VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING

Submitted by

KIRAN M K (1BM19CS073)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
May-2022 to July-2022

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

<u>CERTIFICATE</u>

This is to certify that the Lab work entitled "**LAB COURSE OF MACHINE LEARNING**" carried out by **KIRAN M K (1BM19CS073),** who is a 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. Asha G R
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

`

Index Sheet

Sl. No.	Experiment Title	Page No.
1.	Find S Algorithm	4
2.	Candidate Elimination Algorithm	6
3.	ID3 algorithm	8
4.	Linear Regression	15
5.	Naive Bayes Algorithm	19
6.	Bayesian Network	24
7.	K-Means Algorithm	28
8.	Compare EM with K-Means	34
9.	K-Nearest Neighbors	35
10.	Locally Weighted Regression	38

Course Outcome

CO1	Ability to apply different learning algorithms
CO2	Ability to analyze 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 algorithms.

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

```
import pandas as pd

import numpy as np


df = pd.read_csv('/content/Lab1_Doc.csv')

hypo = []

for i in range(len(df.columns)-1):

  if type(i) == str:

    hypo.append("Null")

  elif type(i) == int:

    hypo.append(0)

  else:

    hypo.append(0.0)


for i in range(len(df)):

  if df.loc[i, 'Poisonous'] == "YES":

    for j in range(len(hypo)):

      if hypo[j] == "Null":

        hypo[j] = df.iloc[i, j]

      elif hypo[j] != df.iloc[i,j]:

        hypo[j] = "?"

print("The hypothesis for the given data: ", hypo)
```

OUTPUT:

	Color	Toughness	Fungus	Appearance	Poisonous
0	GREEN	HARD	NO	WRINKLED	YES
1	GREEN	HARD	YES	SMOOTH	NO
2	BROWN	SOFT	NO	WRINKLED	NO
3	ORANGE	HARD	NO	WRINKLED	YES
4	GREEN	SOFT	YES	SMOOTH	YES
5	GREEN	HARD	YES	WRINKLED	YES
6	ORANGE	HARD	NO	WRINKLED	YES

```
The hypothesis for the given data:  ['?', '?', '?', '?']
```

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.**

```
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 general_h")

    print("\nSpecific Hypothesis: ", specific_h)

    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]

    print("\nGeneral Hypothesis: ",general_h)


    for i, h in enumerate(concepts):

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

        if target[i] == "yes":

            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] == "no":

            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_h)
        print( 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:

```
Initialization of specific_h and general_h

Specific Hypothesis:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

General Hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Instance 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Instance 3 is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Negative
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]


Instance 4 is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Instance is Positive
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

3. Demonstration of ID3 algorithm

```
import pandas as pd

import numpy as np

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.tree import plot_tree

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import ConfusionMatrixDisplay

import matplotlib.pyplot as plt

data = load_iris()

df = pd.DataFrame(data.data, columns = data.feature_names)

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()


X_train, test_x, y_train, test_lab = train_test_split(x,y,test_size =
0.4,random_state = 42)

clf = DecisionTreeClassifier(max_depth =4, random_state = 42,
criterion='entropy')

clf.fit(X_train, y_train)


test_pred = clf.predict(test_x)

ConfusionMatrixDisplay.from_predictions(test_lab, test_pred,
labels=['setosa', 'versicolor', 'virginica'])

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

_ = plot_tree(clf,

                   feature_names=data.feature_names,
```

```
                      class_names=data.target_names,
                      filled=True)
```

Without Sklearn

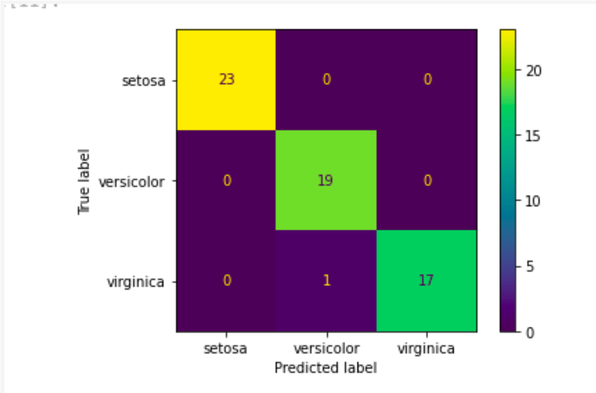
```
import math
import csv
def load_csv(filename):
  lines=csv.reader(open(filename,"r"))
  dataset = list(lines)
  headers = dataset.pop(0)
  return dataset,headers

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]]])
```

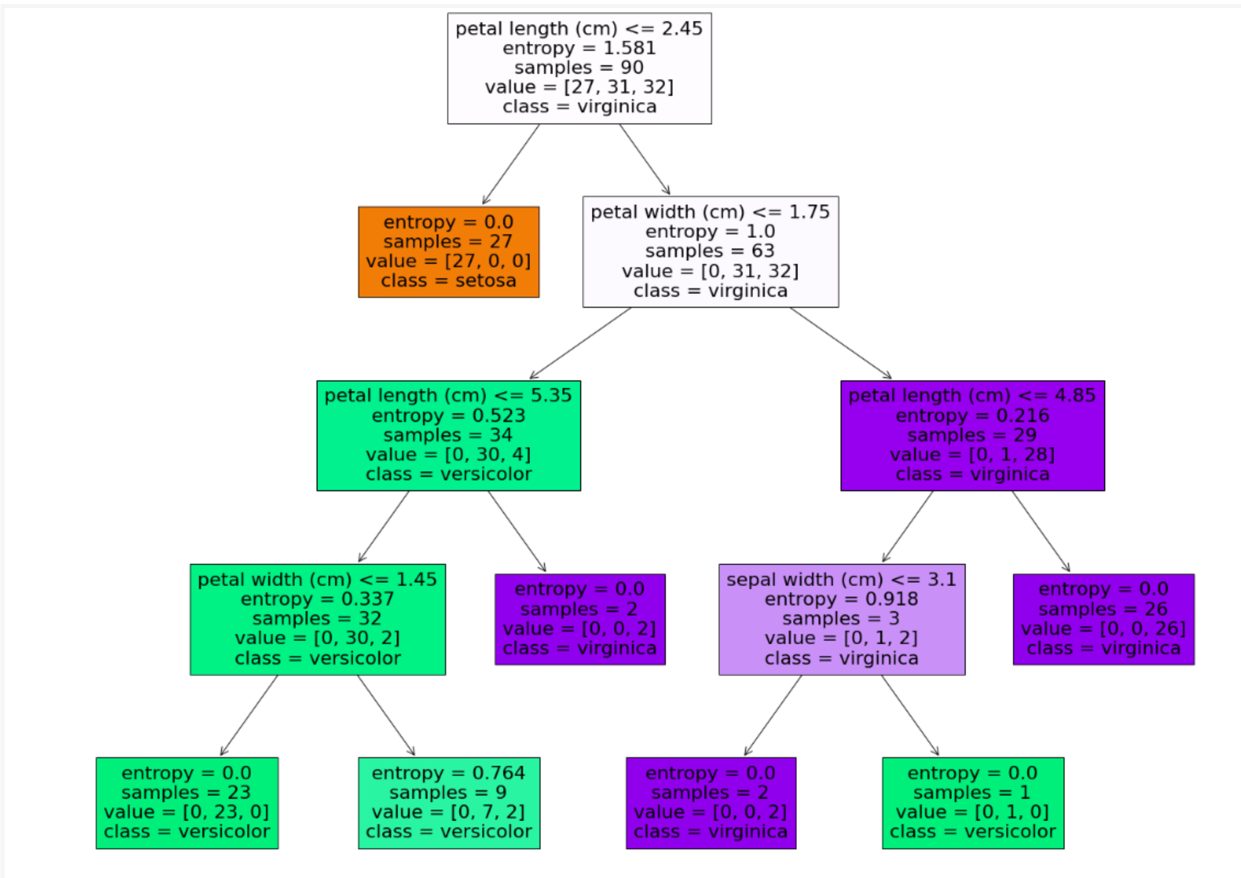
```
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy

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_test.csv")

for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:")
    classify(node1,xtest,features)
```

OUTPUT:



setosa
versicolor
virginica
True label
23
0
0
0
19
0
0
1
17
setosa
versicolor
virginica
Predicted label
20
15
10
5
0



petal length (cm) <= 2.45
entropy = 1.581
samples = 90
value = [27, 31, 32]
class = virginica
entropy = 0.0
samples = 27
value = [27, 0, 0]
class = setosa
petal width (cm) <= 1.75
entropy = 1.0
samples = 63
value = [0, 31, 32]
class = virginica
petal length (cm) <= 5.35
entropy = 0.523
samples = 34
value = [0, 30, 4]
class = versicolor
petal length (cm) <= 4.85
entropy = 0.216
samples = 29
value = [0, 1, 28]
class = virginica
petal width (cm) <= 1.45
entropy = 0.337
samples = 32
value = [0, 30, 2]
class = versicolor
entropy = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
sepal width (cm) <= 3.1
entropy = 0.918
samples = 3
value = [0, 1, 2]
class = virginica
entropy = 0.0
samples = 26
value = [0, 0, 26]
class = virginica
entropy = 0.0
samples = 23
value = [0, 23, 0]
class = versicolor
entropy = 0.764
samples = 9
value = [0, 7, 2]
class = versicolor
entropy = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
entropy = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor

without sklearn:

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   sunny
     Humidity
       normal
         yes
       high
         no
   overcast
     yes
   rain
     Wind
       strong
         no
       weak
         yes
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance:
no
The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance:
yes
```

4. Linear Regression without sklearn:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_california_housing

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
  def score(self, X, y):
    SS_reg = np.sum((X@self.w - y)**2)
    SS_tot = np.sum((y - np.mean(y))**2)
    return (1 - (SS_reg/SS_tot))
data, labels = fetch_california_housing(return_X_y = True)
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.predict(X_test)
lro.score(X_test, y_test)
```

OUTPUT:

```
[30]:  0.5910509795491321
```

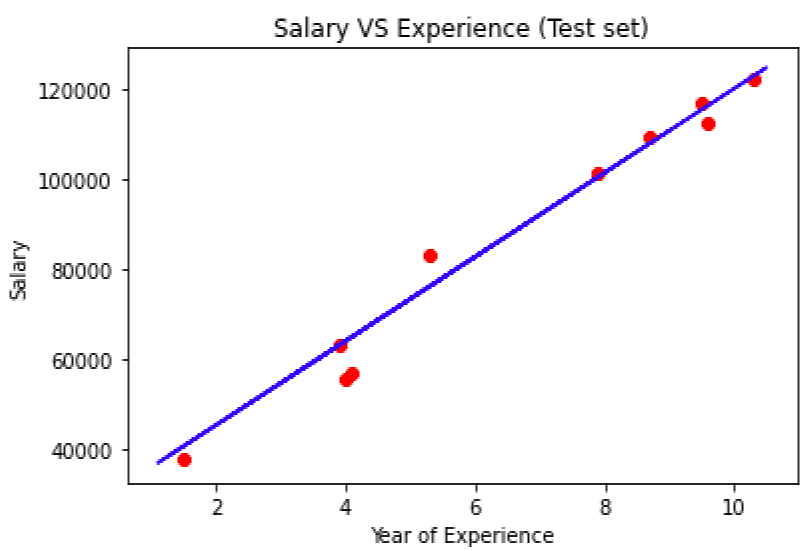
with sklearn:

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

# Importing the dataset
#dataset = pd.read_csv('181105_missing-data.csv')
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)

# Predicting the Test set results
y_pred = regressor.predict(X_test)

# 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()
```

OUTPUT:


Salary VS Experience (Training set)
120000
100000
80000
60000
40000
Salary
2
4
6
8
10
Year of Experience


Salary VS Experience (Test set)
120000
100000
80000
60000
40000
Salary
2
4
6
8
10
Year of Experience

5. Naive Bayes Algorithm without sklearn:

```
import pandas as pd

data = pd.read_csv('PlayTennis.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:]
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:

```
Predicted values: ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
Actual values: ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

Total number of testing instances in the dataset: 6
Number of correct predictions: 4
Number of wrong predictions: 2

Accuracy of Bayes Classifier: 0.6666666666666666
```

With sklearn:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn import metrics


df = pd.read_csv("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 # these are factors for the prediction

y = df[predicted_class_names].values # this is what we want to predict


#splitting the dataset into train and test data

print(df.head)

xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)


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

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



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


clf = GaussianNB().fit(xtrain,ytrain.ravel())

predicted = clf.predict(xtest)

predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])


#printing Confusion matrix, accuracy, Precision and Recall


print('\n Confusion matrix')

print(metrics.confusion_matrix(ytest,predicted))
```

print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))

print('\n The value of Precision', metrics.precision_score(ytest,predicted))

print('\n The value of Recall', metrics.recall_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)

OUTPUT:

```
 the total number of Training Data : (514, 1)

 the total number of Test Data : (254, 1)

 Confusion matrix
[[147  26]
 [ 28  53]]

 Accuracy of the classifier is 0.7874015748031497

 The value of Precision 0.6708860759493671

 The value of Recall 0.654320987654321
Predicted Value for individual Test Data: [1]
```

6. Bayesian Network

without in built

```
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

trainingData = pd.read_csv('/content/bayesian-dataset.csv')

trainingData = trainingData.replace('?',np.nan)

print('The sample instances from the dataset are:')

print(trainingData.head())

print('\n Attributes and datatypes: ')

print(trainingData.dtypes)
```

```
model =
BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','h
eartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdi
sease','chol')])

print('\n Learning CPD using Maximum likelihood estimators')

model.fit(trainingData,estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')

HeartDiseasetest_infer = VariableElimination(model)

print('\n 1.Probability of HeartDisease given evidence = restecg (Rest
ECG): 1')

q1 = HeartDiseasetest_infer.query(variables = ['heartdisease'],
evidence={'restecg':1})

print(q1)

print('\n 2.Probability of HeartDisease given evidence = chol
(Cholestorol): 100 ')

q2 = HeartDiseasetest_infer.query(variables = ['heartdisease'],
evidence={'chol':100})

print(q2)
```

```
 1.Probability of HeartDisease given evidence = restecg (Rest ECG): 1
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/4 [00:00<?, ?it/s]
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+
```

```
  UserWarning,
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/4 [00:00<?, ?it/s]
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              1.0000 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.0000 |
+-----------------+---------------------+
| heartdisease(3) |              0.0000 |
+-----------------+---------------------+
| heartdisease(4) |              0.0000 |
+-----------------+---------------------+
```

without in built:

```
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")

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:

```
Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4}2
Enter Gender: {'Male': 0, 'Female': 1}0
Enter FamilyHistory: {'Yes': 0, 'No': 1}0
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}0
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}0
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}0
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1 1
```

7. K Means Algorithm without sklearn:

```
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)
```

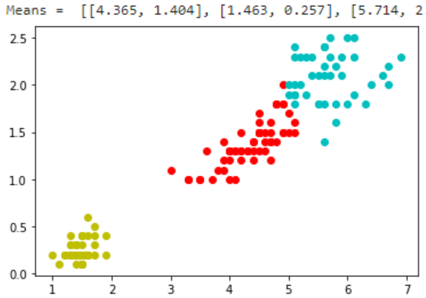
```
def InitializeMeans(items,k,cMin,cMax):
    f = len(items[0])
    means = [[0 for i in range(f)] for j in range(k)]

    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)
```

```python
            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:

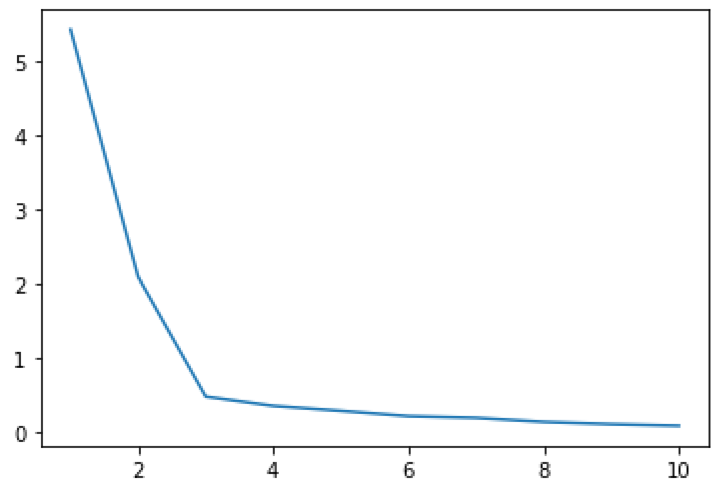
```
[[1.3, 0.2], [1.3, 0.3], [6.7, 2.0], [1.6, 0.4], [4.5, 1.5], [1.5, 0.4], [3.8, 1.1], [1.2, 0.2], [5.2, 2.0], [4.5, 1.5], [5.0, 1.9], [3.5, 1.0], [6.3,
1.8], [6.9, 2.3], [1.6, 0.2], [6.0, 1.8], [5.1, 2.3], [1.5, 0.1], [5.1, 1.9], [1.7, 0.5], [4.0, 1.3], [4.8, 1.8], [1.4, 0.3], [1.5, 0.1], [5.1, 1.5],
[4.0, 1.3], [1.4, 0.2], [1.4, 0.2], [4.5, 1.7], [1.4, 0.2], [4.1, 1.0], [5.4, 2.3], [1.4, 0.3], [6.6, 2.1], [1.3, 0.2], [3.5, 1.0], [4.0, 1.0], [4.5,
1.3], [1.4, 0.2], [6.1, 1.9], [5.4, 2.1], [3.9, 1.1], [5.9, 2.3], [1.5, 0.2], [5.6, 1.4], [1.7, 0.3], [4.4, 1.4], [4.7, 1.2], [1.5, 0.2], [1.5, 0.4],
[1.5, 0.2], [6.1, 2.5], [4.5, 1.5], [1.5, 0.2], [5.1, 2.4], [1.9, 0.4], [5.7, 2.5], [3.3, 1.0], [5.0, 2.0], [5.6, 1.8], [5.3, 2.3], [6.1, 2.3], [5.1,
1.6], [4.2, 1.3], [5.9, 2.1], [4.6, 1.4], [6.7, 2.2], [4.2, 1.2], [3.7, 1.0], [1.9, 0.2], [4.5, 1.5], [4.2, 1.5], [5.6, 2.2], [5.6, 2.1], [1.4, 0.1],
[4.9, 2.0], [4.9, 1.5], [5.7, 2.1], [5.2, 2.3], [5.1, 1.9], [4.7, 1.4], [5.6, 2.4], [3.3, 1.0], [4.3, 1.3], [5.6, 2.4], [1.6, 0.2], [1.0, 0.2], [1.2,
0.2], [6.4, 2.0], [4.4, 1.2], [5.1, 1.8], [1.3, 0.3], [3.9, 1.4], [4.7, 1.6], [4.0, 1.2], [1.6, 0.2], [4.7, 1.5], [5.1, 2.0], [4.9, 1.8], [4.4, 1.4],
[4.9, 1.8], [1.3, 0.4], [1.3, 0.2], [4.0, 1.3], [1.7, 0.4], [5.3, 1.9], [4.6, 1.5], [1.1, 0.1], [5.7, 2.3], [4.8, 1.8], [3.6, 1.3], [1.5, 0.2], [1.4,
0.3], [1.3, 0.2], [4.2, 1.3], [4.5, 1.6], [1.4, 0.2], [5.8, 1.8], [1.6, 0.6], [1.7, 0.2], [4.6, 1.3], [4.8, 1.4], [4.7, 1.4], [1.4, 0.2], [4.1, 1.3],
[5.5, 1.8], [4.3, 1.3], [1.5, 0.1], [5.8, 2.2], [1.6, 0.2], [4.8, 1.8], [5.8, 1.6], [1.5, 0.4], [4.4, 1.3], [3.0, 1.1], [4.1, 1.3], [5.0, 1.5], [1.6,
0.2], [5.0, 1.7], [1.5, 0.3], [5.5, 1.8], [1.5, 0.1], [1.4, 0.2], [3.9, 1.2], [1.5, 0.2], [4.9, 1.5], [5.5, 2.1], [4.5, 1.5], [6.0, 2.5]]

Means =  [[4.365, 1.404], [1.463, 0.257], [5.714, 2.085]]
```



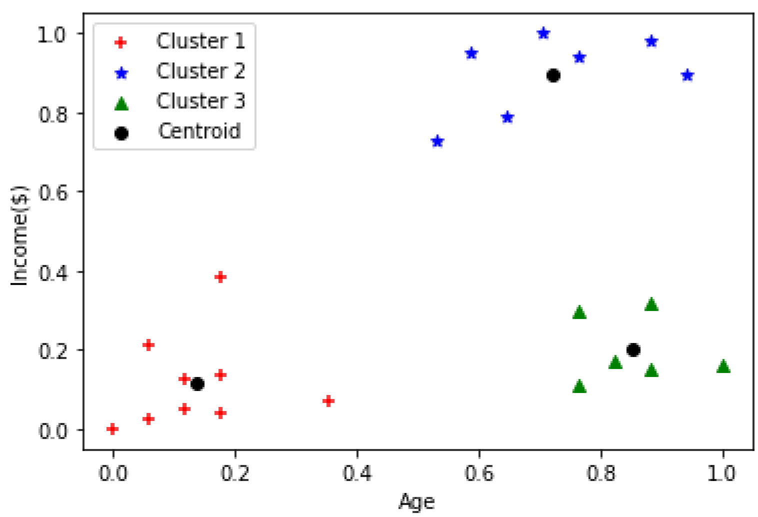
with sklearn;

```
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($)']])
    sse.append(kmc.inertia_)
sse
plt.xlabel = 'Number of Clusters'
plt.ylabel = 'Sum of Squared Errors'
plt.plot(k_range, sse)
km = KMeans(n_clusters=3)
km
y_predict = km.fit_predict(df[['Age', 'Income($)']])
y_predict
df['cluster'] = y_predict
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:



Therefore, the elbow point is 3

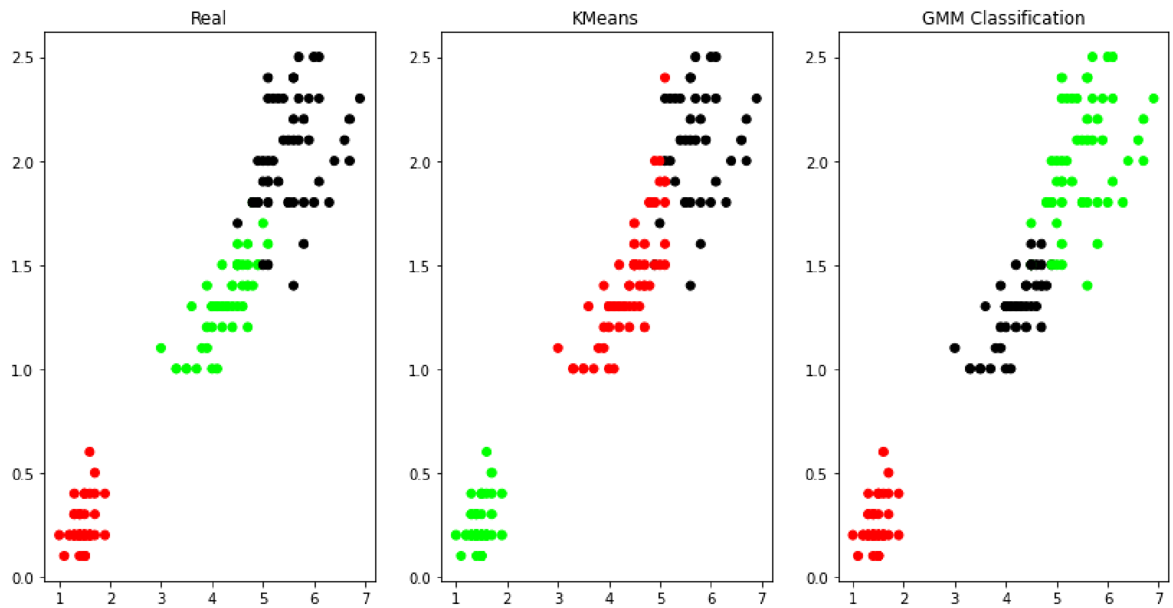


8. Comparing K Means and EM algorithm

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

names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']

dataset = pd.read_csv("/content/dataset.csv", names=names)

X = dataset.iloc[:, :-1]

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])


plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])


model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])

print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y, model.labels_))


gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])

print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

OUTPUT:

```
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:
 [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
The accuracy score of EM:  0.36666666666666664
The Confusion matrix of EM:
  [[50  0  0]
 [ 0  5 45]
 [ 0 50  0]]
```

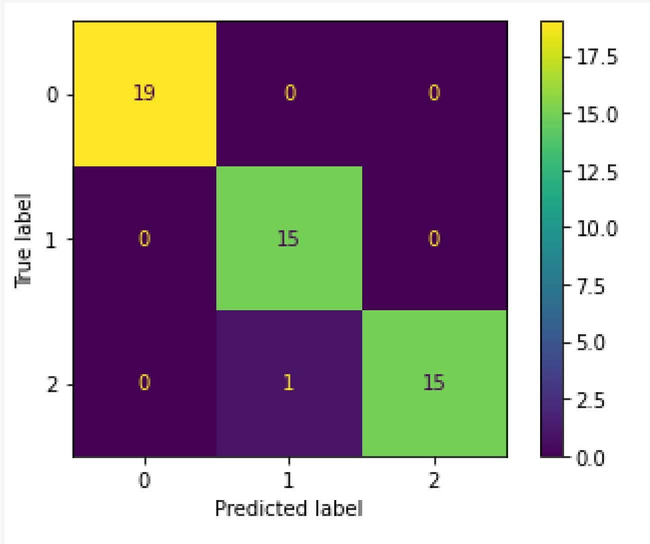


9. K Nearest Neighbors

```
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('target')
print(Y)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=42)

#Training the model with Nearest nighbors K=3
knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
from sklearn.metrics import accuracy_score

y_pred=knn.predict(X_test)
matrix =confusion_matrix(y_test,y_pred)
print(" Confusion matrix:\n",matrix)
print(" Correct predicition",accuracy_score(y_test,y_pred))
print(" Wrong predicition",(1-accuracy_score(y_test,y_pred)))
print(' Accuracy Metrics')
print(classification_report(y_test,y_pred))
from sklearn.metrics import ConfusionMatrixDisplay
cm_plot = ConfusionMatrixDisplay(confusion_matrix = confusion_matrix(y_test, y_pred))
cm_plot.plot()
```

OUTPUT:

```
 Confusion matrix:
 [[19  0  0]
 [ 0 15  0]
 [ 0  1 15]]
 Correct predicition 0.98
 Wrong predicition 0.020000000000000018
 Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.94      1.00      0.97        15
           2       1.00      0.94      0.97        16

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50
```

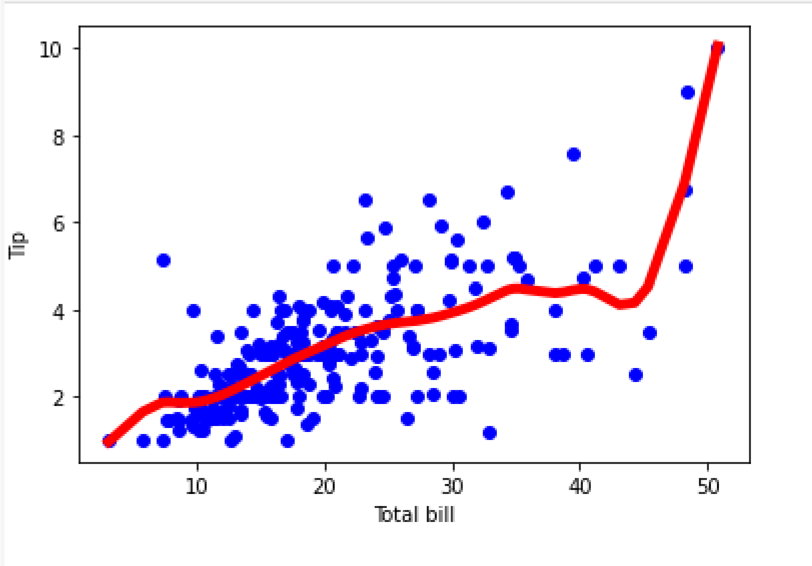


19
0
0
0
15
0
0
1
15
True label
0
1
2
Predicted label
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0

10. Locally weighted regression without sklearn

```
from numpy import *
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr
def kernel(point,xmat, k):
 m,n = np1.shape(xmat)
 weights = np1.mat(np1.eye((m)))
 for j in range(m):
    diff = point - X[j]
    weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
 return weights
def localWeight(point,xmat,ymat,k):
        wei = kernel(point,xmat,k)
        W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
        return W
def localWeightRegression(xmat,ymat,k):
        m,n = np1.shape(xmat)
        ypred = np1.zeros(m)
        for i in range(m):
           ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
        return ypred
data = pd.read_csv('/content/tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)
#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimensional array form
m= np1.shape(mbill)[1]
# print(m) 244 data is stored in m
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
#print(X)
#set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='blue')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
```

OUTPUT:



with python libraries

```
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)]]))
```

```
The Data Set ( 10 Samples) X :
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y :
 [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
 2.11015444 2.10584249 2.10152068]
Normalised (10 Samples) X :
 [-3.12282223 -2.9216174  -3.14051918 -3.09805236 -3.08215798 -2.88090494
 -3.05412007 -3.12734019 -2.98129254]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```