CITY

ENGINEERING COLLEGE

(Doddakallsandra, off. Kanakapura Road, Bangalore – 560061)

NAAC ACCREDITED

Affiliated to Visvesvaraya Technological University, Belagavi

Approved by AICTE, New Delhi

LAB MANUAL

MACHINE LEARNING LABORATORY

(Subject Code: 17CSL76)

For

VII SEMESTER

NAME: ______________________________

USN: ______________________________

SEMESTER & SECTION: ____________________

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ACADEMIC YEAR: 2019-2020

CITY ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VISION

To contribute to Global Development by producing Knowledgeable and Quality professionals who are Innovative and Successful in advanced field of Computer Science & Engineering to adapt the changing Employment demands and social needs.

MISSION

M1: To provide Quality Education for students, to build Confidence by developing their Technical Skills to make them Competitive Computer Science Engineers.

M2: To facilitate Innovation & Research for students and faculty and to provide Internship opportunities

M3: To Collaborate with educational institutions and industries for Excellence in Teaching and Research.

MACHINE LEARNING LABORATORY
[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2017 - 2018)
SEMESTER – VII

Subject Code	17CSL76	IA Marks	40
Number of Lecture Hours/Week	01I + 02P	Exam Marks	60
Total Number of Lecture Hours	40	Exam Hours	03

CREDITS – 02

Description (If any):

1. The programs can be implemented in either JAVA or Python.
2. For Problems 1 to 6 and 10, programs are to be developed without using the built-in classes or APIs of Java/Python.
3. Data sets can be taken from standard repositories (https://archive.ics.uci.edu/ml/datasets.html) or constructed by the students.

Lab Experiments:

1. Implement and demonstratethe **FIND-Salgorithm** for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.
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.
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 toclassify a new sample.
4. Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. Implement the non-parametric **Locally Weighted Regressionalgorithm** in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Study Experiment / Project:

NIL

Course outcomes: The students should be able to:

1. Understand the implementation procedures for the machine learning algorithms.

2. Design Java/Python programs for various Learning algorithms. 3. Apply appropriate data sets to the Machine Learning algorithms. 4. Identify and apply Machine Learning algorithms to solve real world problems.
Conduction of Practical Examination:
• All laboratory experiments are to be included for practical examination. • Students are allowed to pick one experiment from the lot. • Strictly follow the instructions as printed on the cover page of answer script • Marks distribution: Procedure + Conduction + Viva:**15 + 70 +15 (100)** **Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.**

1. **LAB PROGRAM:** 1

2. **TITLE: FIND-S ALGORITHM**
3. **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.

4. **Find-S Algorithm**

```
1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
       For each attribute constraint a i in h :
           If the constraint a i in h is satisfied by x then do nothing
           Else replace a i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h
```

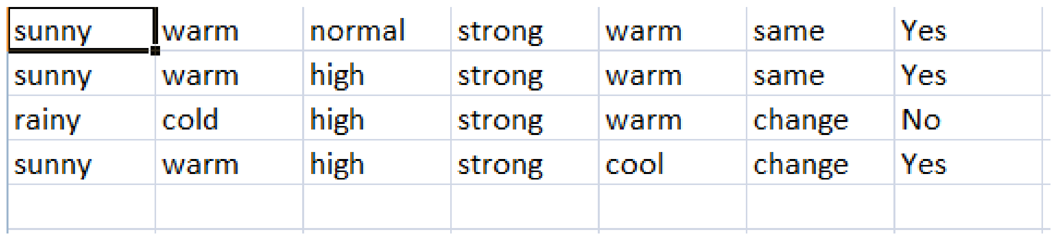
5. Implementation/ Program 1:

```
import csv
num_attribute=6
a=[]
with open('enjoysport.csv', 'r') as csvfile:
    reader=csv.reader(csvfile)
    for row in reader:
        a.append(row)
        print(row)
print("\n The total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\n The initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)
for j in range(0,num_attribute):
    hypothesis[j]=a[0][j]
print("\n Find-S: Finding 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("\n For training Example No:{0} the hypothesis is".format(i),hypothesis)
print("\n The Maximally specific hypothesis for the training instance is ")
print(hypothesis)
```

6. Result/Output:

```
['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 total number of training instances are :  4

 The initial hypothesis is :
['0', '0', '0', '0', '0', '0']

 Find-S: Finding 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 the training instance is
['sunny', 'warm', '?', 'strong', '?', '?']
```

Training Data Set : enjoysport.csv



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

1. **LAB PROGRAM:** 2

2. **TITLE:** Candidate-Elimination algorithm
3. **AIM:**
   - 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.

4. **Candidate-Elimination algorithm:**

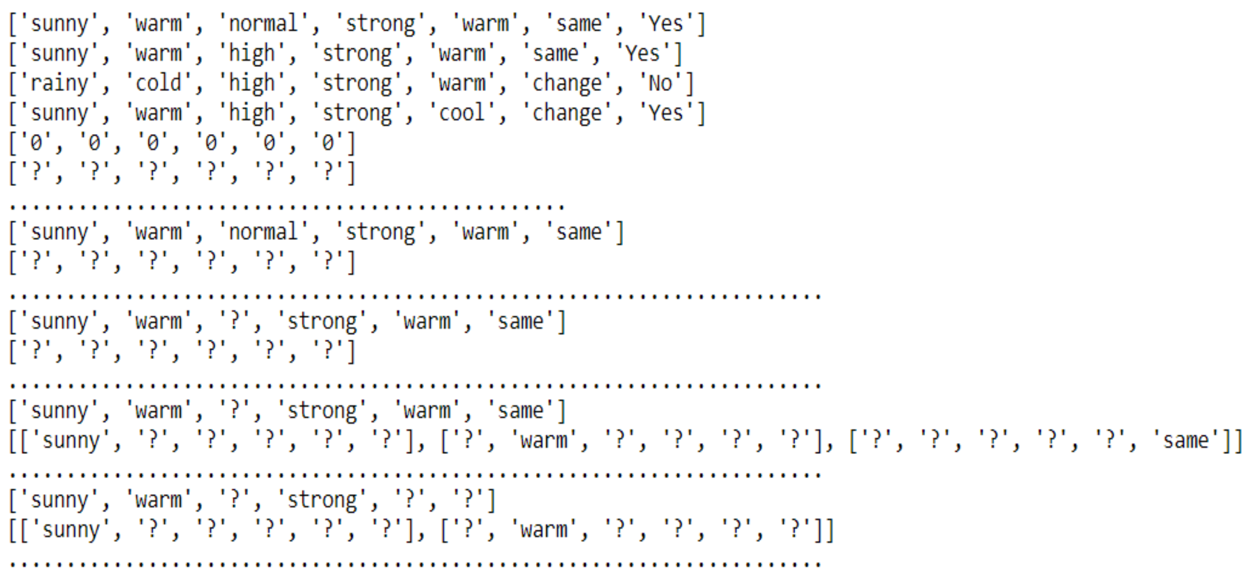
```
Initialize G to the set of maximally general hypotheses in H
1. Initialize S to the set of maximally specific hypotheses in H
2. For each training example d, do
        2.1. If d is a positive example
                Remove from G any hypothesis inconsistent with d , For
                each hypothesis 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, hypothesis that is more general than another hypothesis in S
        2.2.  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
```

5. Implementation/Program2:

```
import csv
a=[]
with open("enjoysport.csv","r") as csvfile:
    fdata=csv.reader(csvfile)
    for row in fdata:
        a.append(row)
        print(row)
num_att=len(a[0])-1
S=['0']*num_att
G=['?']*num_att
print(S)
print(G)
temp=[]
for i in range(0,num_att):
    S[i]=a[0][i]
print("............................................")
for i in range(0,len(a)):
```

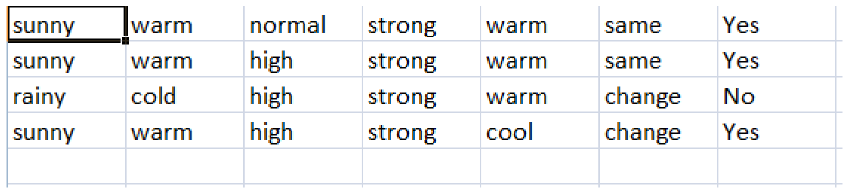
```
if a[i][num_att]=="Yes":
    for j in range(0,num_att):
        if S[j]!=a[i][j]:
               S[j]='?'
    for j in range(0,num_att):
          for k in range(0,len(temp)):
                if temp[k][j]!=S[j] and temp[k][j]!='?':
                    del temp[k]
if a[i][num_att]=='No':
    for j in range(0,num_att):
        if a[i][j]!=S[j] and S[j]!='?':
            G[j]=S[j]
            temp.append(G)
            G=['?']*num_att
print(S)
if len(temp)==0:
    print(G)
else:
    print(temp)
print("...............................................................")
```

6. Result/Output:



```
['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']
['0', '0', '0', '0', '0', '0']
['?', '?', '?', '?', '?', '?']
...............................................
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
['?', '?', '?', '?', '?', '?']
.....................................................................
['sunny', 'warm', '?', 'strong', 'warm', 'same']
['?', '?', '?', '?', '?', '?']
.....................................................................
['sunny', 'warm', '?', 'strong', 'warm', 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
.....................................................................
['sunny', 'warm', '?', 'strong', '?', '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
.....................................................................
```

Training Data Set : enjoysport.csv



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

1. Lab Program : 3

2. **TITLE: ID3 ALGORITHM**

3. **AIM:**

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

4. ID3 algorithm:

```
Algorithm: ID3(Examples, TargetAttribute, Attributes) Input:
  Examples are the training examples.
        Targetattribute is the attribute whose value is to be predicted by the tree.
        Attributes is a list of other attributes that may be tested by the learned decision tree.
  Output: Returns a decision tree that correctly classiJies the given Examples Method:
    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
       TargetAttribute in Examples
    Else
       A ← the attribute from Attributes that best classifies Examples The
       decision attribute for Root ←A
       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 TargetAttribute in Examples
       Else
         below this new branch add the subtree ID3(Examples_vi, TargetAttribute, Attributes–{A})
  End
Return Root
```

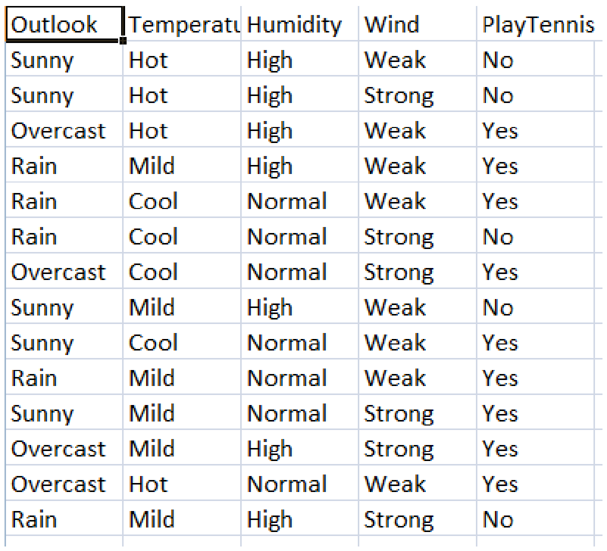
5. Implementation/Program:

```
import pandas as pd
import math

df = pd.read_csv('/Users/Chachu/Documents/Python Scripts/PlayTennis.csv')
print("\n Input Data Set is:\n", df)

t = df.keys()[-1]
print('Target Attribute is: ', t)
# Get the attribute names from input dataset
attribute_names = list(df.keys())
#Remove the target attribute from the attribute names list
attribute_names.remove(t)
print('Predicting Attributes: ', attribute_names)
#Function to calculate the entropy of collection S
def entropy(probs):
    return sum( [-prob*math.log(prob, 2) for prob in probs])
#Function to calulate the entropy of the given Data Sets/List with
#respect to target attributes
def entropy_of_list(ls,value):
    from collections import Counter
    cnt = Counter(x for x in ls)# Counter calculates the propotion of class
    print('Target attribute class count(Yes/No)=',dict(cnt))
    total_instances = len(ls)
    print("Total no of instances/records associated with {0} is: {1}".format(value,total_instances ))
    probs = [x / total_instances for x in cnt.values()]  # x means no of YES/NO
    print("Probability of Class {0} is: {1:.4f}".format(min(cnt),min(probs)))
    print("Probability of Class {0} is: {1:.4f}".format(max(cnt),max(probs)))
    return entropy(probs) # Call Entropy

def information_gain(df, split_attribute, target_attribute,battr):
    print("\n\n-----Information Gain Calculation of ",split_attribute, " --------")
    df_split = df.groupby(split_attribute) # group the data based on attribute values
    glist=[]
    for gname,group in df_split:
        print('Grouped Attribute Values \n',group)
        glist.append(gname)

    glist.reverse()
    nobs = len(df.index) * 1.0
    df_agg1=df_split.agg({target_attribute:lambda x:entropy_of_list(x, glist.pop())})
    df_agg2=df_split.agg({target_attribute :lambda x:len(x)/nobs})

    df_agg1.columns=['Entropy']
    df_agg2.columns=['Proportion']

    # Calculate Information Gain:
    new_entropy = sum( df_agg1['Entropy'] * df_agg2['Proportion'])
    if battr !='S':
        old_entropy = entropy_of_list(df[target_attribute],'S-'+df.iloc[0][df.columns.get_loc(battr)])
    else:
        old_entropy = entropy_of_list(df[target_attribute],battr)
    return old_entropy - new_entropy



def id3(df, target_attribute, attribute_names, default_class=None,default_attr='S'):

    from collections import Counter
    cnt = Counter(x for x in df[target_attribute])# class of YES /NO

    ## First check: Is this split of the dataset homogeneous?
    if len(cnt) == 1:
        return next(iter(cnt))  # next input data set, or raises StopIteration when EOF is hit.

    ## Second check: Is this split of the dataset empty? if yes, return a default value
    elif df.empty or (not attribute_names):
        return default_class  # Return None for Empty Data Set

    ## Otherwise: This dataset is ready to be devied up!
    else:
        # Get Default Value for next recursive call of this function:
        default_class = max(cnt.keys()) #No of YES and NO Class
        # Compute the Information Gain of the attributes:
        gainz=[]
        for attr in attribute_names:
            ig= information_gain(df, attr, target_attribute,default_attr)
            gainz.append(ig)
            print('Information gain of ',attr,' is : ',ig)

        index_of_max = gainz.index(max(gainz))
        best_attr = attribute_names[index_of_max
        print("\nAttribute with the maximum gain is: ", best_attr)
        # Create an empty tree, to be populated in a moment
        tree = {best_attr:{}} # Initiate the tree with best attribute as a node
        remaining_attribute_names =[i for i in attribute_names if i != best_attr]

        # Split dataset-On each split, recursively call this algorithm.Populate the empty tree with
subtrees, which
        # are the result of the recursive call
        for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3(data_subset,target_attribute,
remaining_attribute_names,default_class,best_attr)
            tree[best_attr][attr_val] = subtree
        return tree

    from pprint import pprint
tree = id3(df,t,attribute_names)
print("\nThe Resultant Decision Tree is:")
print(tree)

def classify(instance, tree,default=None): # Instance of Play Tennis with Predicted
    attribute = next(iter(tree)) # Outlook/Humidity/Wind
    if instance[attribute] in tree[attribute].keys(): # Value of the attributs in  set of Tree keys
        result = tree[attribute][instance[attribute]]
        if isinstance(result, dict): # this is a tree, delve deeper
            return classify(instance, result)
        else:
            return result # this is a label
    else:
        return default

df_new=pd.read_csv('/Users/Chachu/Documents/Python Scripts/PlayTennisTest.csv')
df_new['predicted'] = df_new.apply(classify, axis=1, args=(tree,'?'))
print(df_new)
```

6. Result/Output:

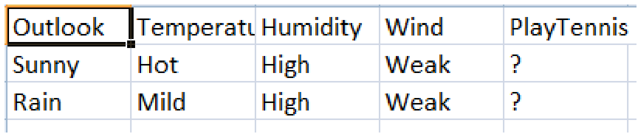
```
 Input Data Set is:
      Outlook Temperature Humidity    Wind PlayTennis
0       Sunny         Hot     High    Weak         No
1       Sunny         Hot     High  Strong         No
2    Overcast         Hot     High    Weak        Yes
3        Rain        Mild     High    Weak        Yes
4        Rain        Cool   Normal    Weak        Yes
5        Rain        Cool   Normal  Strong         No
6    Overcast        Cool   Normal  Strong        Yes
7       Sunny        Mild     High    Weak         No
8       Sunny        Cool   Normal    Weak        Yes
9        Rain        Mild   Normal    Weak        Yes
10      Sunny        Mild   Normal  Strong        Yes
11   Overcast        Mild     High  Strong        Yes
12   Overcast         Hot   Normal    Weak        Yes
13       Rain        Mild     High  Strong         No
Target Attribute is:  PlayTennis
Predicting Attributes:  ['Outlook', 'Temperature', 'Humidity', 'Wind']

The Resultant Decision Tree is:
{'Outlook': {'Overcast': 'Yes', 'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}}, 'Sunny': {'Humidity': {'High': 'No', 'Norma
l': 'Yes'}}}}

 Testing Samples are :
  Outlook Temperature Humidity  Wind PlayTennis predicted
0   Sunny         Hot     High  Weak          ?        No
1    Rain        Mild     High  Weak          ?       Yes
```

Training Data Set : PlayTennis.csv



Outlook	Temperatu	Humidity	Wind	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Testing Data Set : PlayTennisTest.csv

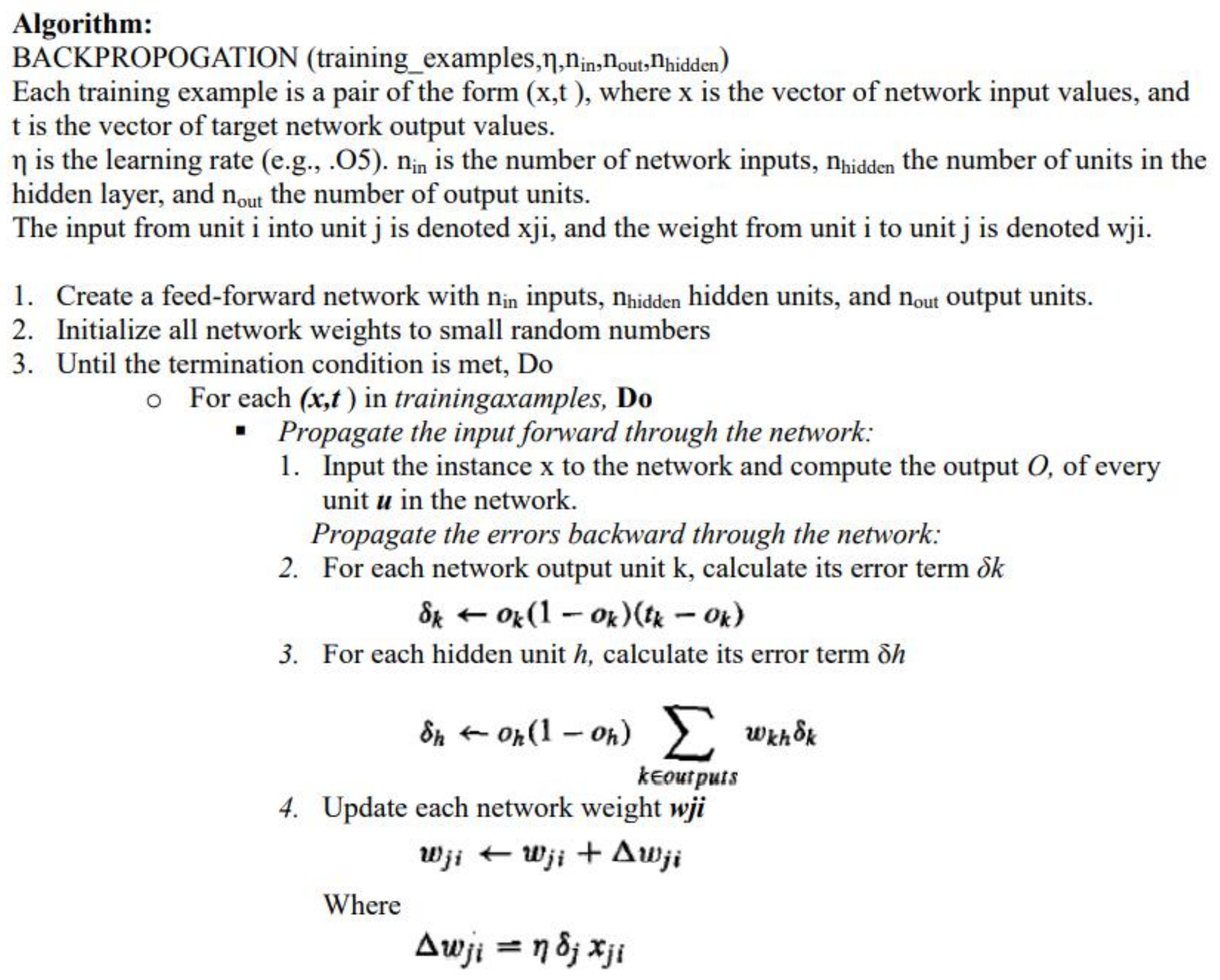


Outlook	Temperatu	Humidity	Wind	PlayTennis
Sunny	Hot	High	Weak	?
Rain	Mild	High	Weak	?

1. **LAB PROGRAM: 4**

2. **TITLE: BACKPROPAGATION ALGORITHM**
3. **AIM:**
   Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

4. **Backpropagation Algorithm:**



Algorithm:
BACKPROPOGATION (training_examples,η,n_{in},n_{out},n_{hidden})
Each training example is a pair of the form (x,t), where x is the vector of network input values, and t is the vector of target network output values.
η is the learning rate (e.g., .O5). n_{in} is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.
The input from unit i into unit j is denoted xji, and the weight from unit i to unit j is denoted wji.

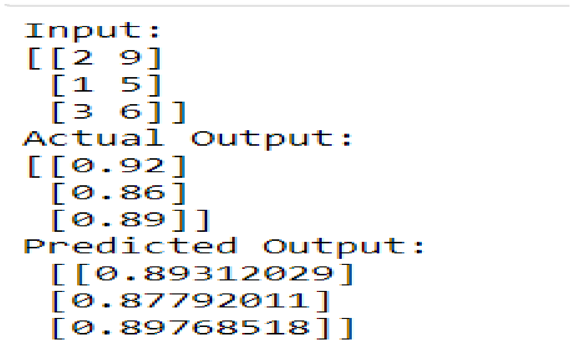
```
1. Create a feed-forward network with n_in inputs, n_hidden hidden units, and n_out output units.
2. Initialize all network weights to small random numbers
3. Until the termination condition is met, Do
      o  For each (x,t ) in trainingaxamples, Do
            ▪  Propagate the input forward through the network:
               1. Input the instance x to the network and compute the output O, of every
                  unit u in the network.
                  Propagate the errors backward through the network:
               2. For each network output unit k, calculate its error term δk
                        $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$
               3. For each hidden unit h, calculate its error term δh
                        $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$
               4. Update each network weight wji
                        $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$
                  Where
                        $\Delta w_{ji} = \eta\, \delta_j\, x_{ji}$
```

5. Implementation/ Program:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]))
y = np.array(([92], [86], [89]))
y = y/100

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

def derivatives_sigmoid(x):
    return x * (1 - x)
```

```
epoch=10000
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bias_hidden=np.random.uniform(size=(1,hiddenlayer_neurons))
weight_hidden=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bias_output=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
    hinp1=np.dot(X,wh)
    hinp= hinp1 + bias_hidden
    hlayer_activation = sigmoid(hinp)

    outinp1=np.dot(hlayer_activation,weight_hidden)
    outinp= outinp1+ bias_output
    output = sigmoid(outinp)

    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(weight_hidden.T)
    hiddengrad = derivatives_sigmoid(hlayer_activation)
    d_hiddenlayer = EH * hiddengrad

    weight_hidden += hlayer_activation.T.dot(d_output) *lr
    bias_hidden += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr

    wh += X.T.dot(d_hiddenlayer) *lr
    bias_output += np.sum(d_output, axis=0,keepdims=True) *lr

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

6. **Result/Output:**



```
Input:
[[2 9]
 [1 5]
 [3 6]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89312029]
 [0.87792011]
 [0.89768518]]
```

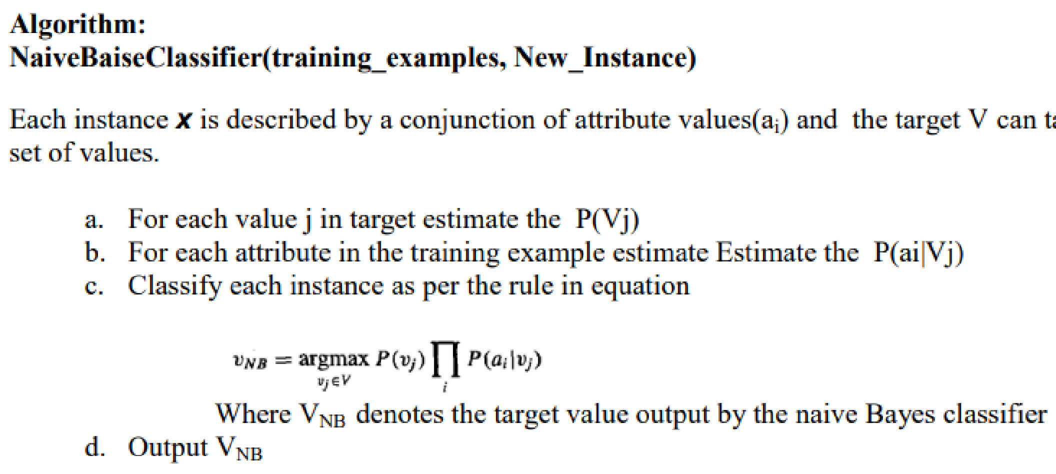
1**. LAB PROGRAM: 5**

2. TITLE: **NAÏVE BAYESIAN CLASSIFIER**

3. 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.

4. Algorithm:



Algorithm:
NaiveBaiseClassifier(training_examples, New_Instance)

Each instance ***x*** is described by a conjunction of attribute values(a_i) and the target V can take j finite set of values.

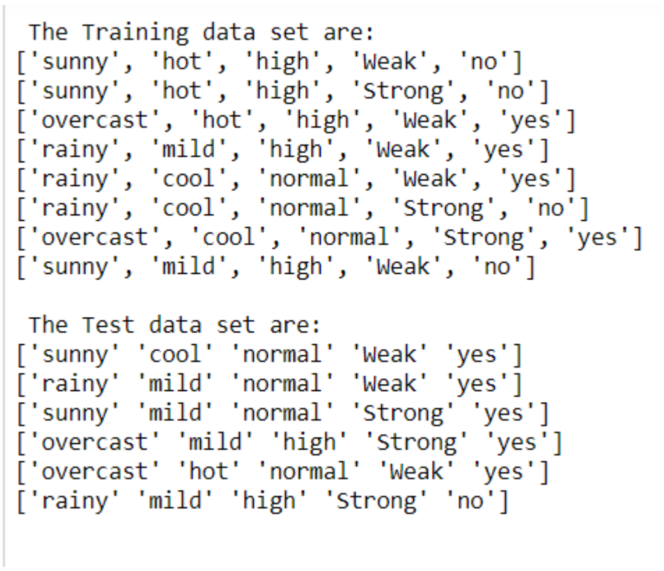
```
a. For each value j in target estimate the P(Vj)
b. For each attribute in the training example estimate Estimate the P(ai|Vj)
c. Classify each instance as per the rule in equation
       $v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i|v_j)$
       Where $V_{NB}$ denotes the target value output by the naive Bayes classifier
d. Output $V_{NB}$
```

5. Implementation/Program :

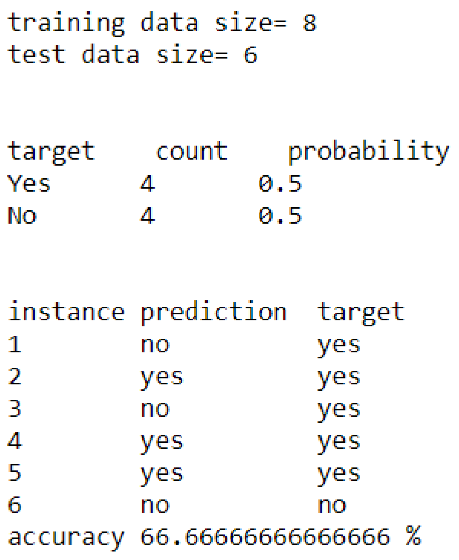
```
import numpy as np
import math
import csv
import pdb
def read_data(filename):

    with open(filename,'r') as csvfile:
        datareader = csv.reader(csvfile)
        metadata = next(datareader)
        traindata=[]
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)

def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    testset = list(dataset)
    i=0
    while len(trainSet) < trainSize:
        trainSet.append(testset.pop(i))
    return [trainSet, testset]
```

```
def classify(data,test):

    total_size = data.shape[0]
    print("\n")
    print("training data size=",total_size)
    print("test data size=",test.shape[0])

    countYes = 0
    countNo = 0
    probYes = 0
    probNo = 0
    print("\n")
    print("target    count    probability")

    for x in range(data.shape[0]):
        if data[x,data.shape[1]-1] == 'yes':
            countYes +=1
        if data[x,data.shape[1]-1] == 'no':
            countNo +=1

    probYes=countYes/total_size
    probNo= countNo / total_size

    print('Yes',"\t",countYes,"\t",probYes)
    print('No',"\t",countNo,"\t",probNo)


    prob0 =np.zeros((test.shape[1]-1))
    prob1 =np.zeros((test.shape[1]-1))
    accuracy=0
    print("\n")
    print("instance prediction  target")

    for t in range(test.shape[0]):
        for k in range (test.shape[1]-1):
            count1=count0=0
            for j in range (data.shape[0]):
                #how many times appeared with no
                if test[t,k] == data[j,k] and data[j,data.shape[1]-1]=='no':
                    count0+=1
                #how many times appeared with yes
                if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='yes':
                    count1+=1
            prob0[k]=count0/countNo
            prob1[k]=count1/countYes

        probno=probNo
        probyes=probYes
        for i in range(test.shape[1]-1):
            probno=probno*prob0[i]
            probyes=probyes*prob1[i]
        if probno>probyes:
            predict='no'
        else:
            predict='yes'

        print(t+1,"\t",predict,"\t    ",test[t,test.shape[1]-1])
        if predict == test[t,test.shape[1]-1]:
            accuracy+=1
    final_accuracy=(accuracy/test.shape[0])*100
    print("accuracy",final_accuracy,"%")
    return

metadata,traindata= read_data("/Users/Chachu/Documents/Python Scripts/tennis.csv")
splitRatio=0.6
trainingset, testset=splitDataset(traindata, splitRatio)
training=np.array(trainingset)
print("\n The Training data set are:")
for x in trainingset:
    print(x)

testing=np.array(testset)
print("\n The Test data set are:")
for x in testing:
    print(x)
classify(training,testing)
```

6. Result /Output:

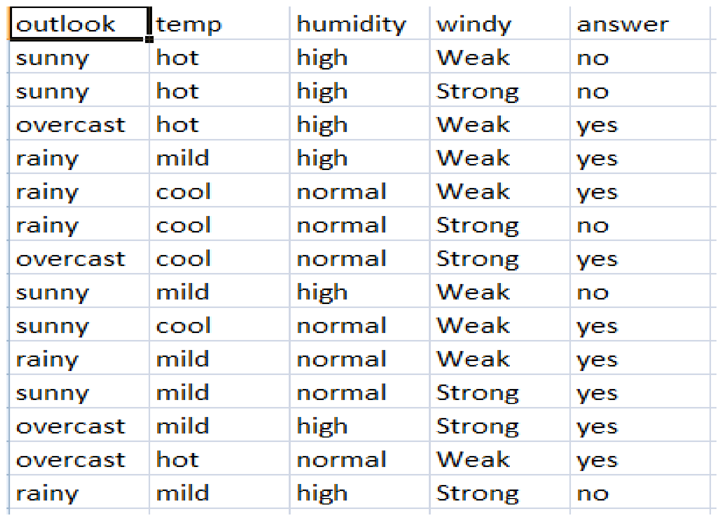


```
 The Training data set are:
['sunny', 'hot', 'high', 'Weak', 'no']
['sunny', 'hot', 'high', 'Strong', 'no']
['overcast', 'hot', 'high', 'Weak', 'yes']
['rainy', 'mild', 'high', 'Weak', 'yes']
['rainy', 'cool', 'normal', 'Weak', 'yes']
['rainy', 'cool', 'normal', 'Strong', 'no']
['overcast', 'cool', 'normal', 'Strong', 'yes']
['sunny', 'mild', 'high', 'Weak', 'no']

 The Test data set are:
['sunny' 'cool' 'normal' 'Weak' 'yes']
['rainy' 'mild' 'normal' 'Weak' 'yes']
['sunny' 'mild' 'normal' 'Strong' 'yes']
['overcast' 'mild' 'high' 'Strong' 'yes']
['overcast' 'hot' 'normal' 'Weak' 'yes']
['rainy' 'mild' 'high' 'Strong' 'no']
```



```
training data size= 8
test data size= 6


target    count    probability
Yes      4       0.5
No       4       0.5


instance prediction  target
1        no          yes
2        yes         yes
3        no          yes
4        yes         yes
5        yes         yes
6        no          no
accuracy 66.66666666666666 %
```

Training Data Set : tennis.csv

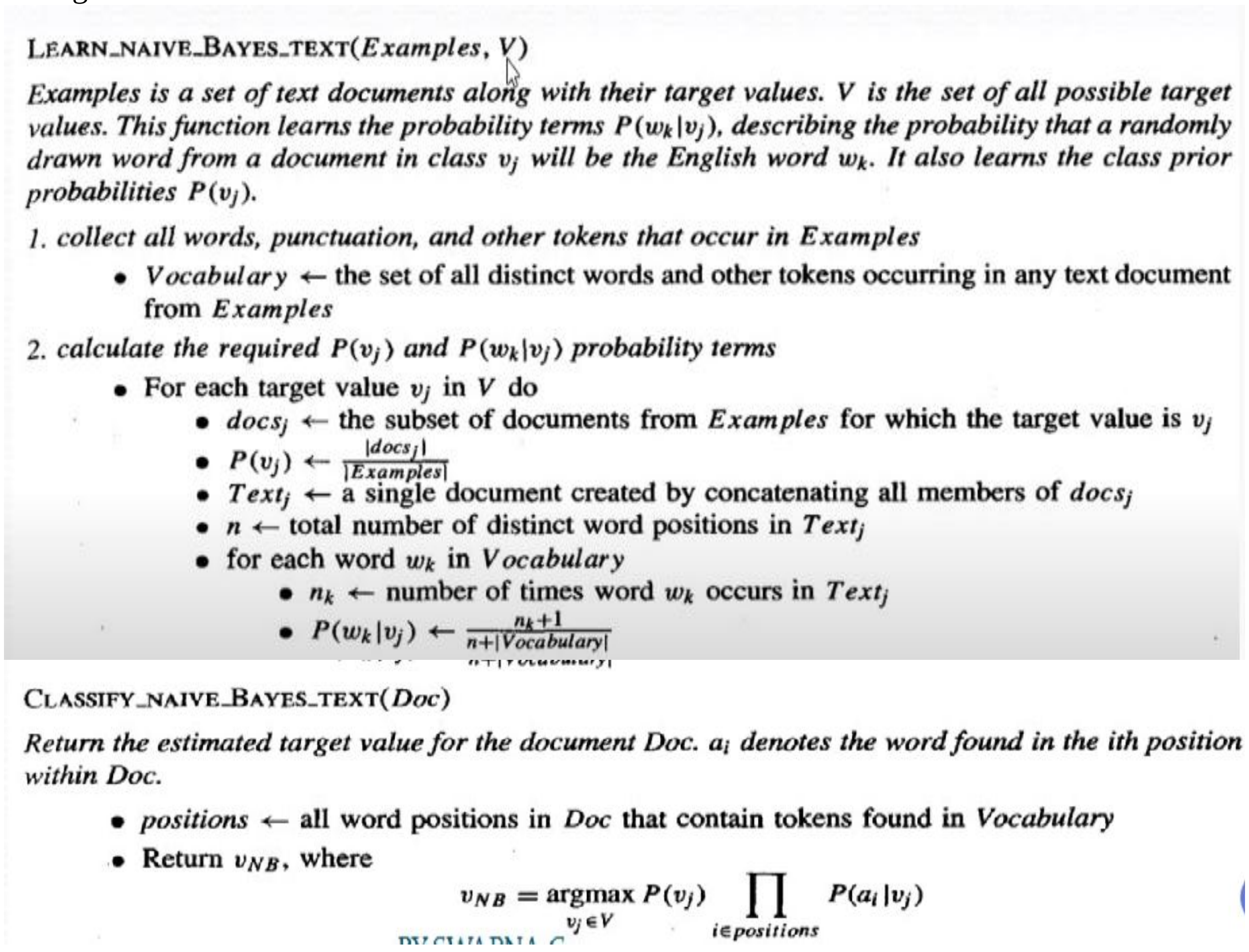


outlook	temp	humidity	windy	answer
sunny	hot	high	Weak	no
sunny	hot	high	Strong	no
overcast	hot	high	Weak	yes
rainy	mild	high	Weak	yes
rainy	cool	normal	Weak	yes
rainy	cool	normal	Strong	no
overcast	cool	normal	Strong	yes
sunny	mild	high	Weak	no
sunny	cool	normal	Weak	yes
rainy	mild	normal	Weak	yes
sunny	mild	normal	Strong	yes
overcast	mild	high	Strong	yes
overcast	hot	normal	Weak	yes
rainy	mild	high	Strong	no

1. **LAB PROGRAM: 6**

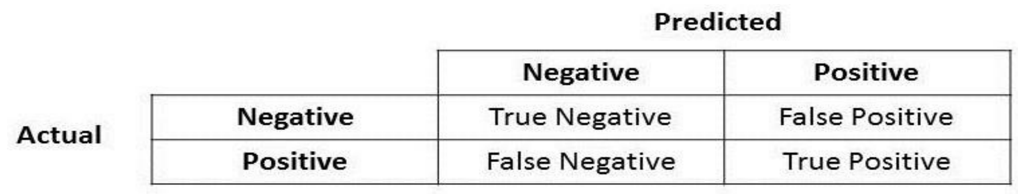
2. TITLE: **DOCUMENT CLASSIFICATION USING NAÏVE BAYESIAN CLASSIFIER**
3. AIM:
   - Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

4. Algorithm:



```
LEARN_NAIVE_BAYES_TEXT($Examples$, $V$)
Examples is a set of text documents along with their target values. V is the set of all possible target
values. This function learns the probability terms $P(w_k|v_j)$, describing the probability that a randomly
drawn word from a document in class $v_j$ will be the English word $w_k$. It also learns the class prior
probabilities $P(v_j)$.
1. collect all words, punctuation, and other tokens that occur in Examples
    • $Vocabulary \leftarrow$ the set of all distinct words and other tokens occurring in any text document
      from $Examples$
2. calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms
    • For each target value $v_j$ in $V$ do
        • $docs_j \leftarrow$ the subset of documents from $Examples$ for which the target value is $v_j$
        • $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
        • $Text_j \leftarrow$ a single document created by concatenating all members of $docs_j$
        • $n \leftarrow$ total number of distinct word positions in $Text_j$
        • for each word $w_k$ in $Vocabulary$
            • $n_k \leftarrow$ number of times word $w_k$ occurs in $Text_j$
            • $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$

CLASSIFY_NAIVE_BAYES_TEXT($Doc$)
Return the estimated target value for the document Doc. $a_i$ denotes the word found in the ith position
within Doc.
    • $positions \leftarrow$ all word positions in $Doc$ that contain tokens found in $Vocabulary$
    • Return $v_{NB}$, where
          $v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_{i \in positions} P(a_i|v_j)$
```

Analysis of Document Classification:



		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

- For classification tasks, the terms true positives, true negatives, false positives, and false negatives compare the results of the classifier under test with trusted external judgments. The terms positive and negative refer to the classifier's prediction (sometimes known as the expectation), and the terms true and false refer to whether that prediction corresponds to the external judgment (sometimes known as the observation).
- Precision - Precision is the ratio of correctly predicted positive documents to the total predicted positive documents. High precision relates to the low false positive rate.

Precision = (Σ True positive) / (Σ True positive + Σ False positive)

- Recall (Sensitivity) - Recall is the ratio of correctly predicted positive documents to the all observations in actual class.

Recall = (Σ True positive) / (Σ True positive + Σ False negative)

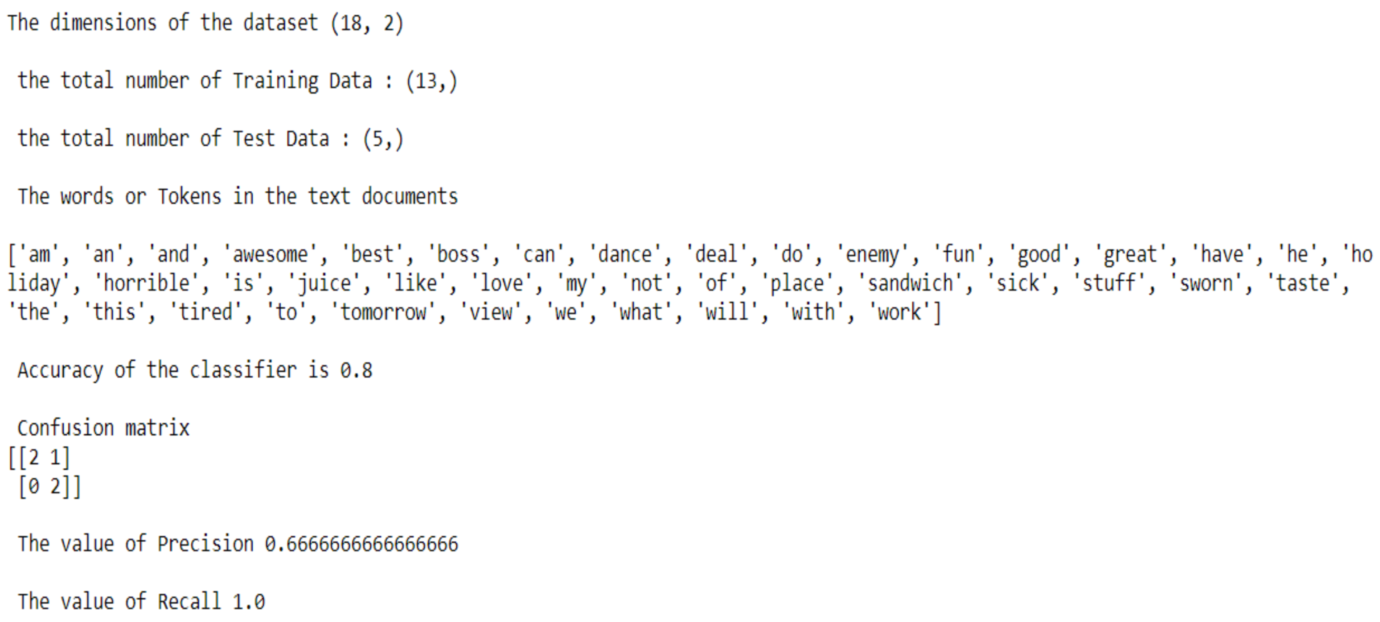
- Accuracy - Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. Therefore, you have to look at other parameters to evaluate the performance of your model. For our model, we have got 0.803 which means our model is approx. 80% accurate.

Accuracy = (Σ True positive + Σ True negative) / Σ Total population

5. Implementation/ Program:

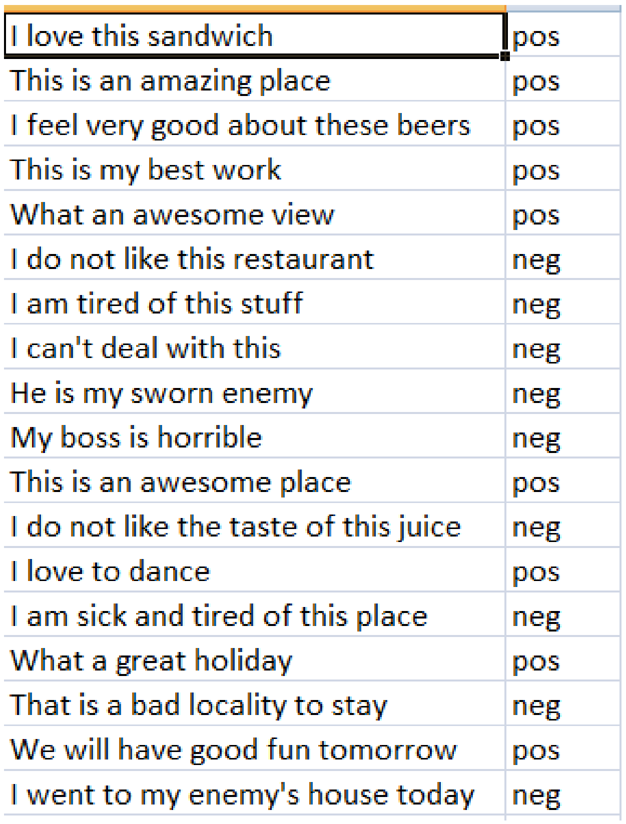
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics

msg=pd.read_csv('/Users/Chachu/Documents/PythonScripts/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

#splitting the dataset into train and test data
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 the words or Tokens in the text documents
cv = CountVectorizer()
xtrain_dtm = cv.fit_transform(xtrain)
xtest_dtm=cv.transform(xtest)
print('\n The words or Tokens in the text documents \n')
print(cv.get_feature_names())
df=pd.DataFrame(xtrain_dtm.toarray(),columns=cv.get_feature_names())
# Training Naive Bayes (NB) classifier on training data.
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)
#printing accuracy, Confusion matrix, Precision and Recall
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))
```

6. Result/ Output:



```
The dimensions of the dataset (18, 2)

 the total number of Training Data : (13,)

 the total number of Test Data : (5,)

 The words or Tokens in the text documents

['am', 'an', 'and', 'awesome', 'best', 'boss', 'can', 'dance', 'deal', 'do', 'enemy', 'fun', 'good', 'great', 'have', 'he', 'ho
liday', 'horrible', 'is', 'juice', 'like', 'love', 'my', 'not', 'of', 'place', 'sandwich', 'sick', 'stuff', 'sworn', 'taste',
'the', 'this', 'tired', 'to', 'tomorrow', 'view', 'we', '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
```

Training Data Set : naivetext.csv



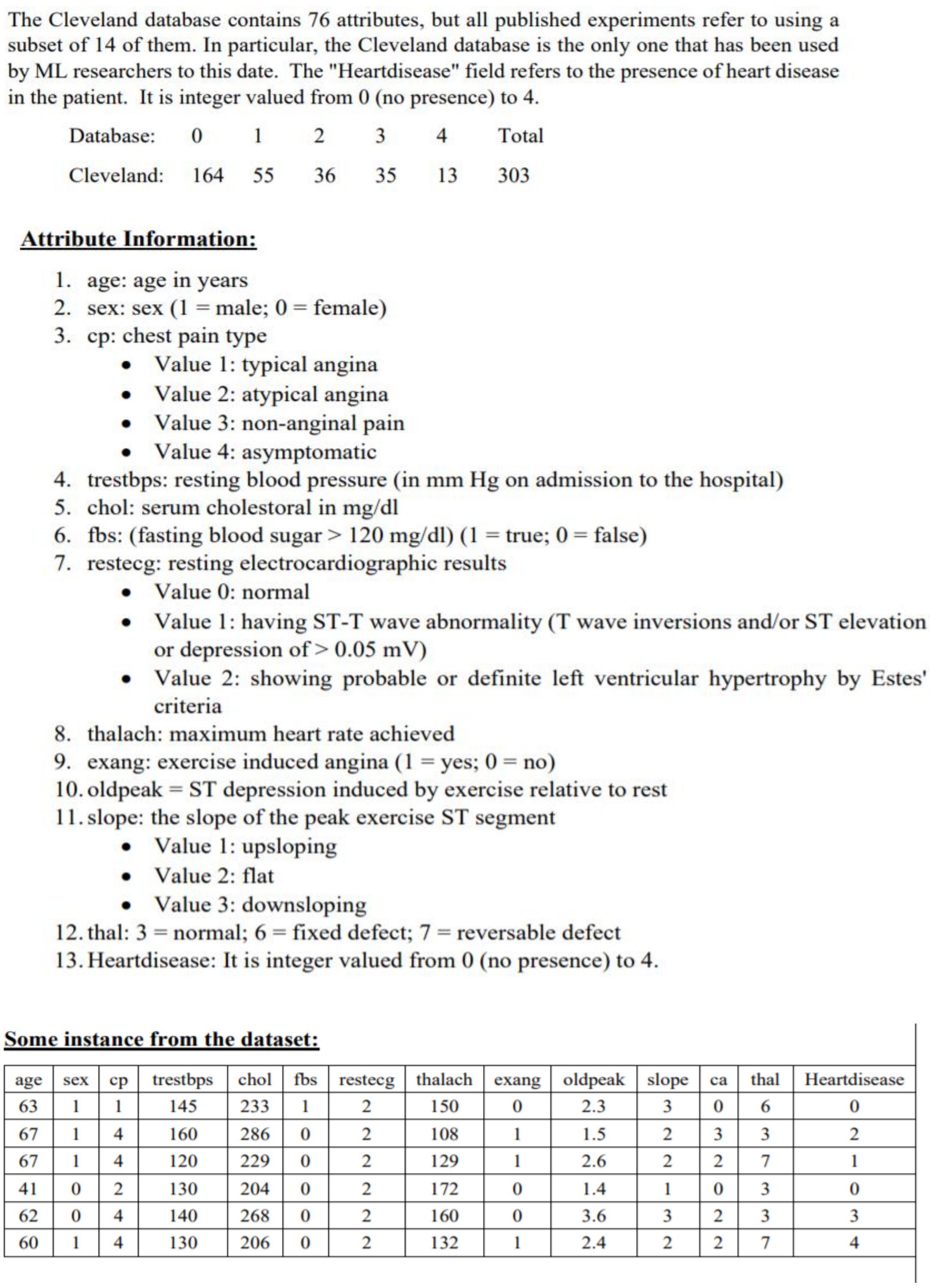
I love this sandwich	pos
This is an amazing place	pos
I feel very good about these beers	pos
This is my best work	pos
What an awesome view	pos
I do not like this restaurant	neg
I am tired of this stuff	neg
I can't deal with this	neg
He is my sworn enemy	neg
My boss is horrible	neg
This is an awesome place	pos
I do not like the taste of this juice	neg
I love to dance	pos
I am sick and tired of this place	neg
What a great holiday	pos
That is a bad locality to stay	neg
We will have good fun tomorrow	pos
I went to my enemy's house today	neg

1. **LAB PROGRAM: 7**

2. TITLE: **BAYESIAN NETWORK**
3. 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.

4. Heart Disease Data set



The Cleveland database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

Database:	0	1	2	3	4	Total
Cleveland:	164	55	36	35	13	303

Attribute Information:

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type
   - Value 1: typical angina
   - Value 2: atypical angina
   - Value 3: non-anginal pain
   - Value 4: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
   - Value 0: normal
   - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
   - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: downsloping
12. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
13. Heartdisease: It is integer valued from 0 (no presence) to 4.

Some instance from the 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

5. Implementation/ Program 7:

```
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('/Users/Chachu/Documents/Python Scripts/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 datatyes
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
#Creat Model- Bayesian Network
model
=BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisease'),('
heartdisease',
'restecg'),('heartdisease','chol')])
#Learning CPDs using Maximum Likelihood Estimators
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
# Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
#computing the Probability of HeartDisease given restecg
print('\n 1.Probability of HeartDisease given evidence=restecg :1')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
#computing the Probability of HeartDisease given cp
print('\n 2.Probability of HeartDisease given evidence= cp:2 ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

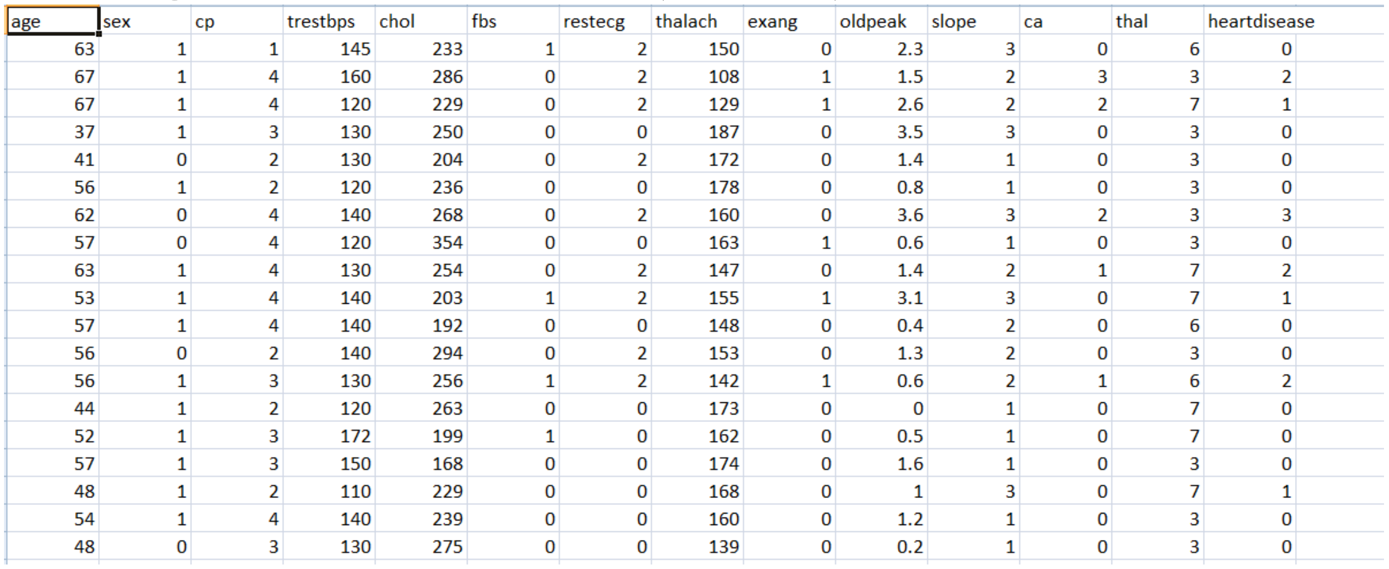
6. Result/Output:

```
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   1       145   233    1        2      150      0      2.3      3
1   67    1   4       160   286    0        2      108      1      1.5      2
2   67    1   4       120   229    0        2      129      1      2.6      2
3   37    1   3       130   250    0        0      187      0      3.5      3
4   41    0   2       130   204    0        2      172      0      1.4      1

  ca thal  heartdisease
0  0    6             0
1  3    3             2
2  2    7             1
3  0    3             0
4  0    3             0

 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
```

```
Learning CPD using Maximum likelihood estimators
 Inferencing with Bayesian Network:

 1.Probability of HeartDisease given evidence=restecg :1
+-----------------+---------------------+
| 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:2

+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

Training Data Set : heart.csv (Sample)



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
37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
57	0	4	120	354	0	0	163	1	0.6	1	0	3	0
63	1	4	130	254	0	2	147	0	1.4	2	1	7	2
53	1	4	140	203	1	2	155	1	3.1	3	0	7	1
57	1	4	140	192	0	0	148	0	0.4	2	0	6	0
56	0	2	140	294	0	2	153	0	1.3	2	0	3	0
56	1	3	130	256	1	2	142	1	0.6	2	1	6	2
44	1	2	120	263	0	0	173	0	0	1	0	7	0
52	1	3	172	199	1	0	162	0	0.5	1	0	7	0
57	1	3	150	168	0	0	174	0	1.6	1	0	3	0
48	1	2	110	229	0	0	168	0	1	3	0	7	1
54	1	4	140	239	0	0	160	0	1.2	1	0	3	0
48	0	3	130	275	0	0	139	0	0.2	1	0	3	0

1. **LAB PROGRAM: 8**

2. TITLE: **CLUSTERING BASED ON EM ALGORITHM AND K-MEANS**

3. 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.

4. THEORY:

Expectation Maximization algorithm

- The basic approach and logic of this clustering method is as follows.
- Suppose we measure a single continuous variable in a large sample of observations. Further, suppose that the sample consists of two clusters of observations with different means (and perhaps different standard deviations); within each sample, the distribution of values for the continuous variable follows the normal distribution.
- The goal of EM clustering is to estimate the means and standard deviations for each cluster so as to maximize the likelihood of the observed data (distribution).
- Put another way, the EM algorithm attempts to approximate the observed distributions of values based on mixtures of different distributions in different clusters. The results of EM clustering are different from those computed by k-means clustering.
- The latter will assign observations to clusters to maximize the distances between clusters. The EM algorithm does not compute actual assignments of observations to clusters, but classification probabilities.
- In other words, each observation belongs to each cluster with a certain probability. Of course, as a final result we can usually review an actual assignment of observations to clusters, based on the (largest) classification probability.

K means Clustering

- The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement.
- The algorithm works as follows:
  1. First we initialize k points, called means, randomly.
  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 mean so far.
  3. We repeat the process for a given number of iterations and at the end, we have our clusters.
- The "points" mentioned above are called means, because they hold the mean values of the items categorized in it. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set (if for a feature x the items have values in [0,3], we will initialize the means with values for x at [0,3]).
- **Pseudocode:**

```
1. Initialize k means with random values
2. For a given number of iterations: Iterate
       through items:
           Find the mean closest to the item Assign
           item to mean
             Update mean
```

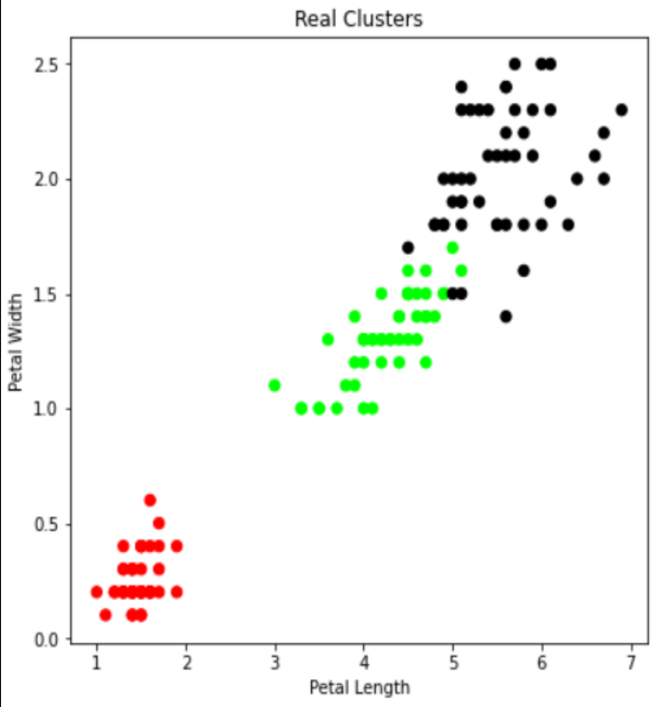
5. Implementation/Program :

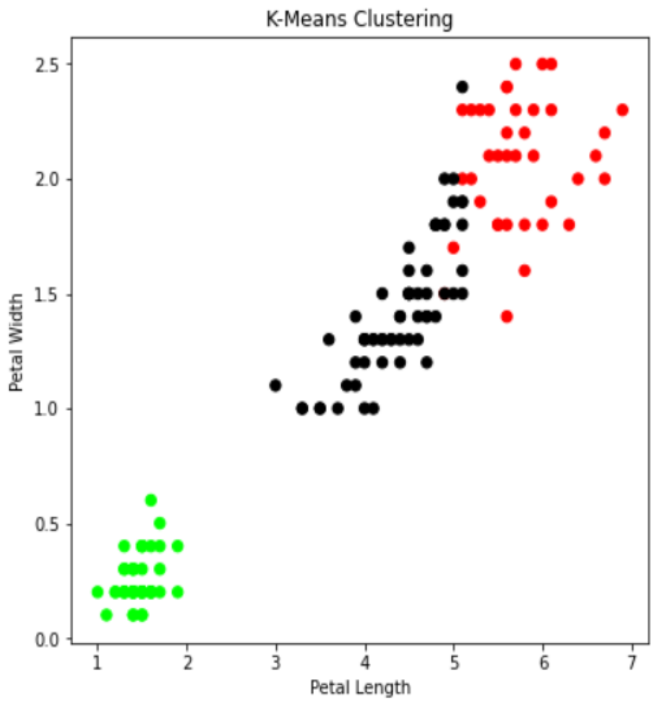
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
                              # import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns =  ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
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 belongs to

# # Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications using Petal features
plt.subplot(2, 2, 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(2, 2, 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 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=3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)
plt.subplot(2, 2, 3)
```

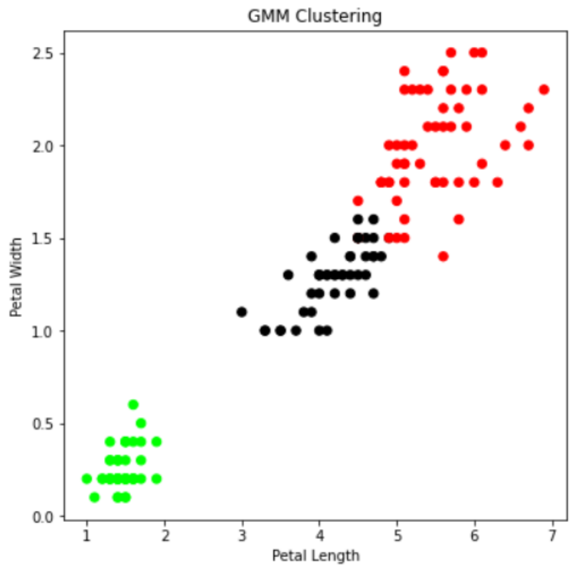
```
 plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_y], s=40)
 plt.title('GMM Clustering')
 plt.xlabel('Petal Length')
 plt.ylabel('Petal Width')
 print('Observation: The GMM using EM algorithm based clustering matched the true labels
more closely than the Kmeans.')
```

6. Result/Output:

```
Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.
```







1. **LAB PROGRAM: 9**

2. TITLE: **K-NEAREST NEIGHBOUR**
3. 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.

4. THEORY:

- K-Nearest Neighbors 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.

- **Algorithm**
  Input: Let m be the number of training data samples. Let p be an unknown point.
  Method:

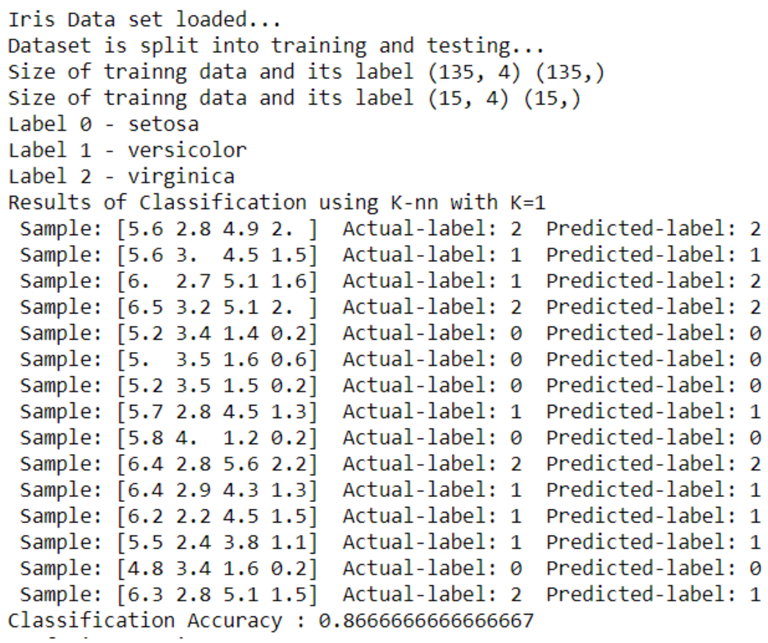
```
1. Store the training samples in an array of data points arr[]. This means each element of this
   array represents a tuple (x, y).
2. for i=0 to m
       Calculate Euclidean distance d(arr[i], p).
3. Make set S of K smallest distances obtained. Each of these distances correspond to an
   already classified data point.
   Return the majority label among S.
```

5. Implementation/ Program:

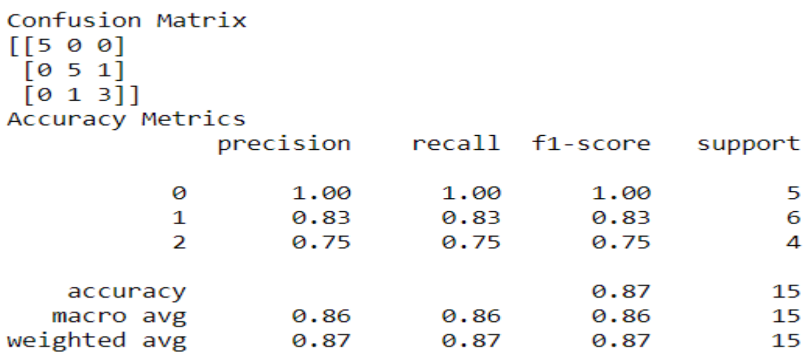
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets

# Load dataset
iris=datasets.load_iris()
print("Iris Data set loaded...")

# Split the data into train and test samples
x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.1)
print("Dataset is split into training and testing...")
print("Size of trainng data and its label",x_train.shape,y_train.shape)
print("Size of trainng data and its label",x_test.shape, y_test.shape)

# Prints Label no. and their names
for i in range(len(iris.target_names)):
    print("Label", i , "-",str(iris.target_names[i]))
    # Create object of KNN classifier
classifier = KNeighborsClassifier(n_neighbors=1)
```

```
# Perform Training
classifier.fit(x_train, y_train) # Perform testing
y_pred=classifier.predict(x_test)

# Display the results
print("Results of Classification using K-nn with K=1 ")
for r in range(0,len(x_test)):
    print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r]), " Predicted-label:", str(y_pred[r]))
print("Classification Accuracy :" , classifier.score(x_test,y_test));

from sklearn.metrics import classification_report, confusion_matrix
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

6. Result/ Output:



```
Iris Data set loaded...
Dataset is split into training and testing...
Size of trainng data and its label (135, 4) (135,)
Size of trainng data and its label (15, 4) (15,)
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=1
 Sample: [5.6 2.8 4.9 2. ]  Actual-label: 2  Predicted-label: 2
 Sample: [5.6 3.  4.5 1.5]  Actual-label: 1  Predicted-label: 1
 Sample: [6.  2.7 5.1 1.6]  Actual-label: 1  Predicted-label: 2
 Sample: [6.5 3.2 5.1 2. ]  Actual-label: 2  Predicted-label: 2
 Sample: [5.2 3.4 1.4 0.2]  Actual-label: 0  Predicted-label: 0
 Sample: [5.  3.5 1.6 0.6]  Actual-label: 0  Predicted-label: 0
 Sample: [5.2 3.5 1.5 0.2]  Actual-label: 0  Predicted-label: 0
 Sample: [5.7 2.8 4.5 1.3]  Actual-label: 1  Predicted-label: 1
 Sample: [5.8 4.  1.2 0.2]  Actual-label: 0  Predicted-label: 0
 Sample: [6.4 2.8 5.6 2.2]  Actual-label: 2  Predicted-label: 2
 Sample: [6.4 2.9 4.3 1.3]  Actual-label: 1  Predicted-label: 1
 Sample: [6.2 2.2 4.5 1.5]  Actual-label: 1  Predicted-label: 1
 Sample: [5.5 2.4 3.8 1.1]  Actual-label: 1  Predicted-label: 1
 Sample: [4.8 3.4 1.6 0.2]  Actual-label: 0  Predicted-label: 0
 Sample: [6.3 2.8 5.1 1.5]  Actual-label: 2  Predicted-label: 1
Classification Accuracy : 0.8666666666666667
```



```
Confusion Matrix
[[5 0 0]
 [0 5 1]
 [0 1 3]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         5
           1       0.83      0.83      0.83         6
           2       0.75      0.75      0.75         4

    accuracy                           0.87        15
   macro avg       0.86      0.86      0.86        15
weighted avg       0.87      0.87      0.87        15
```

1. **LAB PROGRAM: 10**

2. TITLE: **LOCALLY WEIGHTED REGRESSION ALGORITHM**
3. 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.

4. THEORY:

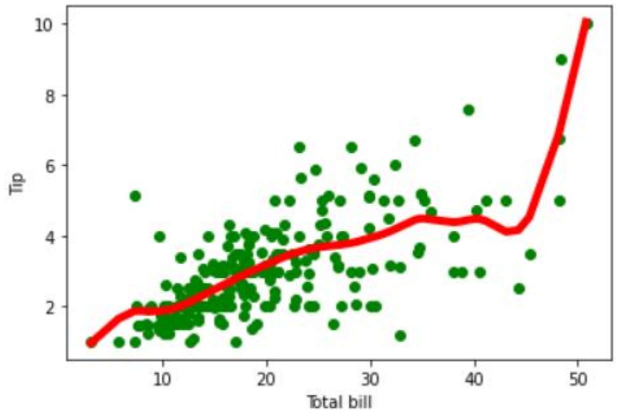
- Given a dataset X, y, we attempt to find a linear model h(x) that minimizes residual sum of squared errors. The solution is given by Normal equations.
- Linear model can only fit a straight line, however, it can be empowered by polynomial features to get more powerful models. Still, we have to decide and fix the number and types of features ahead.
- Alternate approach is given by locally weighted regression.
- Given a dataset X, y, we attempt to find a model h(x) that minimizes residual sum of weighted squared errors.
- The weights are given by a kernel function which can be chosen arbitrarily and in my case I chose a Gaussian kernel.
- The solution is very similar to Normal equations, we only need to insert diagonal weight matrix W.

5. Implementation/ Program:

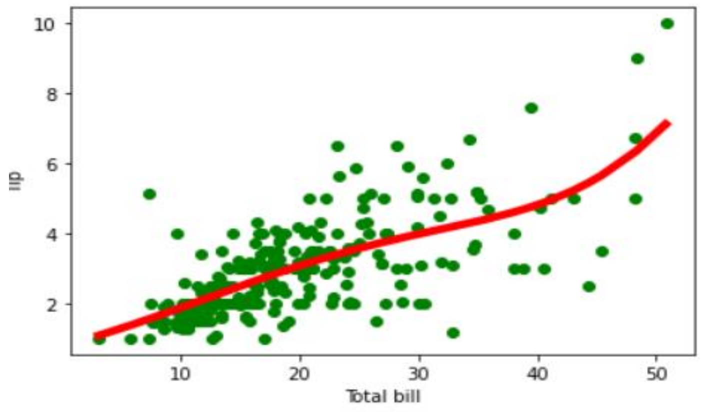
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m))) # eye - identity matrix
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.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 = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    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();

# load data points
data = pd.read_csv('/Users/Chachu/Documents/Python Scripts/data10_tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips data
tip = np.array(data.tip)

mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols

ypred = localWeightRegression(X,mtip,2) # increase k to get smooth curves
graphPlot(X,ypred)
```

6. Result/ Output:

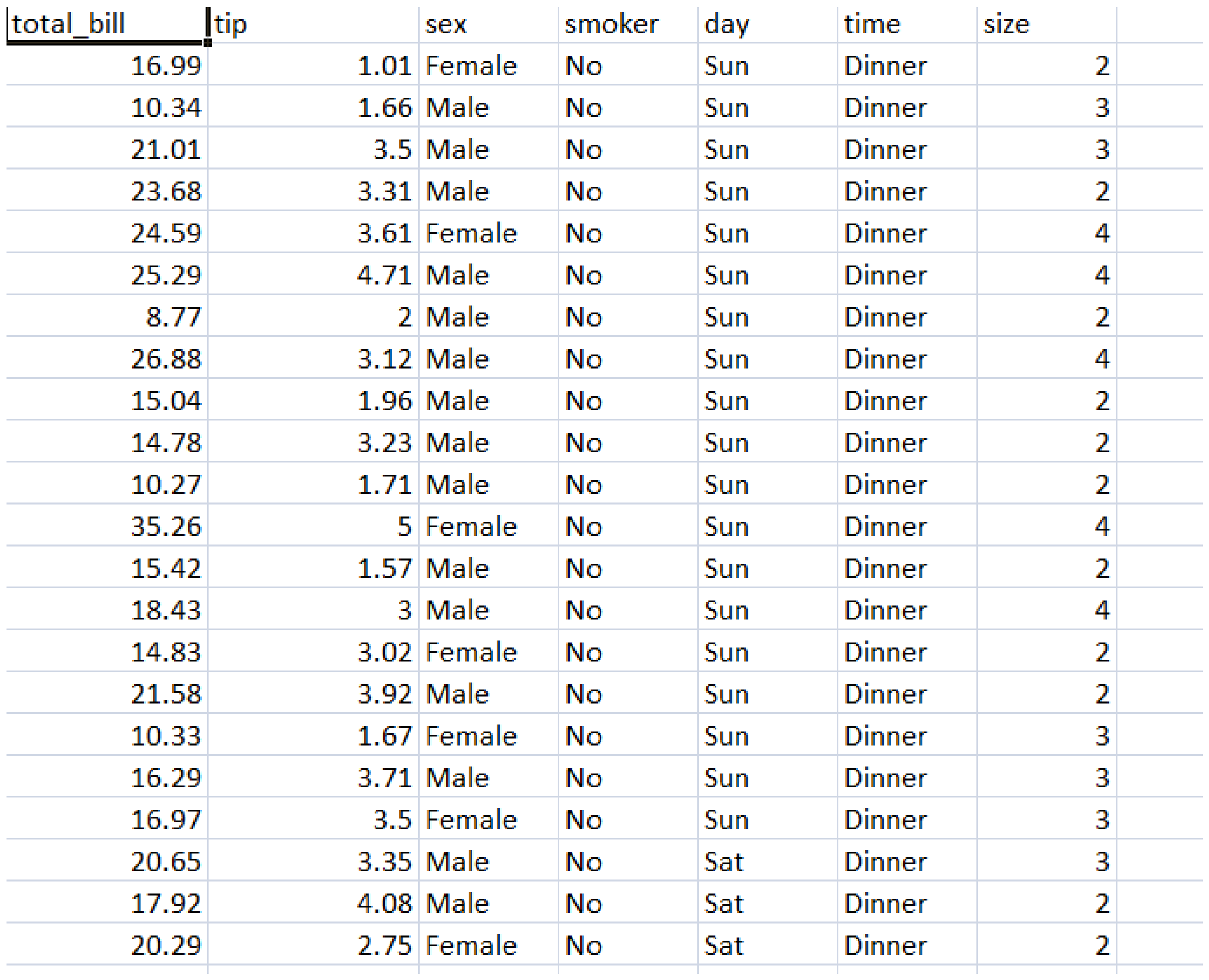
Regression with parameter k = 2 (ypred = localWeightRegression(X,mtip,**2**))



Regression with parameter k = 8 (ypred = localWeightRegression(X,mtip,**8**))



Training Data Set : data10_tips.csv (Sample)



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