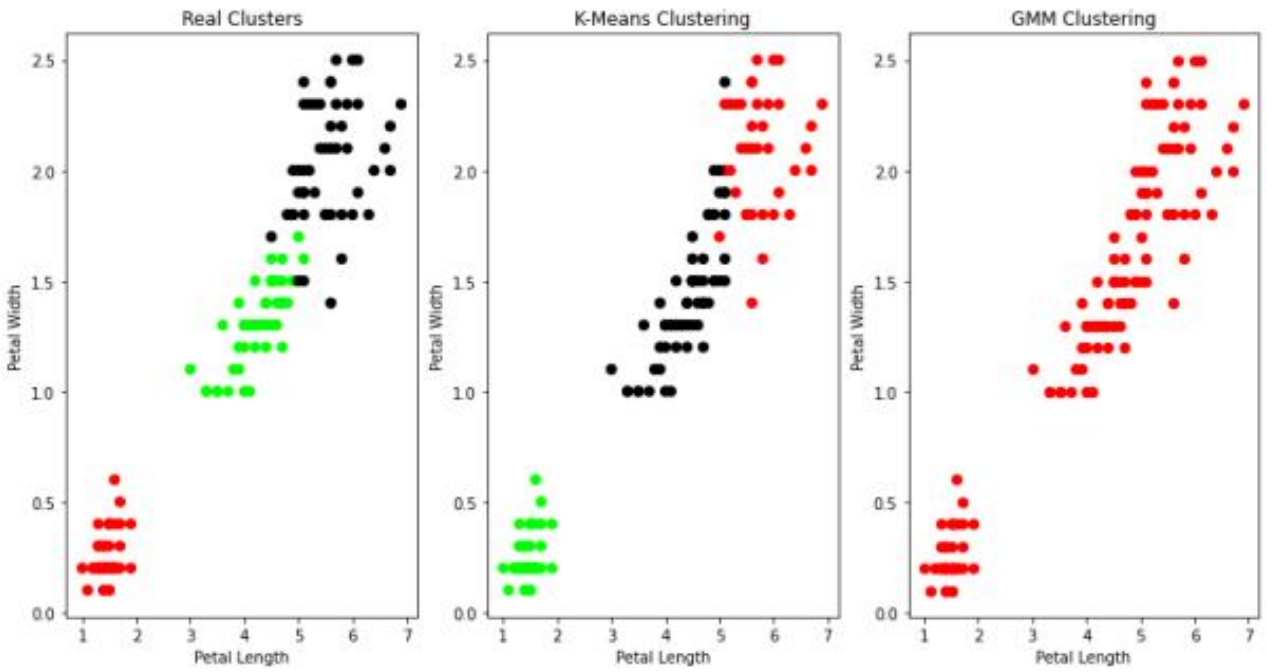
11.Note that GMM using the EM (Expectation-Maximization) algorithm matches the true labels more closely than K-Means.

PROGRAM:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

# Import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']

# Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X)  # model.labels_ : Gives cluster no for which samples belong to

# Visualize the clustering results
plt.figure(figsize=(14, 7))
colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications using Petal features
plt.subplot(1, 3, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# Plot the Models Classifications
plt.subplot(1, 3, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
```

```
plt.ylabel('Petal Width')

# General EM for GMM
from sklearn import preprocessing

# Transform your data such that its distribution will have a mean value of 0 and standard deviation of 1.
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns=X.columns)

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=40)
gmm.fit(xs)

plt.subplot(1, 3, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[0], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm-based clustering matched the true labels more closely than the Kmeans.')
```

OUTPUT:



RESULT: The program was implemented successfully

EXP NO: 9 DATE:	K-NEAREST NEIGHBOR

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:

1.For each training example (x, f (x)), add the example to the list training examples

Classification algorithm:

2.Given a query instance xq to be classified

Let x1 . . .xk denote the k instances from training examples that are nearest to xq

Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Where, f(xi) function to calculate the mean value of the k nearest training examples.

DATASET:

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

PROGRAM:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets

iris = datasets.load_iris()
x = iris.data
y = iris.target

print('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)

print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

# To Train the model and Nearest neighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

# To make predictions on our test data
y_pred = classifier.predict(x_test)

print('Confusion Matrix')
print(confusion_matrix(y_test, y_pred))

print('Accuracy Metrics')
print(classification_report(y_test, y_pred))
```

OUTPUT:

sepal-length sepal-width petal-length petal-width

[[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]

. . . . . .

[6.2 3.4 5.4 2.3]

[5.9 3. 5.1 1.8]]

class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica

[0 0 0 ……0 0 1 1 1 ……1 1 2 2 2 …… 2 2]

Confusion Matrix

Accuracy Metrics

	Precision	recall	f1-score	support
0	1.00	1.00	1.00	20
1	0.91	1.00	0.95	10
2	1.00	0.93	0.97	15

RESULT: The program was implemented successfully

EXP NO: 10 DATE:	LOCALLY WEIGHTED REGRESSION

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:

Read the Given data Sample to X and the curve (linear or non linear) to Y

Set the value for Smoothening parameter or Free parameter say τ

Set the bias /Point of interest set x0 which is a subset of X

Determine the weight matrix using :

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

Determine the value of model term parameter β using :

$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

Prediction = x0*β

DATASET:

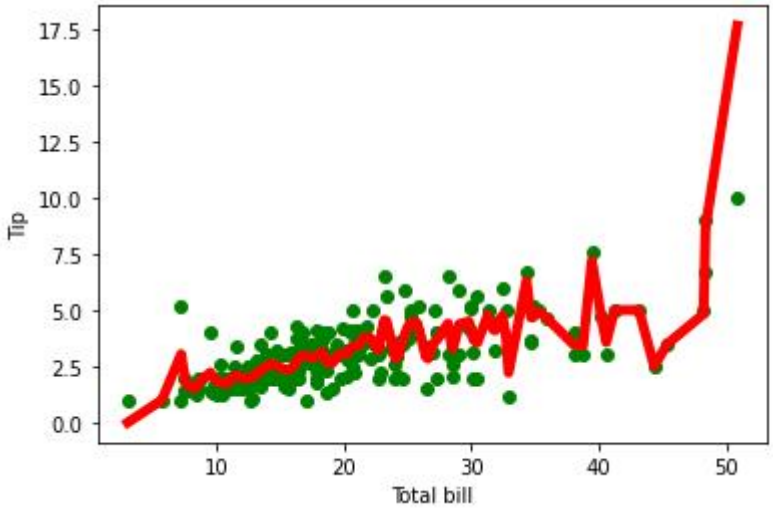
total_bill	tip	sex	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.5	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4
25.29	4.71	Male	No	Sun	Dinner	4
8.77	2	Male	No	Sun	Dinner	2
26.88	3.12	Male	No	Sun	Dinner	4
15.04	1.96	Male	No	Sun	Dinner	2
14.78	3.23	Male	No	Sun	Dinner	2
10.27	1.71	Male	No	Sun	Dinner	2
35.26	5	Female	No	Sun	Dinner	4
15.42	1.57	Male	No	Sun	Dinner	2
18.43	3	Male	No	Sun	Dinner	4
14.83	3.02	Female	No	Sun	Dinner	2
21.58	3.92	Male	No	Sun	Dinner	2
10.33	1.67	Female	No	Sun	Dinner	3
16.29	3.71	Male	No	Sun	Dinner	3
16.97	3.5	Female	No	Sun	Dinner	3
20.65	3.35	Male	No	Sat	Dinner	3
17.92	4.08	Male	No	Sat	Dinner	2
20.29	2.75	Female	No	Sat	Dinner	2
15.77	2.23	Female	No	Sat	Dinner	2
39.42	7.58	Male	No	Sat	Dinner	4

PROGRAM:

```
from numpy import *
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr

def kernel(point, xmat, k):
    m, n = np1.shape(xmat)
    weights = np1.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j, j] = np1.exp(diff * diff.T / (-2.0 * k**2))
    return weights

def localWeight(point, xmat, ymat, k):
    wei = kernel(point, xmat, k)
    W = (X.T * (wei * X)).I * (X.T * (wei * ymat.T))
    return W

def localWeightRegression(xmat, ymat, k):
    m, n = np1.shape(xmat)
    ypred = np1.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i] * localWeight(xmat[i], xmat, ymat, k)
    return ypred

# Load data points
data = pd.read_csv('tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)
```

```
# Preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip)  # mat is used to convert to n-dimensional to 2-dimensional array form
m = np1.shape(mbill)[1]

one = np1.mat(np1.ones(m))
X = np1.hstack((one.T, mbill.T))  # create a stack of bill from ONE

ypred = localWeightRegression(X, mtip, 0.3)
SortIndex = X[:, 1].argsort(0)
xsort = X[SortIndex][:, 0]

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.scatter(bill, tip, color='green')
ax.plot(xsort[:, 1], ypred[SortIndex], color='red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

OUTPUT:



RESULT: The program was implemented successfully