VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING
(20CS6PCMAL)

Submitted by

ANAGHA ACHARYA(1BM19CS224)

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

CERTIFICATE

This is to certify that the Lab work entitled "**MACHINE LEARNING**" was carried out by **ANAGHA ACHARYA (1BM19CS224),** who is a bona fide 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 theacademic requirements in respect of the course **MACHINE LEARNING (20CS6PCMAL)** work prescribed for the said degree.

Dr. K. Panimozhi
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

`

Index Sheet

Course Outcome:-

At the end of the course the student will be able to

CO1	Ability to apply the 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 techniques.

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

Program

```
import csv
a=[]
with open('C:/Users/bmsce/Desktop/enjoysport.csv', 'r') as csvfile:
    next(csvfile)
    for row in csv.reader(csvfile):
        a.append(row)
    print(a)

print('\nTotal instances=',len(a))

num_attributes=len(a[0])-1

hypothesis=['0']*num_attributes
print('The initial hypothesis h0 is:\n',hypothesis)

for i in range(0,len(a)):
    if a[i][num_attributes]=='yes':
        print('Instance',(i+1),'is',a[i],'and is a positive instance')
        for j in range(0,num_attributes):
            if hypothesis[j]=='0' or hypothesis[j]==a[i][j]:
                hypothesis[j]=a[i][j]
            else:
                hypothesis[j]='?'
        print('The hypothesis for training instance',(i+1),'is',hypothesis,'\n')
    if a[i][num_attributes]=='no':
        print('Instance',(i+1),'is',a[i],'and is a negative instance and therefore ignored')
        print('The hypothesis for training instance',(i+1),'is',hypothesis,'\n')

print('The final hypothesis is', hypothesis)
```

enjoysport.csv

	sky	air_temp	humidity	wind	water	forecast	enjoy_sport
1	sky	air_temp	humidity	wind	water	forecast	enjoy_sport
2	sunny	warm	normal	strong	warm	same	yes
3	sunny	warm	high	strong	warm	same	yes
4	rainy	cold	high	strong	warm	change	no
5	sunny	warm	high	strong	cool	change	yes

Output

```
[['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'], ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'], ['rainy', 'cold', 'high',
'strong', 'warm', 'change', 'no'], ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']]

Total instances= 4
The initial hypothesis h0 is:
 ['0', '0', '0', '0', '0', '0']
Instance 1 is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'] and is a positive instance
The hypothesis for training instance 1 is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

Instance 2 is ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'] and is a positive instance
The hypothesis for training instance 2 is ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Instance 3 is ['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no'] and is a negative instance and therefore ignored
The hypothesis for training instance 3 is ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Instance 4 is ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes'] and is a positive instance
The hypothesis for training instance 4 is ['sunny', 'warm', '?', 'strong', '?', '?']

The final hypothesis is ['sunny', 'warm', '?', 'strong', '?', '?']
```

Program 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.

Program

```
import csv
a = []
with open('C:/Users/admin/Desktop/enjoysport.csv','r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        a.append (row)
        print(row)
num_attributes = len(a[0])-1
print("The initial value of hypothesis: ")
S = ['0'] * num_attributes
G = ['?'] * num_attributes
print ("\n The most specific hypothesis S0 : [0,0,0,0,0,0]")
print (" \n The most general hypothesis G0 : [?,?,?,?,?,?]")

for j in range(0,num_attributes):
    S[j]=a[0][j];
print("\n Candidate Elimination algorithm Hypotheses Version Space Computation\n")
temp=[]
for i in range(0,len(a)):
    if a[i][num_attributes]=='yes':
        for j in range(0,num_attributes):
            if a[i][j]!=S[j]:
                S[j]='?'
        for j in range(0,num_attributes):
            for k in range(1,len(temp)):
                if temp[k][j]!='?' and temp[k][j]!=S[j]:
                    del temp[k]
        print("--------------------------------------------------------------------------- ")
        print(" For Training Example No :{0} the hypothesis is S{0} ".format(i+1),S)
        if (len(temp)==0):
            print(" For Training Example No :{0} the hypothesis is G{0} ".format(i+1),G)
        else:
            print(" For  Positive Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)
    if a[i][num_attributes]=='no':
        for j in range(0,num_attributes):
            if S[j] != a[i][j] and S[j]!= '?':
                G[j]=S[j]
                temp.append(G)
                G = ['?'] * num_attributes
        print("--------------------------------------------------------------------------- ")
        print(" For Training Example No :{0} the hypothesis is S{0} ".format(i+1),S)
        print(" For Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)
```

enjoysport.csv

	sky	air_temp	humidity	wind	water	forecast	enjoy_sport
2	sunny	warm	normal	strong	warm	same	yes
3	sunny	warm	high	strong	warm	same	yes
4	rainy	cold	high	strong	warm	change	no
5	sunny	warm	high	strong	cool	change	yes

Output

```
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']
The initial value of hypothesis:

 The most specific hypothesis S0 : [0,0,0,0,0,0]

 The most general hypothesis G0 : [?,?,?,?,?,?]
```

```
 Candidate Elimination algorithm Hypotheses Version Space Computation

-----------------------------------------------------------------------------
 For Training Example No :1 the hypothesis is S1  ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
 For Training Example No :1 the hypothesis is G1  ['?', '?', '?', '?', '?', '?']
-----------------------------------------------------------------------------
 For Training Example No :2 the hypothesis is S2  ['sunny', 'warm', '?', 'strong', 'warm', 'same']
 For Training Example No :2 the hypothesis is G2  ['?', '?', '?', '?', '?', '?']
-----------------------------------------------------------------------------
 For Training Example No :3 the hypothesis is S3  ['sunny', 'warm', '?', 'strong', 'warm', 'same']
 For Training Example No :3 the hypothesis is G3 [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'same']]
-----------------------------------------------------------------------------
 For Training Example No :4 the hypothesis is S4  ['sunny', 'warm', '?', 'strong', '?', '?']
 For  Positive Training Example No :4 the hypothesis is G4 [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Program 3: Write a program to demonstrate the working of the decision tree-based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Program

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

class Node:
    def __init__(self,attribute):
        self.attribute=attribute
        self.children=[]
        self.answer=""
def subtables(data,col,delete):
    dic={}
    coldata=[row[col] for row in data]
    attr=list(set(coldata))

    counts=[0]*len(attr)
    r=len(data)
    c=len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col]==attr[x]:
                counts[x]+=1

    for x in range(len(attr)):
        dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
        pos=0
        for y in range(r):
            if data[y][col]==attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos]=data[y]
                pos+=1
    return attr,dic
def entropy(S):
    attr=list(set(S))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

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

```
    return sums

def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy
def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]

    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node
def print_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)
        return

    print("  "*level,node.attribute)
    for value,n in node.children:
        print("  "*(level+1),value)
        print_tree(n,level+2)

def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)
node1=build_tree(dataset,features)
```

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

id3.csv

Outlook	Temperature	Humidity	Wind	Answer
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

Output

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   rain
     Wind
       weak
         yes
       strong
         no
   sunny
     Humidity
       normal
         yes
       high
         no
   overcast
     yes
```

Program 4a: 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.

Program

```
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(r'C:\Users\admin\Desktop\Bird.csv')
feature_col_names = ['Color','Legs','Height']
predicted_class_names = ['Species']

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

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:,0] = le.fit_transform(X[:,0])
X[:,2] = le.fit_transform(X[:,2])
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([[1,3,1]])
#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("Predicted Value for individual Test Data:", predictTestData)
```

bird.csv

1	Color	Legs	Height	Species
2	White	3	Short	M
3	Green	2	Tall	M
4	Green	3	Short	M
5	White	3	Short	M
6	Green	2	Short	H
7	White	2	Tall	H
8	White	2	Tall	H
9	White	2	Short	H

Output

```
<bound method NDFrame.head of    Color  Legs Height Species
0  White     3  Short       M
1  Green     2   Tall       M
2  Green     3  Short       M
3  White     3  Short       M
4  Green     2  Short       H
5  White     2   Tall       H
6  White     2   Tall       H
7  White     2  Short       H>

 the total number of Training Data : (5, 1)

 the total number of Test Data : (3, 1)

 Confusion matrix
[[1 0]
 [1 1]]

 Accuracy of the classifier is 0.6666666666666666
Predicted Value for individual Test Data: ['M']
```

Program 5: Implement the Linear Regression algorithmin order to fit data points. Select appropriate data set for your experiment and draw graphs.

Program

```
import numpy as np
import pandas as pd
import csv
import matplotlib.pyplot as plt
dataset=pd.read_csv(r"C:\Users\admin\Downloads\canada_per_capita_income.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
dataset.head()

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)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
y_pred = regressor.predict(X_test)
pd.DataFrame(data={'Actuals': y_test, 'Predictions': y_pred})

plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()

plt.figure(figsize=(4,3))
plt.scatter(y_test,y_pred)
plt.plot([0,50],[0,50],'—k')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.show(
```

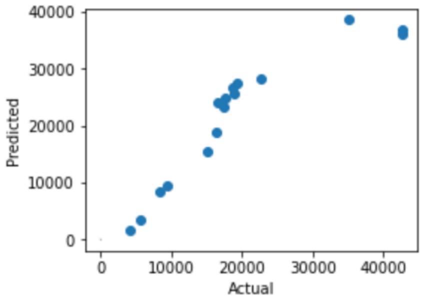
canada_per_capita_income.csv

1	year	per capita income (US$)
2	1970	3399.299037
3	1971	3768.297935
4	1972	4251.175484
5	1973	4804.463248
6	1974	5576.514583
7	1975	5998.144346
8	1976	7062.131392
9	1977	7100.12617
10	1978	7247.967035

…..

Output

	Actuals	Predictions
0	16622.671870	24043.751185
1	22739.426280	28343.029760
2	18987.382410	25763.462615
3	5576.514583	3407.214025
4	15080.283450	15445.194035
5	9434.390652	9426.204030
6	42665.255970	36081.731195
7	18601.397240	26623.318330
8	16412.083090	18884.616895
9	8355.968120	8566.348315
10	17310.757750	23183.895470
11	19232.175560	27483.174045
12	17581.024140	24903.606900
13	42676.468370	36941.586910
14	4251.175484	1687.502595
15	35175.188980	38661.298340



Program 6: Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

Program

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
heartDisease = pd.read_csv("C:/Users/admin/Downloads/heart.csv")
heartDisease = heartDisease.replace('?',np.nan)
model = BayesianModel([('age','heartdisease'),('sex','heartdisease'),(
'exang','heartdisease'),('cp','heartdisease'),('heartdisease',
'restecg'),('heartdisease','chol')])
#Learning CPDs using Maximum Likelihood Estimators
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

#Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
#computing the Probability of HeartDisease given restecg
print('\n 1.Probability of HeartDisease given evidence= restecg :1')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
#computing the Probability of HeartDisease given cp
print('\n 2.Probability of HeartDisease given evidence= cp:2 ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

heart.csv

1	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
2	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
3	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
4	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
5	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
6	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
7	56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
8	62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
9	57	0	4	120	354	0	0	163	1	0.6	1	0	3	0
10	63	1	4	130	254	0	2	147	0	1.4	2	1	7	2

…

Output

```
Inferencing with Bayesian Network:

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

2.Probability of HeartDisease given evidence= cp:2
 0%|          | 0/3 [00:00<?, ?it/s]
 0%|          | 0/3 [00:00<?, ?it/s]
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

Program 7: Apply k-Means algorithm to cluster a set of data stored in a.CSV file.

Program

```
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('C:/Users/admin/downloads/income.csv')
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
df.head()

df0 = df[df.cluster == 0]
df1 = df[df.cluster == 1]
df2 = df[df.cluster == 2]
km.cluster_centers_

p1 = plt.scatter(df0['Age'], df0['Income($)'],color='red')
p2 = plt.scatter(df1['Age'], df1['Income($)'],color='blue')
p3 = plt.scatter(df2['Age'], df2['Income($)'],color='green')
c = plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], color='black')
plt.legend((p1, p2, p3, c),
        ('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroid'))
```

income.csv

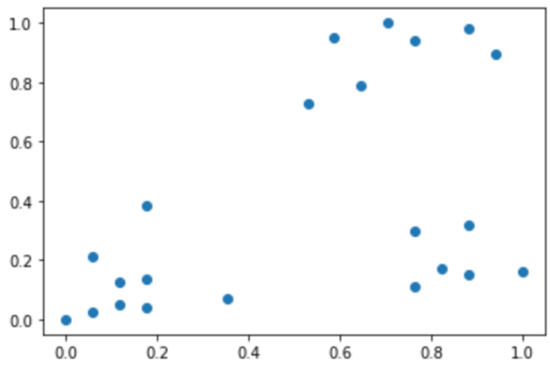
	Name	Age	Income($)
1	Name	Age	Income($)
2	Rob	27	70000
3	Michael	29	90000
4	Mohan	29	61000
5	Ismail	28	60000
6	Kory	42	150000
7	Gautam	39	155000
8	David	41	160000
9	Andrea	38	162000
10	Brad	36	156000

…

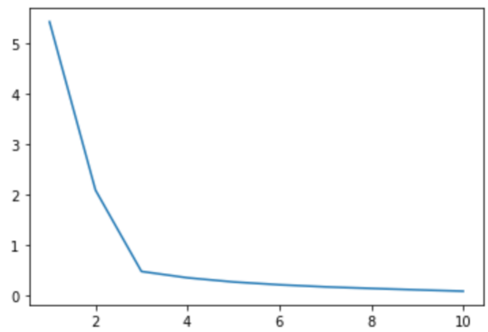
Output

	Name	Age	Income($)
0	Rob	0.058824	0.213675
1	Michael	0.176471	0.384615
2	Mohan	0.176471	0.136752
3	Ismail	0.117647	0.128205
4	Kory	0.941176	0.897436
5	Gautam	0.764706	0.940171
6	David	0.882353	0.982906
7	Andrea	0.705882	1.000000
8	Brad	0.588235	0.948718
9	Angelina	0.529412	0.726496

`<matplotlib.collections.PathCollection at 0x263bb51d3d0>`

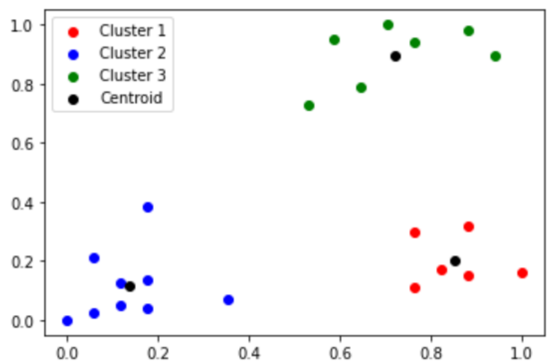


`[<matplotlib.lines.Line2D at 0x263bb67da60>]`



	Name	Age	Income($)	cluster
0	Rob	0.058824	0.213675	1
1	Michael	0.176471	0.384615	1
2	Mohan	0.176471	0.136752	1
3	Ismail	0.117647	0.128205	1
4	Kory	0.941176	0.897436	2

`<matplotlib.legend.Legend at 0x263bbb8fbb0>`



Program 8: Apply EM algorithm to cluster a set of data stored in a .CSV file.Compare the results of k-Means algorithm and EM algorithm.

Program

```
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("C:/Users/admin/Downloads/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 confusion matrix of K-Mean:\n',metrics.confusion_matrix(y, model.labels_))
print('The accuracy score of K-Mean: ',metrics.accuracy_score(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 confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
```

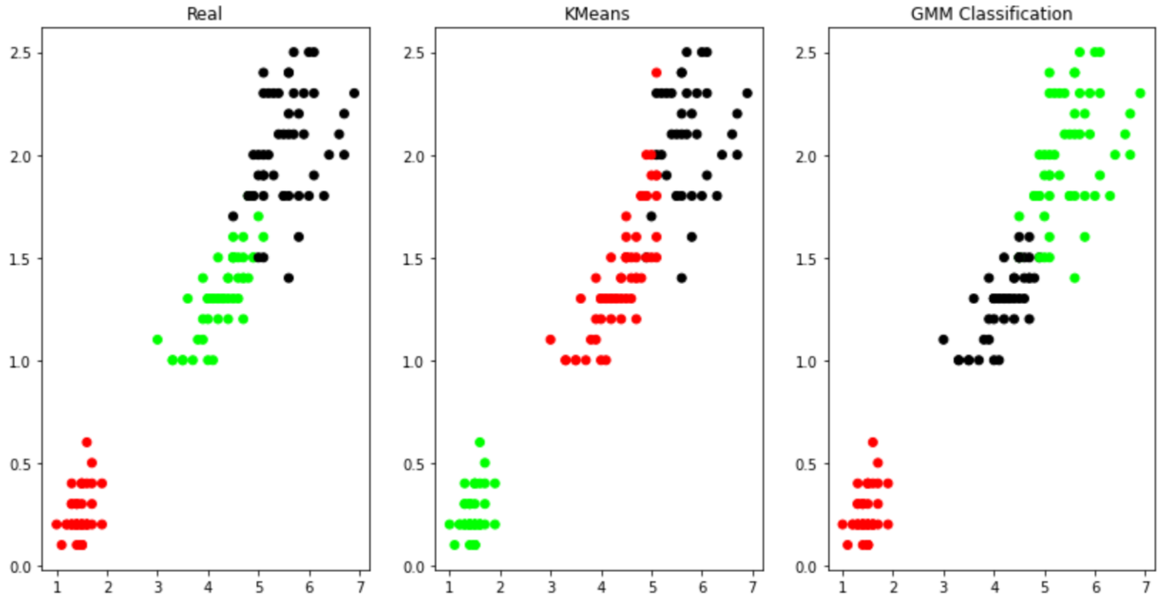
dataset.csv

1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa

…

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]]
```



Program 9: Write a program to implement k-NearestNeighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

Program

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

iris=datasets.load_iris()

x =  iris.data
y = iris.target

print(iris.feature_names)
print(iris.target_names)
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

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))
from sklearn.neighbors import RadiusNeighborsClassifier
rnc = RadiusNeighborsClassifier(radius = 5)
rnc.fit(x_train, y_train)

classes = {0:'setosa',1:'versicolor',2:'virginicia'}
x_new = [[1,1,1,1],[4,3,1.3,0.2]]
y_predict = rnc.predict(x_new)
print(classes[y_predict[0]])
print(classes[y_predict[1]])
```

Output

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
['setosa' 'versicolor' 'virginica']
Confusion Matrix
[[14  0  0]
 [ 0 18  1]
 [ 0  2 10]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.90      0.95      0.92        19
           2       0.91      0.83      0.87        12

    accuracy                           0.93        45
   macro avg       0.94      0.93      0.93        45
weighted avg       0.93      0.93      0.93        45

setosa
setosa
```

Program 10: Implement the non-parametricLocally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Program

```
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

```
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 :
 [-3.07038137 -2.97903806 -3.07809225 -2.93627863 -2.95209929 -3.03687263
 -2.8601589  -2.83440865 -2.98620123]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```