1)Find\_S

Code:

import csv

def loadCsv(filename):

  lines = csv.reader(open(filename, "r"))

  dataset = list(lines)

  for i in range(len(dataset)):

    dataset[i] = dataset[i]

  return dataset

attributes = ['Sky','Temp','Humidity','Wind','Water','Forecast']

print('Attributes =',attributes)

num\_attributes = len(attributes)

filename = "ENJOYSPORT.csv"

dataset = loadCsv(filename)

print(dataset)

hypothesis=['0'] \* num\_attributes

print("Intial Hypothesis")

print(hypothesis)

print("The Hypothesis are")

for i in range(1,len(dataset)):

  target = dataset[i][-1]

  if(target == '1'):

    for j in range(num\_attributes):

        if(hypothesis[j]=='0'):

          hypothesis[j] = dataset[i][j]

        if(hypothesis[j]!= dataset[i][j]):

          hypothesis[j]='?'

  print(i+1,'=',hypothesis)

print("Final Hypothesis")

print(hypothesis)

Output:

[['Sky', 'AirTemp', 'Humidity', 'Wind', 'Water', 'Forecast', 'EnjoySport'], ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', '1'], ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', '1'], ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', '0'], ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', '1']]

Intial Hypothesis

['0', '0', '0', '0', '0', '0']

The Hypothesis are

2 = ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

3 = ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

4 = ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

5 = ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final Hypothesis

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

2)Candidate\_elimination:

import numpy as np

import pandas as pd

data = pd.read\_csv('ENJOYSPORT.csv')

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

print("\nInstances are:\n",concepts)

target = np.array(data.iloc[:,-1])

print("\nTarget Values are: ",target)

def learn(concepts, target):

    specific\_h = concepts[0].copy()

    print("\nInitialization of specific\_h and genearal\_h")

    print("\nSpecific Boundary: ", specific\_h)

    general\_h = [["?" for i in range(len(specific\_h))] for i in range(l

en(specific\_h))]

    print("\nGeneric Boundary: ",general\_h)

    for i, h in enumerate(concepts):

        print("\nInstance", i+1 , "is ", h)

        if target[i] == 1:

            print("Instance is Positive ")

            for x in range(len(specific\_h)):

                if h[x]!= specific\_h[x]:

                    specific\_h[x] ='?'

                    general\_h[x][x] ='?'

        if target[i] == 0:

            print("Instance is Negative ")

            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("Specific Bundary after ", i+1,"Instance is ",specific\_h)        print("Generic Boundary after ", i+1,"Instance is ",general\_h)

        print("\n")

    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:

Instances are:

[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']

['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]

Target Values are: [1 1 0 1]

Initialization of specific\_h and genearal\_h

Specific Boundary: ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

Generic Boundary: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

Instance is Positive

Specific Bundary after 1 Instance is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

Generic Boundary after 1 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 2 is ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

Instance is Positive

Specific Bundary after 2 Instance is ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

Generic Boundary after 2 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 3 is ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']

Instance is Negative

Specific Bundary after 3 Instance is ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

Generic Boundary after 3 Instance is [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Instance 4 is ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']

Instance is Positive

Specific Bundary after 4 Instance is ['Sunny' 'Warm' '?' 'Strong' '?' '?']

Generic Boundary after 4 Instance is [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific\_h:

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

Final General\_h:

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

**3)Decision Tree**

import pandas as pd

import numpy as np

dataset= pd.read\_csv('play\_tennis.csv',names=['outlook','temperature','humidity','wind','class'])

print(dataset)

def entropy(target\_col):

    elements,counts = np.unique(target\_col,return\_counts = True)

    entropy = np.sum([(-counts[i]/np.sum(counts))\*np.log2(counts[i]/np.sum(counts))for i in range(len(elements))])

    return entropy

def InfoGain(data,split\_attribute\_name,target\_name="class"):

    total\_entropy = entropy(data[target\_name])

    vals,counts= np.unique(data[split\_attribute\_name],return\_counts=True)

    Weighted\_Entropy = np.sum([(counts[i]/np.sum(counts))\*entropy(data.where(data[split\_attribute\_name]==vals[i]).dropna()[target\_name]) for i in range(len(vals))])

    Information\_Gain = total\_entropy - Weighted\_Entropy

    return Information\_Gain

def ID3(data,originaldata,features,target\_attribute\_name="class",parent\_node\_class = None):

    if len(np.unique(data[target\_attribute\_name])) <= 1:

        return np.unique(data[target\_attribute\_name])[0]

    elif len(data)==0:

        return np.unique(originaldata[target\_attribute\_name])[np.argmax(np.unique(originaldata[target\_attribute\_name],return\_counts=True)[1])]    elif len(features) ==0:

        return parent\_node\_class

    else:

        parent\_node\_class = np.unique(data[target\_attribute\_name])[np.argmax(np.unique(data[target\_attribute\_name],return\_counts=True)[1])]

        item\_values = [InfoGain(data,feature,target\_attribute\_name) for feature in features]

        best\_feature\_index = np.argmax(item\_values)

        best\_feature = features[best\_feature\_index]

        tree = {best\_feature:{}}

        features = [i for i in features if i != best\_feature]

        for value in np.unique(data[best\_feature]):

            value = value

            sub\_data = data.where(data[best\_feature] == value).dropna()

            subtree = ID3(sub\_data,dataset,features,target\_attribute\_name,parent\_node\_class)

            tree[best\_feature][value] = subtree

        return(tree)

tree = ID3(dataset,dataset,dataset.columns[:-1])

print(' \nDisplay Tree\n',tree)

**Output:**

outlook temperature humidity wind class

day outlook temp humidity wind play

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

Display Tree

{'outlook': {'Overcast': 'Yes', 'Rain': {'wind': {'Strong': 'No', 'Weak': 'Yes'}}, 'Sunny': {'humidity': {'High': 'No', 'Normal': 'Yes'}}, 'outlook': 'play'}}

**4)Backpropagation algorithm**

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=5 #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 Propogation

    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)

    hiddengrad = derivatives\_sigmoid(hlayer\_act)#how much hidden layer wts contributed to error

    d\_hiddenlayer = EH \* hiddengrad

    wout += hlayer\_act.T.dot(d\_output) \*lr   # dotproduct of nextlayererror and currentlayerop

    wh += X.T.dot(d\_hiddenlayer) \*lr

    print ("-----------Epoch-", i+1, "Starts----------")

    print("Input: \n" + str(X))

    print("Actual Output: \n" + str(y))

    print("Predicted Output: \n" ,output)

    print ("-----------Epoch-", i+1, "Ends----------\n")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

Input Dataset:

|  |  |  |
| --- | --- | --- |
| X | Y | class |
| 2 | 9 | 92 |
| 1 | 5 | 86 |
| 3 | 6 | 89 |

**Output:**

----------Epoch- 1 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.81946901]

[0.80312503]

[0.82285168]]

-----------Epoch- 1 Ends----------

-----------Epoch- 2 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.82027619]

[0.80391667]

[0.82366284]]

-----------Epoch- 2 Ends----------

-----------Epoch- 3 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.82106961]

[0.80469506]

[0.82446007]]

-----------Epoch- 3 Ends----------

-----------Epoch- 4 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.82184962]

[0.80546054]

[0.82524371]]

-----------Epoch- 4 Ends----------

-----------Epoch- 5 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.82261656]

[0.80621342]

[0.8260141 ]]

-----------Epoch- 5 Ends----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.82261656]

[0.80621342]

[0.8260141 ]]

5.Bayes classification

import csv

import random

import math

import numpy as np

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[1:len(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("play\_tennis.csv")

print(traindata)

print("the attribute names of training data are:",metadata)

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)

Output:

[['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']]

the attribute names of training data are: ['day', 'outlook', 'temp', 'humidity', 'wind', 'play']

the training data set are:

['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']

the test data set are:

['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']

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 %

6.Bayes classification for text classification

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('text\_classification.csv',names=['message','label'])

print('the dimension of the dataset',msg.shape)

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

X=msg.message

y=msg.labelnum

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)

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())

clf = MultinomialNB().fit(xtrain\_dtm,ytrain)

predicted = clf.predict(xtest\_dtm)

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 dimension 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

['about', 'am', 'amazing', 'an', 'and', 'awesome', 'beers', 'boss', 'can', 'dance', 'deal', 'donot', 'enemy', 'feel', 'fun', 'good', 'great', 'have', 'he', 'holiday', 'horrible', 'house', 'is', 'juice', 'like', 'love', 'my', 'of', 'place', 'sandwich', 'sick', 'sworn', 'taste', 'the', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'we', 'went', 'what', 'will', 'with']

Accuracy of the classifier is 0.6

confusion matrix

[[2 1]

[1 1]]

the value of precision 0.5

the value of recall 0.5

7.Bayesian Belief network

pip install pgmpy

import pandas as pd

import csv

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

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 datatyes

print('\n Attributes and datatypes')

print(heartDisease.dtypes)

model = BayesianModel([('age','target'),('sex','target'),('exang','target'),('cp','target'),('target','restecg'),('target','chol')])

# learning CPDs using Maximum likelihood estimators

print("\n learning CPD using Maximum likelihood estimators")

model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print(model.get\_cpds('age'))

print(model.get\_cpds('exang'))

print(model.get\_cpds('sex'))

print(model.get\_cpds('cp'))

print(model.get\_cpds('restecg'))

print("\n inferencing with Bayesian Netwok:")

HeartDisease\_infer = VariableElimination(model)

q1=HeartDisease\_infer.query(variables=['target'],evidence={'restecg':1})

print(q1)

q2=HeartDisease\_infer.query(variables=['target'],evidence={'age':40})

print(q2)

q3=HeartDisease\_infer.query(variables=['target'],evidence={'cp':3})

print(q3)

**output:**

learning CPD using Maximum likelihood estimators

+---------+------------+

| age(29) | 0.00330033 |

+---------+------------+

| age(34) | 0.00660066 |

+---------+------------+

| age(35) | 0.0132013 |

+---------+------------+

| age(37) | 0.00660066 |

+---------+------------+

| age(38) | 0.00990099 |

+---------+------------+

| age(39) | 0.0132013 |

+---------+------------+

| age(40) | 0.00990099 |

+---------+------------+

| age(41) | 0.0330033 |

+---------+------------+

| age(42) | 0.0264026 |

+---------+------------+

| age(43) | 0.0264026 |

+---------+------------+

| age(44) | 0.0363036 |

+---------+------------+

| age(45) | 0.0264026 |

+---------+------------+

| age(46) | 0.0231023 |

+---------+------------+

| age(47) | 0.0165017 |

+---------+------------+

| age(48) | 0.0231023 |

+---------+------------+

| age(49) | 0.0165017 |

+---------+------------+

| age(50) | 0.0231023 |

+---------+------------+

| age(51) | 0.039604 |

+---------+------------+

| age(52) | 0.0429043 |

+---------+------------+

| age(53) | 0.0264026 |

+---------+------------+

| age(54) | 0.0528053 |

+---------+------------+

| age(55) | 0.0264026 |

+---------+------------+

| age(56) | 0.0363036 |

+---------+------------+

| age(57) | 0.0561056 |

+---------+------------+

| age(58) | 0.0627063 |

+---------+------------+

| age(59) | 0.0462046 |

+---------+------------+

| age(60) | 0.0363036 |

+---------+------------+

| age(61) | 0.0264026 |

+---------+------------+

| age(62) | 0.0363036 |

+---------+------------+

| age(63) | 0.029703 |

+---------+------------+

| age(64) | 0.0330033 |

+---------+------------+

| age(65) | 0.0264026 |

+---------+------------+

| age(66) | 0.0231023 |

+---------+------------+

| age(67) | 0.029703 |

+---------+------------+

| age(68) | 0.0132013 |

+---------+------------+

| age(69) | 0.00990099 |

+---------+------------+

| age(70) | 0.0132013 |

+---------+------------+

| age(71) | 0.00990099 |

+---------+------------+

| age(74) | 0.00330033 |

+---------+------------+

| age(76) | 0.00330033 |

+---------+------------+

| age(77) | 0.00330033 |

+---------+------------+

+----------+----------+

| exang(0) | 0.673267 |

+----------+----------+

| exang(1) | 0.326733 |

+----------+----------+

+--------+----------+

| sex(0) | 0.316832 |

+--------+----------+

| sex(1) | 0.683168 |

+--------+----------+

+-------+-----------+

| cp(0) | 0.471947 |

+-------+-----------+

| cp(1) | 0.165017 |

+-------+-----------+

| cp(2) | 0.287129 |

+-------+-----------+

| cp(3) | 0.0759076 |

+-------+-----------+

+------------+----------------------+----------------------+

| target | target(0) | target(1) |

+------------+----------------------+----------------------+

| restecg(0) | 0.572463768115942 | 0.4121212121212121 |

+------------+----------------------+----------------------+

| restecg(1) | 0.4057971014492754 | 0.5818181818181818 |

+------------+----------------------+----------------------+

| restecg(2) | 0.021739130434782608 | 0.006060606060606061 |

+------------+----------------------+----------------------+

inferencing with Bayesian Netwok:

Finding Elimination Order: : 100%

4/4 [00:00<00:00, 18.92it/s]

Eliminating: cp: 100%

4/4 [00:00<00:00, 6.14it/s]

+-----------+---------------+

| target | phi(target) |

+===========+===============+

| target(0) | 0.4242 |

+-----------+---------------+

| target(1) | 0.5758 |

+-----------+---------------+

Finding Elimination Order: : 100%

3/3 [00:00<00:00, 15.30it/s]

Eliminating: cp: 100%

3/3 [00:00<00:00, 36.25it/s]

+-----------+---------------+

| target | phi(target) |

+===========+===============+

| target(0) | 0.6527 |

+-----------+---------------+

| target(1) | 0.3473 |

+-----------+---------------+

Finding Elimination Order: : 0%

0/3 [00:00<?, ?it/s]

Eliminating: sex: 100%

3/3 [00:00<00:00, 38.03it/s]

+-----------+---------------+

| target | phi(target) |

+===========+===============+

| target(0) | 0.4588 |

+-----------+---------------+

| target(1) | 0.5412 |

+-----------+---------------+

**8)EM and K-means**

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import pandas as pd

import numpy as np

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 sample belongs to

## visualise the clustering results

plt.figure(figsize=(14,14))

colormap = np.array(['red','lime','black'])

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 classification

plt.subplot(2,2,2)

plt.scatter(X.Petal\_Length,X.Petal\_width,c=colormap[model.labels\_],s=40)

plt.title('K-Mean Clustering')

plt.xlabel('PetalLenght')

plt.ylabel('Petal width')

#general EM for GMM

from sklearn import preprocessing

#transform your sata such that its distribution will have a #mean value 0 and standard deviation of 1

from sklearn.preprocessing import StandardScaler

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')

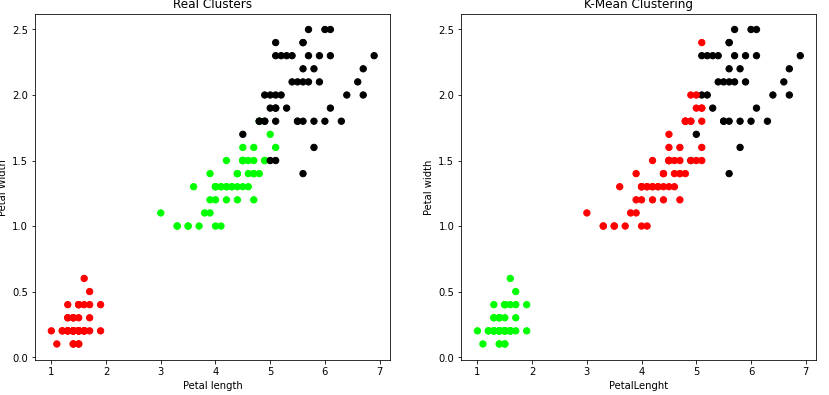
plt.xlabel('petal lenght')

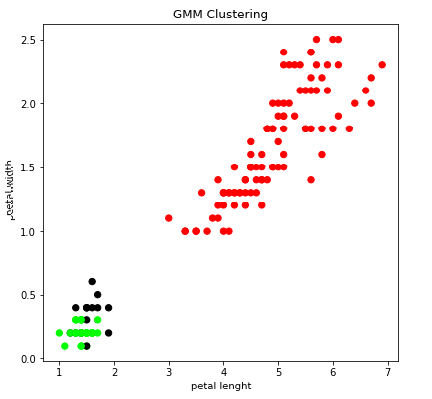
plt.ylabel('petal width')

print('observation :the GMM using EM algorithm based clustering matched the true labels are closely than the kmeans')

**Output:**

observation :the GMM using EM algorithm based clustering matched the true labels are closely than the kmeans





**9)KNN algorithm**

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=0.2)

print("Data set is split into traning and testing..")

print("size of traning data and its label",X\_train.shape,y\_train.shape)

print("size of testing data and its label",x\_test.shape,y\_test.shape)

#print label no. and their names

for i in range(len(iris.target\_names)):

    print("label" , i , "-",str(iris.target\_names[i]))

classifier = KNeighborsClassifier(n\_neighbors=1)

classifier.fit(X\_train,y\_train)

y\_pred=classifier.predict(x\_test)

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))

**Output:**

iris data set loaded...

Data set is split into traning and testing..

size of traning data and its label (120, 4) (120,)

size of testing data and its label (30, 4) (30,)

label 0 - setosa

label 1 - versicolor

label 2 - virginica

results of classification using K-NN with k=1

sample: [5.8 2.7 4.1 1. ] Actual-label: 1 predicted-label: 1

sample: [5.1 3.8 1.9 0.4] Actual-label: 0 predicted-label: 0

sample: [6.4 3.2 4.5 1.5] Actual-label: 1 predicted-label: 1

sample: [6.1 2.8 4.7 1.2] Actual-label: 1 predicted-label: 1

sample: [5.2 3.4 1.4 0.2] Actual-label: 0 predicted-label: 0

sample: [5.8 2.7 5.1 1.9] Actual-label: 2 predicted-label: 2

sample: [6.3 2.8 5.1 1.5] Actual-label: 2 predicted-label: 1

sample: [7.1 3. 5.9 2.1] Actual-label: 2 predicted-label: 2

sample: [6.7 2.5 5.8 1.8] Actual-label: 2 predicted-label: 2

sample: [6.8 2.8 4.8 1.4] Actual-label: 1 predicted-label: 1

sample: [5.1 3.7 1.5 0.4] Actual-label: 0 predicted-label: 0

sample: [5. 3.6 1.4 0.2] Actual-label: 0 predicted-label: 0

sample: [6.5 3. 5.8 2.2] Actual-label: 2 predicted-label: 2

sample: [6. 2.7 5.1 1.6] Actual-label: 1 predicted-label: 2

sample: [5.1 3.3 1.7 0.5] Actual-label: 0 predicted-label: 0

sample: [6.8 3.2 5.9 2.3] Actual-label: 2 predicted-label: 2

sample: [5.1 3.8 1.5 0.3] Actual-label: 0 predicted-label: 0

sample: [5.3 3.7 1.5 0.2] Actual-label: 0 predicted-label: 0

sample: [5.4 3.9 1.7 0.4] Actual-label: 0 predicted-label: 0

sample: [6.9 3.1 4.9 1.5] Actual-label: 1 predicted-label: 1

sample: [4.9 3.1 1.5 0.1] Actual-label: 0 predicted-label: 0

sample: [4.4 2.9 1.4 0.2] Actual-label: 0 predicted-label: 0

sample: [7.6 3. 6.6 2.1] Actual-label: 2 predicted-label: 2

sample: [6.3 3.4 5.6 2.4] Actual-label: 2 predicted-label: 2

sample: [5. 2. 3.5 1. ] Actual-label: 1 predicted-label: 1

sample: [6.5 3. 5.5 1.8] Actual-label: 2 predicted-label: 2

sample: [5.6 2.9 3.6 1.3] Actual-label: 1 predicted-label: 1

sample: [6.8 3. 5.5 2.1] Actual-label: 2 predicted-label: 2

sample: [6. 2.9 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

classification accuracy: 0.9333333333333333

confusion matrix

[[10 0 0]

[ 0 9 1]

[ 0 1 9]]

Accuracy Metrics

precision recall f1-score support

0 1.00 1.00 1.00 10

1 0.90 0.90 0.90 10

2 0.90 0.90 0.90 10

accuracy 0.93 30

macro avg 0.93 0.93 0.93 30

weighted avg 0.93 0.93 0.93 30

**10)LWR algorithm**

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

# kernel smoothing function

def kernel(point, xmat, k):

    m,n = np.shape(xmat)

    weights = np.mat(np.eye((m)))

    for j in range(m):

        diff = point - X[j]

        weights[j, j] = np.exp(diff \* diff.T / (-2.0 \* k\*\*2))

    return weights

# function to return local weight of eah traiining example

def localWeight(point, xmat, ymat, k):

    wt = kernel(point, xmat, k)

    W = (X.T \* (wt\*X)).I \* (X.T \* wt \* ymat.T)

    return W

# root function that drives the algorithm

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

#import data

data = pd.read\_csv('10-dataset.csv')

# place them in suitable data types

colA = np.array(data.total\_bill)

colB = np.array(data.tip)

mcolA = np.mat(colA)

mcolB = np.mat(colB)

m = np.shape(mcolB)[1]

one = np.ones((1, m), dtype = int)

# horizontal stacking

X = np.hstack((one.T, mcolA.T))

print(X.shape)

# predicting values using LWLR

ypred = localWeightRegression(X, mcolB, 0.8)

# plotting the predicted graph

xsort = X.copy()

xsort.sort(axis=0)

plt.scatter(colA, colB, color='blue')

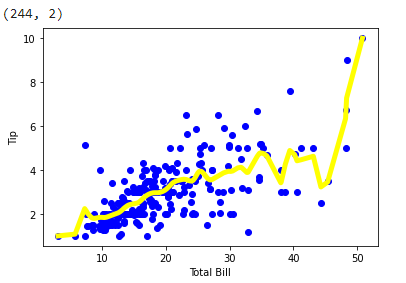
plt.plot(xsort[:, 1], ypred[X[:, 1].argsort(0)], color='yellow', linewidth=5)

plt.xlabel('Total Bill')

plt.ylabel('Tip')

plt.show()

**Output:**

****