**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaNangama”, Belgaum -590014, Karnataka.**



# LAB REPORT on

Machine Learning

***Submitted by***

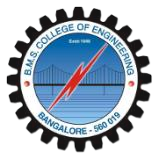
# ISHIKA SINGHAL(1BM19CS064)

***in partial fulfilment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

# COMPUTER SCIENCE AND ENGINEERING



**B.M.S. COLLEGE OF ENGINEERING** **BENGALURU-560019** May**-2022 to July-2022**

## (Autonomous Institution under VTU)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “Machine Learning” carried out by **Ishika Singhal(1BM19CS064), who is bonafide student of B. M. S. College of Engineering. It is in partial** fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering**

of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Machine Learning - (20CS6PCMAL)** work prescribed for the said degree.

**Dr G R Asha Dr. Jyothi S Nayak**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE,Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl. No**  **.** | **Experiment Title** | **Page No.** |
| **1** | **Find-S** | **4** |
| **2** | **Candidate Elimination** | **7** |
| **3** | **Decision Tree** | **9** |
| **4** | **Naïve Bayes** | **16** |
| **5** | **Linear Regression** | **20** |
| **6** | **Bayesian network** | **24** |
| **7** | **k-Means algorithm** | **28** |
| **8** | **EM algorithm** | **34** |
| **9** | **k-Nearest Neighbour algorithm** | **36** |
| **10** | **Non-Parametric Locally Weighted Regression algorithm** | **38** |

**Course Outcome**

|  |  |
| --- | --- |
| CO1 | Ability to apply the different learning algorithms. |
| CO2 | Ability to analyse the learning techniques for given dataset |
| CO3 | Ability to design a model using machine learning to solve a problem. |
| CO4 | Ability to conduct practical experiments to solve problems using appropriate machine learning Techniques. |

1. **Implement and demonstrate the FIND-S algorithm for ﬁnding the most speciﬁc hypothesis based on a given set of training data samples.**

# a) Using CSV as input:

**import** csv

**def** updateHypothesis(x,h):

**if** h**==**[]:

## return x

**for** i **in** range(0,len(h)):

**if** x[i]**.**upper()**!=**h[i]**.**upper(): h[i] **=** '?'

## return h

**if** name **==** " main ": data **=** []

h **=** []

*# reading csv file*

**with** open('Desktop/FindS.csv', 'r') **as** file: reader **=** csv**.**reader(file)

print("Data: ")

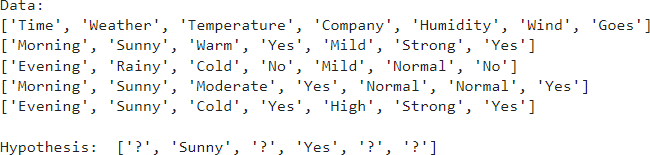
**for** row **in** reader: data**.**append(row) print(row)

**if** data:

**for** x **in** data:

**if** x[**-**1]**.**upper()**==**"YES": x**.**pop() *# removing last field* h **=** updateHypothesis(x,h)

print("\nHypothesis: ",h)

**Output:**

# B) Using user Input:

**import** numpy **as** np

**import** pandas **as** pd

n**=**int(input("Enter the number of attributes ")) l**=**int(input("Enter the number of rows "))

print("Enter the ",n,"ättributes") attributes**=**[]

**for** i **in** range(1,n**+**1):

print("Enter the name of ",i," attribute ") name**=**input()

**for** i **in** range(1,l**+**1):

print("Ënter the values of ",i," row") print("Enter the values of attributes") res**=**[]

**for** j **in** range(1,l**+**1): res**.**append(input()) attributes**.**append(res)

print("Enter the target values") target**=**[]

**for** i **in** range(1,l**+**1):

print("Enter the value of ",i," target") x**=**input()

target**.**append(x)

**def** findS(c,t):

**for** i, val **in** enumerate(t):

**if** val **==** "Yes": specific\_hypothesis **=** c[i]**.**copy() **break**

**for** i, val **in** enumerate(c):

**if** t[i] **==** "Yes":

**for** x **in** range(len(specific\_hypothesis)):

**if** val[x] **!=** specific\_hypothesis[x]: specific\_hypothesis[x] **=** '?'

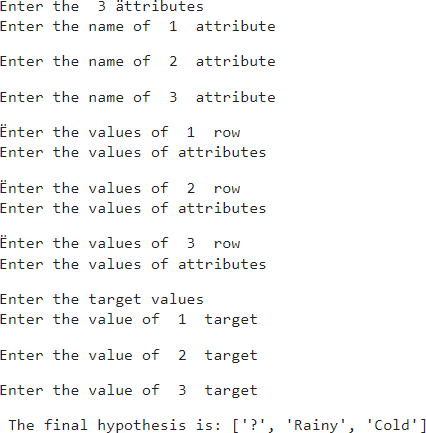
## else:

**pass**

**return** specific\_hypothesis

print("\n The final hypothesis is:",findS(attributes,target))

**Output:**



# For a given set of training data examples stored in a .CSV ﬁle, implement and demonstrate the Candidate-Elimination algorithm to output a descripon of the set

**of all hypotheses consistent with the training examples**

**import** numpy **as** np

**import** pandas **as** pd

*#to read the data in the csv file*

data **=** pd**.**DataFrame(data**=**pd**.**read\_csv('/content/drive/MyDrive/enjoysport.csv')) print(data,"\n")

*#making an array of all the attributes* concepts **=** np**.**array(data**.**iloc[:,0:**-**1]) print("The attributes are: ",concepts)

*#segregating the target that has positive and negative examples*

target **=** np**.**array(data**.**iloc[:,**-**1]) print("\n The target is: ",target)

*#training function to implement candidate\_elimination algorithm*

**def** learn(concepts, target):

specific\_h **=** concepts[0]**.**copy()

print("\n 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)*

**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("\n 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(['?', '?', '?', '?', '?', '?'])

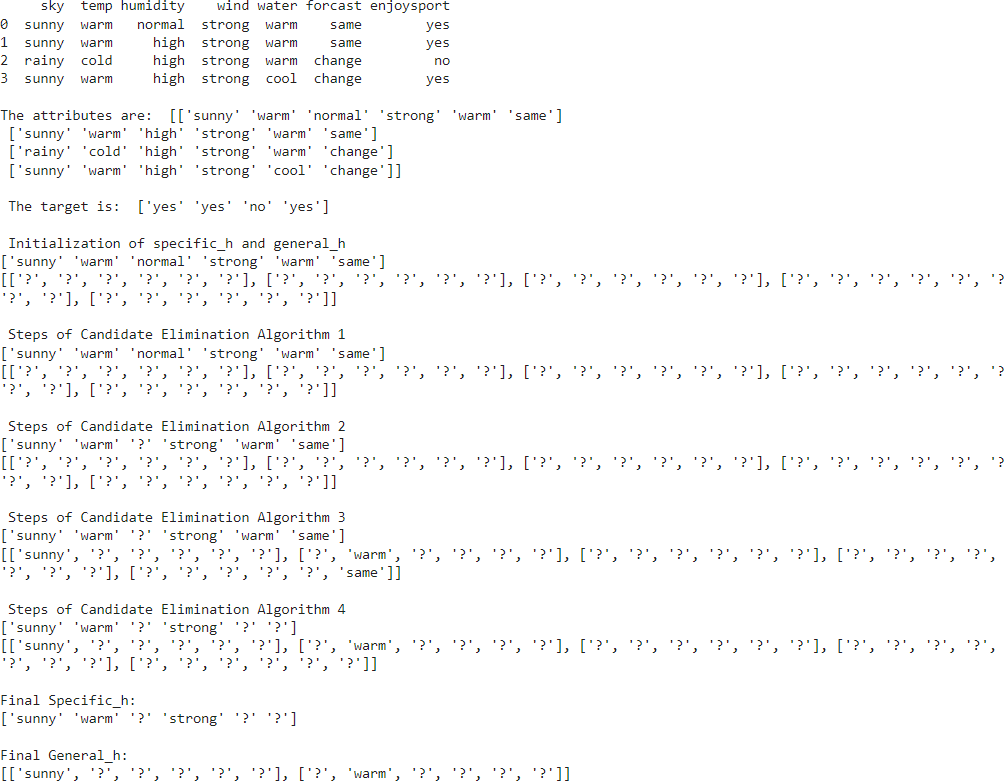
**return** specific\_h, general\_h

s\_final, g\_final **=** learn(concepts, target)

*#obtaining the final hypothesis*

print("\nFinal Specific\_h:", s\_final, sep**=**"\n") print("\nFinal General\_h:", g\_final, sep**=**"\n")

**Output:**



# 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.**

# ID3 :

**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("id3.csv") node1**=**build\_tree(dataset,features)

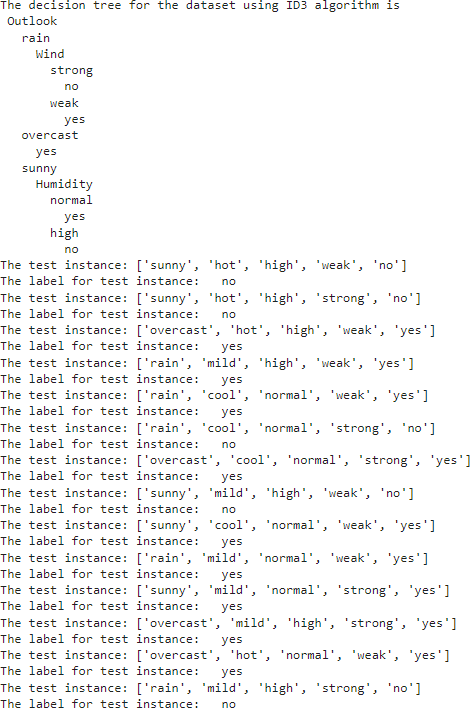
print("The decision tree for the dataset using ID3 algorithm is") print\_tree(node1,0)

testdata,features**=**load\_csv("id3.csv")

**for** xtest **in** testdata:

print("The test instance:",xtest)

print("The label for test instance:",end**=**" ") classify(node1,xtest,features)

**Output:**

* 1. **Using SKlearn:**

**import** pandas **as** pd

**import** numpy **as** np

**from** sklearn.datasets **import** load\_iris

data **=** load\_iris()

df **=** pd**.**DataFrame(data**.**data, columns **=** data**.**feature\_names) df**.**head()

df['Species'] **=** data**.**target

*#replace this with the actual names*

target **=** np**.**unique(data**.**target)

target\_names **=** np**.**unique(data**.**target\_names) targets **=** dict(zip(target, target\_names)) df['Species'] **=** df['Species']**.**replace(targets)

x **=** df**.**drop(columns**=**"Species") y **=** df["Species"]

feature\_names **=** x**.**columns labels **=** y**.**unique()

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, test\_x, y\_train, test\_lab **=** train\_test\_split(x,y,test\_size **=** 0.4,random\_state **=** 42)

**from** sklearn.tree **import** DecisionTreeClassifier

clf **=** DecisionTreeClassifier(max\_depth **=**4, random\_state **=** 42)

clf**.**fit(X\_train, y\_train) test\_pred **=** clf**.**predict(test\_x)

**from** sklearn **import** metrics **import** seaborn **as** sns **import** matplotlib.pyplot **as** plt

In [2]:

In [3]:

In [5]:

In [6]:

In [7]:

In [8]:

In [9]:

In [11]:

confusion\_matrix **=** metrics**.**confusion\_matrix(test\_lab,test\_pred)

confusion\_matrix

matrix\_df **=** pd**.**DataFrame(confusion\_matrix) ax **=** plt**.**axes()

sns**.**set(font\_scale**=**1.3) plt**.**figure(figsize**=**(10,7))

sns**.**heatmap(matrix\_df, annot**=True**, fmt**=**"g", ax**=**ax, cmap**=**"magma") ax**.**set\_title('Confusion Matrix - Decision Tree') ax**.**set\_xlabel("Predicted label", fontsize **=**15) ax**.**set\_xticklabels(['']**+**labels)

ax**.**set\_ylabel("True Label", fontsize**=**15) ax**.**set\_yticklabels(list(labels), rotation **=** 0) plt**.**show()

clf**.**score(test\_x,test\_lab)

**from** sklearn **import** tree

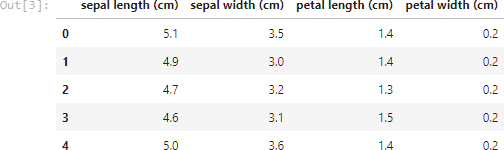
fig **=** plt**.**figure(figsize**=**(25,20))

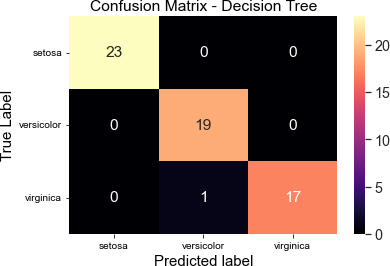
\_ **=** tree**.**plot\_tree(clf,

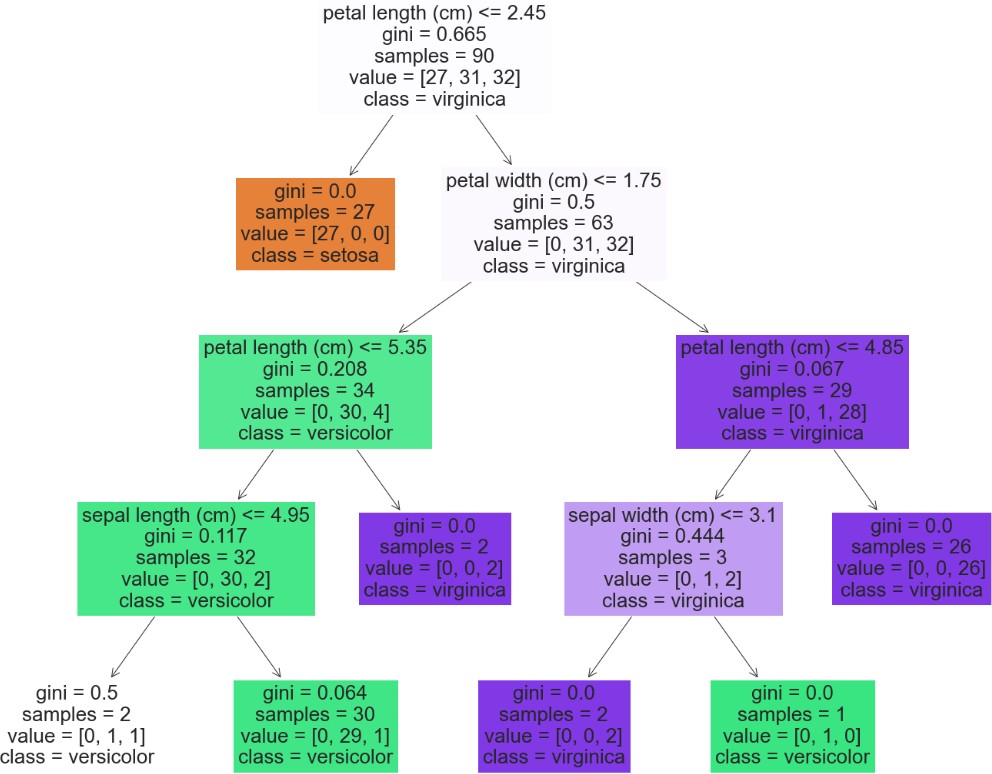
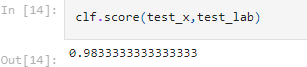
feature\_names**=**data**.**feature\_names, class\_names**=**data**.**target\_names, filled**=True**)

In [12]:

**Output:**







# 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**

# Without using SKlearn:

**import** numpy **as** np

**import** pandas **as** pd

data **=** pd**.**read\_csv('/content/dataset.csv') data**.**head()

y **=** list(data['PlayTennis']**.**values) X **=** data**.**iloc[:,1:]**.**values print(f'Target Values: {y}') print(f'Features: \n{X}')

y\_train **=** y[:8] y\_val **=** y[8:] X\_train **=** X[:8] X\_val **=** X[8:]

print(f"Number of instances in training set: {len(X\_train)}") print(f"Number of instances in testing set: {len(X\_val)}") **class** NaiveBayesClassifier:

**def** init (self, X, y): self**.**X, self**.**y **=** X, y self**.**N **=** len(self**.**X) self**.**dim **=** len(self**.**X[0])

self**.**attrs **=** [[] **for** \_ **in** range(self**.**dim)] self**.**output\_dom **=** {}

self**.**data **=** []

**for** i **in** range(len(self**.**X)):

**for** j **in** range(self**.**dim):

**if not** self**.**X[i][j] **in** self**.**attrs[j]: self**.**attrs[j]**.**append(self**.**X[i][j])

**if not** self**.**y[i] **in** self**.**output\_dom**.**keys(): self**.**output\_dom[self**.**y[i]] **=** 1

## else:

self**.**output\_dom[self**.**y[i]] **+=** 1 self**.**data**.**append([self**.**X[i], self**.**y[i]])

**def** classify(self, entry): solve **= None** max\_arg **= -**1

**for** y **in** self**.**output\_dom**.**keys(): prob **=** self**.**output\_dom[y]**/**self**.**N **for** i **in** range(self**.**dim):

cases **=** [x **for** x **in** self**.**data **if** x[0][i] **==** entry[i] **and** x[1] **==** y] n **=** len(cases)

prob **\*=** n**/**self**.**N **if** prob **>** max\_arg: max\_arg **=** prob

solve **=** y

**return** solve

nbc **=** NaiveBayesClassifier(X\_train, y\_train) total\_cases **=** len(y\_val)

good **=** 0

bad **=** 0 predictions **=** []

**for** i **in** range(total\_cases): predict **=** nbc**.**classify(X\_val[i]) predictions**.**append(predict)

**if** y\_val[i] **==** predict: good **+=** 1

## else:

bad **+=** 1

print('Predicted values:', predictions) print('Actual values:', y\_val)

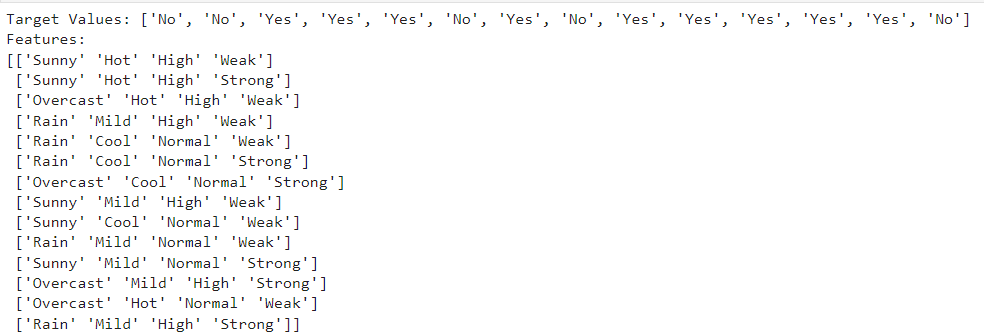
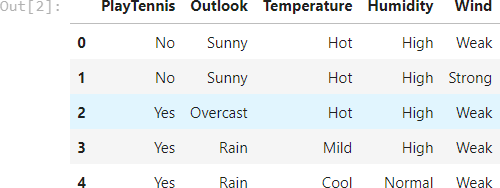
print()

print('Total number of testing instances in the dataset:', total\_cases) print('Number of correct predictions:', good)

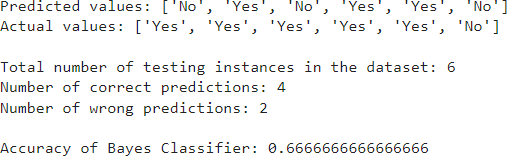
print('Number of wrong predictions:', bad) print()

print('Accuracy of Bayes Classifier:', good**/**total\_cases)

**Output:**







* 1. **Using SKlearn:**

**import** numpy **as** np *# linear algebra*

**import** pandas **as** pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

**from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.naive\_bayes **import** GaussianNB **from** sklearn **import** metrics

df **=** pd**.**read\_csv("/content/pima\_indian.csv")

feature\_col\_names **=** ['num\_preg', 'glucose\_conc', 'diastolic\_bp', 'thickness', 'insulin', 'bmi', 'diab\_pred', 'age'] predicted\_class\_names **=** ['diabetes']

X **=** df[feature\_col\_names]**.**values

y **=** df[predicted\_class\_names]**.**values print(df**.**head)

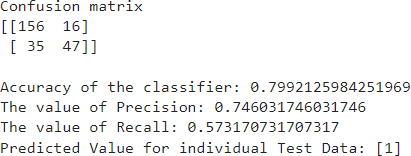
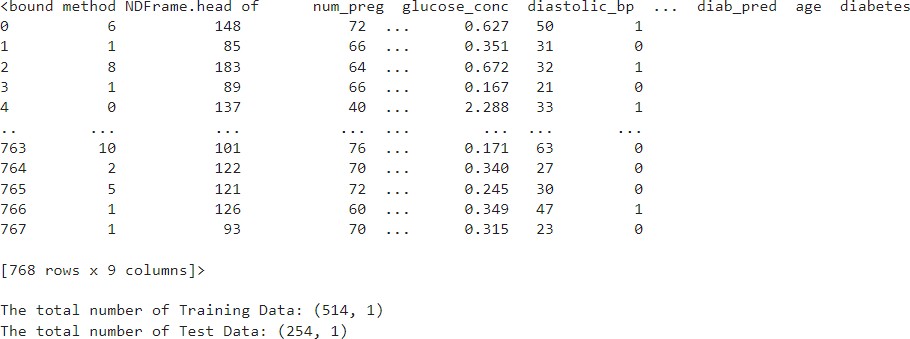
xtrain,xtest,ytrain,ytest**=**train\_test\_split(X,y,test\_size**=**0.33) print ('\nThe total number of Training Data:',ytrain**.**shape) print ('The total number of Test Data:',ytest**.**shape)

clf **=** GaussianNB()**.**fit(xtrain,ytrain**.**ravel()) predicted **=** clf**.**predict(xtest)

predictTestData**=** clf**.**predict([[6,148,72,35,0,33.6,0.627,50]]) print('\nConfusion matrix') print(metrics**.**confusion\_matrix(ytest,predicted))

print('\nAccuracy of the classifier:',metrics**.**accuracy\_score(ytest,predicted)) print('The value of Precision:', metrics**.**precision\_score(ytest,predicted)) print('The value of Recall:', metrics**.**recall\_score(ytest,predicted)) print("Predicted Value for individual Test Data:", predictTestData)

# Output:



1. **Implement the Linear Regression algorithm in order to ﬁt data points. Select appropriate data set for your experiment and draw graphs.**
   1. **Using SKlearn:**

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** pandas **as** pd

*# Importing the dataset*

dataset **=** pd**.**read\_csv('salary\_data.csv')

X **=** dataset**.**iloc[:, :**-**1]**.**values *#get a copy of dataset exclude last column*

y **=** dataset**.**iloc[:, 1]**.**values *#get array of dataset in column 1st*:

*# Splitting the dataset into the Training set and Test set*

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**1**/**3, random\_state**=**0)

*# Fitting Simple Linear Regression to the Training set* **from** sklearn.linear\_model **import** LinearRegression regressor **=** LinearRegression()

regressor**.**fit(X\_train, y\_train)

*# Visualizing the Training set results*

viz\_train **=** plt

viz\_train**.**scatter(X\_train, y\_train, color**=**'red') viz\_train**.**plot(X\_train, regressor**.**predict(X\_train), color**=**'blue') viz\_train**.**title('Salary VS Experience (Training set)') viz\_train**.**xlabel('Year of Experience')

viz\_train**.**ylabel('Salary') viz\_train**.**show()

*# Visualizing the Test set results*

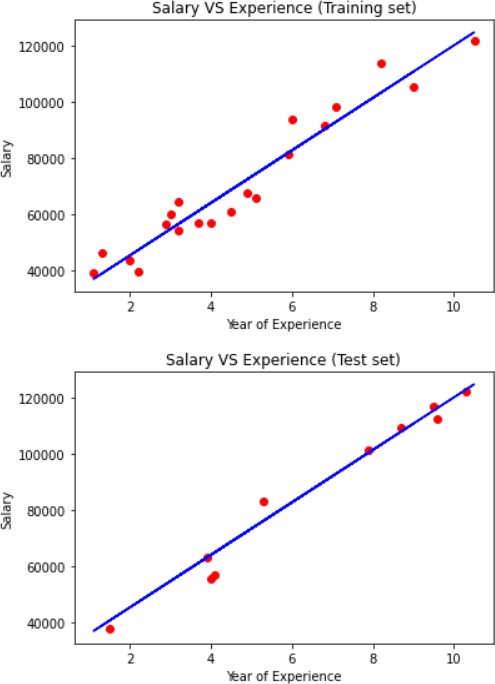
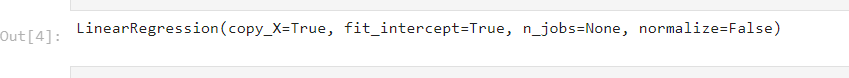
viz\_test **=** plt

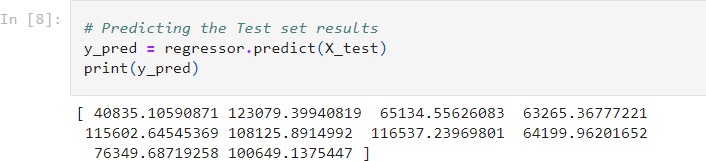
viz\_test**.**scatter(X\_test, y\_test, color**=**'red') viz\_test**.**plot(X\_train, regressor**.**predict(X\_train), color**=**'blue') viz\_test**.**title('Salary VS Experience (Test set)') viz\_test**.**xlabel('Year of Experience')

viz\_test**.**ylabel('Salary') viz\_test**.**show()

*# Predicting the Test set results* y\_pred **=** regressor**.**predict(X\_test) print(y\_pred)

**Output:**





* 1. **Without using SKlearn:**

**import** pandas **as** pd

**import** numpy **as** np

**class** LR():

**def** init (self): self**.**w **=** []

**def** fit(self, X, y):

self**.**w **=** np**.**linalg**.**solve(X**.**T@X, X**.**T@y)

**def** predict(self, X):

**return** [X@self**.**w](mailto:X@self.w)

**def** score(self, X, y):

SS\_reg **=** [np**.**sum((X@self**.**w](mailto:np.sum((X@self.w) **-** y)**\*\***2) SS\_tot **=** np**.**sum((y **-** np**.**mean(y))**\*\***2) **return** (1 **-** (SS\_reg**/**SS\_tot))

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.datasets **import** fetch\_california\_housing fetch\_california\_housing

data, labels **=** fetch\_california\_housing(return\_X\_y **= True**) data**.**shape, labels**.**shape

one **=** np**.**ones(data**.**shape[0])

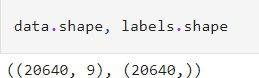
data **=** np**.**column\_stack((one, data))

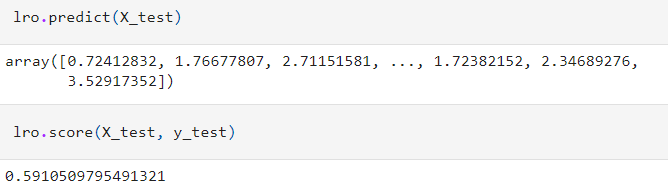
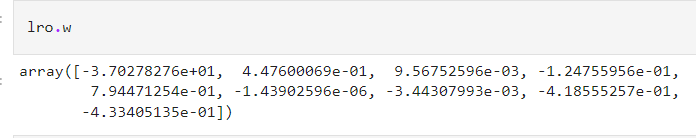
X\_train,X\_test, y\_train, y\_test **=** train\_test\_split(data, labels, train\_size **=** 0.75, random\_state **=** 42) lro **=** LR()

lro**.**fit(X\_train, y\_train) lro**.**w lro**.**predict(X\_test)

lro**.**score(X\_test, y\_test)

**Output:**





# Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

* 1. **Using built-in:**

!pip install pgmpy 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 heartDisease = pd.read\_csv('heart\_disease.csv') heartDisease = heartDisease.replace('?',np.nan)

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

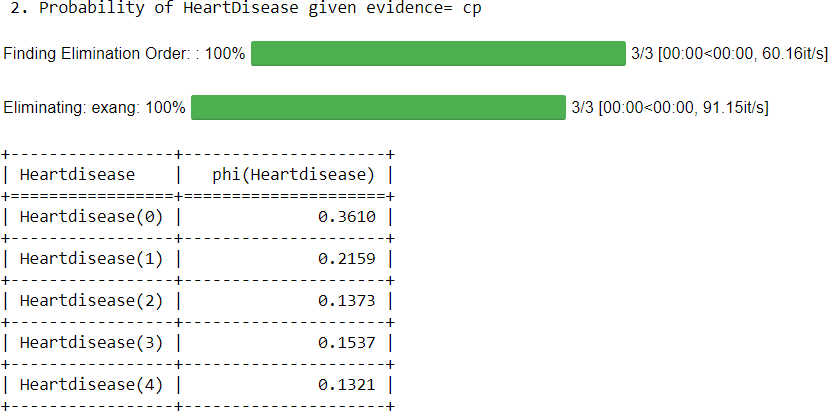
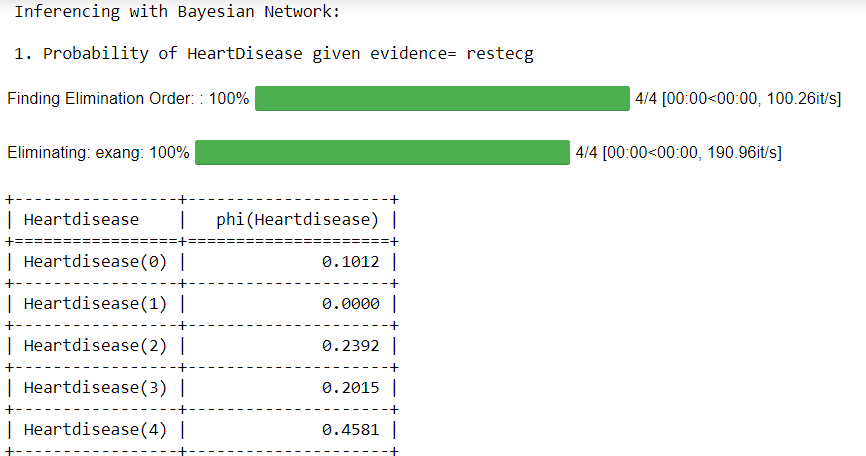
print('\n Attributes and datatypes') print(heartDisease.dtypes)

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

print('\nLearning CPD using Maximum likelihood estimators') model.fit(heartDisease,estimator=MaximumLikelihoodEstimator) print('\n Inferencing with Bayesian Network:') HeartDiseasetest\_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg') q1=HeartDiseasetest\_infer.query(variables=['Heartdisease'],evidence={'restecg':1}) print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ') q2=HeartDiseasetest\_infer.query(variables=['Heartdisease'],evidence={'cp':2}) print(q2)

**Output:**

* 1. **Without using built-in:**

import bayespy as bp import numpy as np import csv

from colorama import init

from colorama import Fore, Back, Style

init()

# Define Parameter Enum values # Age

ageEnum = {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1,

'MiddleAged': 2, 'Youth': 3, 'Teen': 4} # Gender

genderEnum = {'Male': 0, 'Female': 1} # FamilyHistory

familyHistoryEnum = {'Yes': 0, 'No': 1} # Diet(Calorie Intake)

dietEnum = {'High': 0, 'Medium': 1, 'Low': 2} # LifeStyle

lifeStyleEnum = {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3} # Cholesterol

cholesterolEnum = {'High': 0, 'BorderLine': 1, 'Normal': 2} # HeartDisease

heartDiseaseEnum = {'Yes': 0, 'No': 1} import pandas as pd

data = pd.read\_csv("heart\_disease\_data.csv") data =np.array(data, dtype='int8')

N = len(data)

# Input data column assignment

p\_age = bp.nodes.Dirichlet(1.0\*np.ones(5))

age = bp.nodes.Categorical(p\_age, plates=(N,)) age.observe(data[:, 0])

p\_gender = bp.nodes.Dirichlet(1.0\*np.ones(2))

gender = bp.nodes.Categorical(p\_gender, plates=(N,)) gender.observe(data[:, 1])

p\_familyhistory = bp.nodes.Dirichlet(1.0\*np.ones(2))

familyhistory = bp.nodes.Categorical(p\_familyhistory, plates=(N,)) familyhistory.observe(data[:, 2])

p\_diet = bp.nodes.Dirichlet(1.0\*np.ones(3))

diet = bp.nodes.Categorical(p\_diet, plates=(N,)) diet.observe(data[:, 3])

p\_lifestyle = bp.nodes.Dirichlet(1.0\*np.ones(4))

lifestyle = bp.nodes.Categorical(p\_lifestyle, plates=(N,)) lifestyle.observe(data[:, 4])

p\_cholesterol = bp.nodes.Dirichlet(1.0\*np.ones(3))

cholesterol = bp.nodes.Categorical(p\_cholesterol, plates=(N,)) cholesterol.observe(data[:, 5])

p\_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3)) heartdisease = bp.nodes.MultiMixture(

[age, gender, familyhistory, diet, lifestyle, cholesterol], bp.nodes.Categorical, p\_heartdisease) heartdisease.observe(data[:, 6])

p\_heartdisease.update() m = 0

while m == 0: print("\n")

res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))), int(input('Enter Gender: ' + str(genderEnum))), int(input('Enter FamilyHistory: ' + str(familyHistoryEnum))), int(input('Enter dietEnum: ' + str(

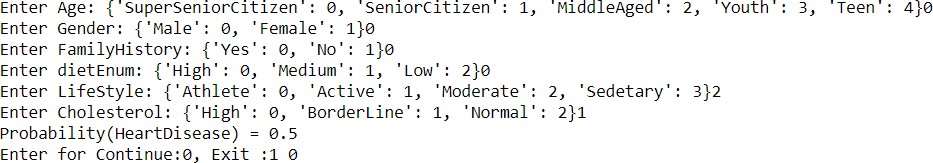
dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter Cholesterol: ' + str(cholesterolEnum)))], bp.nodes.Categorical, p\_heartdisease).get\_moments()[0][heartDiseaseEnum['Yes']]

print("Probability(HeartDisease) = " + str(res))

# print(Style.RESET\_ALL)

m = int(input("Enter for Continue:0, Exit :1 "))

**Output:**



# Apply k-Means algorithm to cluster a set of data stored in a .CSV ﬁle

* 1. **Using built-in:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import MinMaxScaler from matplotlib import pyplot as plt

%matplotlib inline

df = pd.read\_csv('income.csv') df.head(10)

scaler = MinMaxScaler() scaler.fit(df[['Age']])

df[['Age']] = scaler.transform(df[['Age']])

scaler.fit(df[['Income($)']])

df[['Income($)']] = scaler.transform(df[['Income($)']]) df.head(10)

plt.scatter(df['Age'], df['Income($)'])

k\_range = range(1, 11) sse = []

for k in k\_range:

kmc = KMeans(n\_clusters=k) kmc.fit(df[['Age', 'Income($)']]) sse.append(kmc.inertia\_)

plt.xlabel = 'Number of Clusters' plt.ylabel = 'Sum of Squared Errors' plt.plot(k\_range, sse)

km = KMeans(n\_clusters=3) km

df0 = df[df.cluster == 0] df0

df1 = df[df.cluster == 1] df1

df2 = df[df.cluster == 2] df2

p1 = plt.scatter(df0['Age'], df0['Income($)'], marker='+', color='red') p2 = plt.scatter(df1['Age'], df1['Income($)'], marker='\*', color='blue') p3 = plt.scatter(df2['Age'], df2['Income($)'], marker='^', color='green')

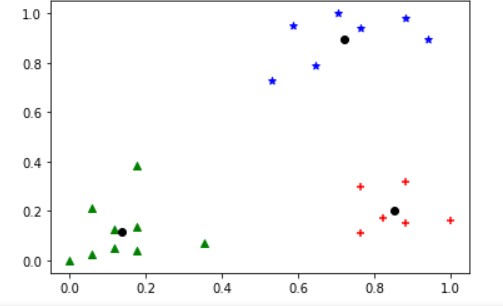
c = plt.scatter(km.cluster\_centers\_[:,0], km.cluster\_centers\_[:,1], color='black') plt.xlabel('Age')

plt.ylabel('Income($)')

plt.legend((p1, p2, p3, c),

('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroid'))

**Output:**



* 1. **Without using built-in:**

import math; import sys;

import pandas as pd import numpy as np

from random import choice from matplotlib import pyplot

from random import shuffle, uniform; def ReadData(fileName):

f = open(fileName,'r') lines = f.read().splitlines()

f.close()

items = []

for i in range(1,len(lines)): line = lines[i].split(',') itemFeatures = []

for j in range(len(line)-1): v = float(line[j])

itemFeatures.append(v) items.append(itemFeatures)

shuffle(items) return items

def FindColMinMax(items): n = len(items[0])

minima = [float('inf') for i in range(n)] maxima = [float('-inf') -1 for i in range(n)] for item in items:

for f in range(len(item)): if(item[f] < minima[f]):

minima[f] = item[f] if(item[f] > maxima[f]):

maxima[f] = item[f] return minima,maxima

def EuclideanDistance(x,y):

S = 0

for i in range(len(x)):

S += math.pow(x[i]-y[i],2) return math.sqrt(S)

def InitializeMeans(items,k,cMin,cMax): f = len(items[0])

means = [[0 for i in range(f)] for j in range(k)] for mean in means:

for i in range(len(mean)):

mean[i] = uniform(cMin[i]+1,cMax[i]-1) return means

def UpdateMean(n,mean,item): for i in range(len(mean)):

m = mean[i]

m = (m\*(n-1)+item[i])/float(n) mean[i] = round(m,3)

return mean

def FindClusters(means,items):

clusters = [[] for i in range(len(means))] for item in items:

index = Classify(means,item) clusters[index].append(item)

return clusters

def Classify(means,item): minimum = float('inf');

index = -1

for i in range(len(means)):

dis = EuclideanDistance(item,means[i]) if(dis < minimum):

minimum = dis index = i

return index

def CalculateMeans(k,items,maxIterations=100000): cMin, cMax = FindColMinMax(items)

means = InitializeMeans(items,k,cMin,cMax) clusterSizes = [0 for i in range(len(means))] belongsTo = [0 for i in range(len(items))]

for e in range(maxIterations): noChange = True;

for i in range(len(items)): item = items[i];

index = Classify(means,item) clusterSizes[index] += 1 cSize = clusterSizes[index]

means[index] = UpdateMean(cSize,means[index],item) if(index != belongsTo[i]):

noChange = False belongsTo[i] = index

if (noChange): break

return means

def CutToTwoFeatures(items,indexA,indexB): n = len(items)

X = []

for i in range(n): item = items[i]

newItem = [item[indexA],item[indexB]] X.append(newItem)

return X

def PlotClusters(clusters): n = len(clusters)

X = [[] for i in range(n)] for i in range(n):

cluster = clusters[i] for item in cluster:

X[i].append(item) colors = ['r','b','g','c','m','y'] for x in X:

c = choice(colors) colors.remove(c) Xa = []

Xb = []

for item in x: Xa.append(item[0]) Xb.append(item[1])

pyplot.plot(Xa,Xb,'o',color=c) pyplot.show()

def main():

items = ReadData('data.txt') k = 3

items = CutToTwoFeatures(items,2,3)

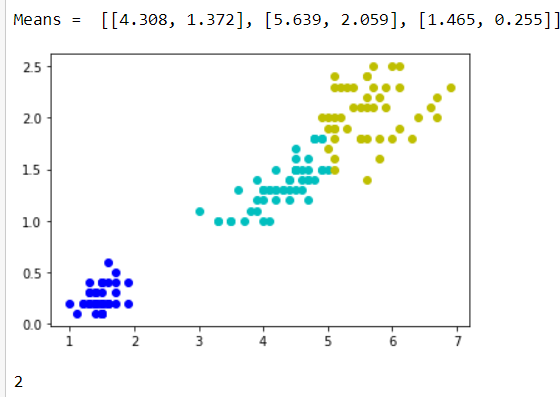
print(items)

means = CalculateMeans(k,items) print("\nMeans = ", means)

clusters = FindClusters(means,items) PlotClusters(clusters)

newItem = [1.5,0.2] print(Classify(means,newItem))

if name == " main ": main()

**Output:**

# Apply EM algorithm to cluster a set of data stored in a .CSV ﬁle. Compare the results of k-Means algorithm and EM algorithm.

import matplotlib.pyplot as plt from sklearn import datasets

from sklearn.cluster import KMeans import sklearn.metrics as sm 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']

model = KMeans(n\_clusters=3) model.fit(X) plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black']) # Plot the Original Classifications plt.subplot(1, 2, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y.Targets], s=40) plt.title('Real Classification')

plt.xlabel('Petal Length') plt.ylabel('Petal Width')

# Plot the Models Classifications plt.subplot(1, 2, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_], s=40) plt.title('K Mean Classification')

plt.xlabel('Petal Length') plt.ylabel('Petal Width')

print('The accuracy score of K-Mean: ',sm.accuracy\_score(y, model.labels\_)) print('The Confusion matrixof K-Mean: ',sm.confusion\_matrix(y, model.labels\_))

from sklearn import preprocessing scaler = preprocessing.StandardScaler() scaler.fit(X)

xsa = scaler.transform(X)

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

#xs.sample(5)

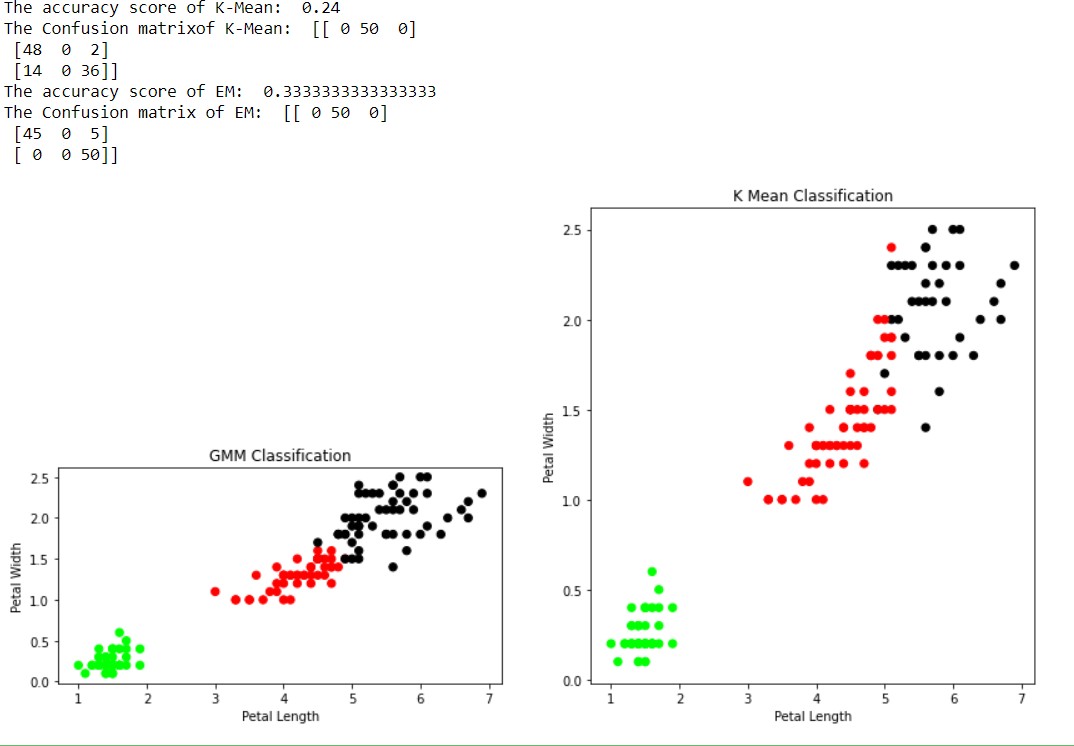
from sklearn.mixture import GaussianMixture gmm = GaussianMixture(n\_components=3) gmm.fit(xs)

y\_gmm = gmm.predict(xs) #y\_cluster\_gmm plt.subplot(2, 2, 3)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y\_gmm], s=40) plt.title('GMM Classification')

plt.xlabel('Petal Length') plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy\_score(y, y\_gmm)) print('The Confusion matrix of EM: ',sm.confusion\_matrix(y, y\_gmm))

**Output:**

# Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

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

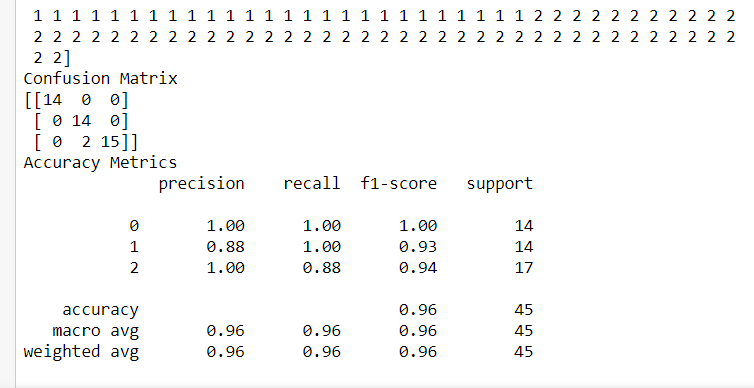
print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width') print(x)

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 Training the model and Nearest nighbors 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:**

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

1. **Using built-in:**

import numpy as np

from bokeh.plotting import figure, show, output\_notebook from bokeh.layouts import gridplot

from bokeh.io import push\_notebook

def local\_regression(x0, X, Y, tau):# add bias term

x0 = np.r\_[1, x0] # Add one to avoid the loss in information X = np.c\_[np.ones(len(X)), X]

# fit model: normal equations with kernel

xw = X.T \* radial\_kernel(x0, X, tau) # XTranspose \* W

beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot Product

# predict value

return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction def radial\_kernel(x0, X, tau):

return np.exp(np.sum((X - x0) \*\* 2, axis=1) / (-2 \* tau \* tau)) # Weight or Radial Kernal Bias Function

n = 1000

# generate dataset

X = np.linspace(-3, 3, num=n)

print("The Data Set ( 10 Samples) X :\n",X[1:10]) Y = np.log(np.abs(X \*\* 2 - 1) + .5)

print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10]) # jitter X

X += np.random.normal(scale=.1, size=n) print("Normalised (10 Samples) X :\n",X[1:10])

domain = np.linspace(-3, 3, num=300)

print(" Xo Domain Space(10 Samples) :\n",domain[1:10])

def plot\_lwr(tau):

# prediction through regression

prediction = [local\_regression(x0, X, Y, tau) for x0 in domain] plot = figure(plot\_width=400, plot\_height=400) plot.title.text='tau=%g' % tau

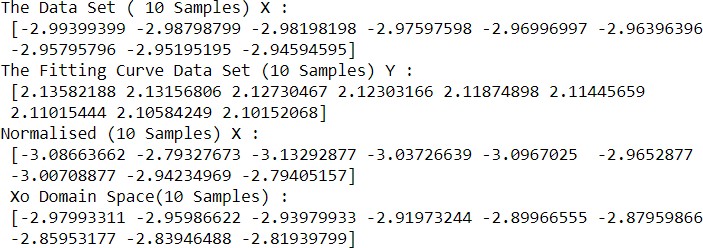
plot.scatter(X, Y, alpha=.3)

plot.line(domain, prediction, line\_width=2, color='red') return plot

show(gridplot([ [plot\_lwr(10.), plot\_lwr(1.)],

[plot\_lwr(0.1), plot\_lwr(0.01)]]))

**Output:**



1. **Without using built-in:**

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))) 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

data = pd.read\_csv('10-dataset.csv') bill = np.array(data.total\_bill)

tip = np.array(data.tip)

#preparing and add 1 in bill mbill = np.mat(bill)

mtip = np.mat(tip)

m= np.shape(mbill)[1] one = np.mat(np.ones(m))

X = np.hstack((one.T,mbill.T))

ypred = localWeightRegression(X,mtip,0.5) 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:**

