VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

COURSE TITLE

Submitted by

Lakshmi s kumar (1BM19CS078)

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" carried out by **Lakshmi s kumar(1BM19CS078),** 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** work prescribed for the said degree.

Name of the Lab-Incharge	**Dr. Jyothi S Nayak**
Designation	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

`

Index Sheet

LAB 1

FIND S ALGORITHM

```
import pandas as pd
import numpy as np

#to read the data in the csv file
print("USN:1BM19CS105")
data = pd.read_csv(r"C:\Users\admin\Downloads\data.csv")
print(data,"\n")

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

```
#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("The target is: ",target)

#training function to implement find-s algorithm
def train(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

#obtaining the final hypothesis
print("n The final hypothesis is:",train(d,target))
```

OUTPUT

```
USN:1BM19CS095
      Time Weather Temperature Company Humidity    Wind Goes
0  Morning   Sunny        Warm     Yes     Mild  Strong  Yes
1  Evening   Rainy        Cold      No     Mild  Normal   No
2  Morning   Sunny    Moderate     Yes   Normal  Normal  Yes
3  Evening   Sunny        Cold     Yes     High  Strong  Yes

The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]
The target is:  ['Yes' 'No' 'Yes' 'Yes']
n The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

WEEK 2

CANDIDATE ELIMINATION ALGORITHM

```
Import
numpy
as np
         import pandas as pd

         data = pd.read_csv(r'C:\Users\admin\Downloads\enjoysport.csv')
         concepts = np.array(data.iloc[:,0:-1])
         print("\nInstances are:\n",concepts)
         target = np.array(data.iloc[:,-1])
         print("\nTarget Values are: ",target)

         def learn(concepts, target):
             specific_h = concepts[0].copy()
             print("\nInitialization of specific_h and genearal_h")
             print("\nSpecific Boundary: ", specific_h)
             general_h = [["?" for i in range(len(specific_h))] for i in
         range(len(specific_h))]
             print("\nGeneric Boundary: ",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 Bundary after ", i+1, "Instance is ", specific_h)
                 print("Generic Boundary after ", i+1, "Instance is ", general_h)
                 print("\n")

             indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?',
         '?', '?']]
             for i in indices:
```

```python
            general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

OUTPUT

```
Instances are:
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Target Values are:  ['yes' 'yes' 'no' 'yes']

Initialization of specific_h and genearal_h

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

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

Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  1 Instance is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  2 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance 3 is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Negative
Specific Bundary after  3 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  3 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]



Instance 4 is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Instance is Positive
Specific Bundary after  4 Instance is  ['sunny' 'warm' '?' 'strong' '?' '?']
Generic Boundary after  4 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


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

LAB3

DECISION TREE USING ID3 ALGORITHM

```
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(r"C:\Users\admin\Downloads\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(r"C:\Users\admin\Downloads\id3_test.csv")

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

OUTPUT

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

LAB 4

LINEAR REGRESSION

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

dataset = pd.read_csv('salary_data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

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)
```

```python
# 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

LAB 5

NAÏVE BAYES NETWORK

```
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\Downloads\data5.csv")
col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness',
'insulin', 'bmi', 'diab_pred', 'age']
predicted_class = ['diabetes']


X = df[col_names].values
y = df[predicted_class].values


print(df.head)
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.4)

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

```
<bound method NDFrame.head of      num_preg  glucose_conc  diastolic_bp  thickness  insulin   bmi  \
0           6           148            72         35        0  33.6
1           1            85            66         29        0  26.6
2           8           183            64          0        0  23.3
3           1            89            66         23       94  28.1
4           0           137            40         35      168  43.1
..        ...           ...           ...        ...      ...   ...
763        10           101            76         48      180  32.9
764         2           122            70         27        0  36.8
765         5           121            72         23      112  26.2
766         1           126            60          0        0  30.1
767         1            93            70         31        0  30.4

     diab_pred  age  diabetes
0        0.627   50         1
1        0.351   31         0
2        0.672   32         1
3        0.167   21         0
4        2.288   33         1
..         ...  ...       ...
763      0.171   63         0
764      0.340   27         0
765      0.245   30         0
766      0.349   47         1
767      0.315   23         0

[768 rows x 9 columns]>

 the total number of Training Data : (460, 1)

 the total number of Test Data : (308, 1)
```

```
 Confusion matrix
[[176  29]
 [ 40  63]]

 Accuracy of the classifier is 0.775974025974026

 The value of Precision 0.6847826086956522

 The value of Recall 0.6116504854368932
Predicted Value for individual Test Data: [1]
```

LAB6

BAYESIAN NETWORK

```
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
cancer_model = BayesianModel([('Pollution', 'Cancer'),
                              ('Smoker', 'Cancer'),
                              ('Cancer', 'Xray'),
                              ('Cancer', 'Dyspnoea')])
print('Bayesian network nodes:')
print('\t', cancer_model.nodes())
print('Bayesian network edges:')
print('\t', cancer_model.edges())
cpd_poll = TabularCPD(variable='Pollution', variable_card=2,
                      values=[[0.9], [0.1]])
cpd_smoke = TabularCPD(variable='Smoker', variable_card=2,
                       values=[[0.3], [0.7]])
cpd_cancer = TabularCPD(variable='Cancer', variable_card=2,
                        values=[[0.03, 0.05, 0.001, 0.02],
                                [0.97, 0.95, 0.999, 0.98]],
                        evidence=['Smoker', 'Pollution'],
                        evidence_card=[2, 2])
cpd_xray = TabularCPD(variable='Xray', variable_card=2,
                      values=[[0.9, 0.2], [0.1, 0.8]],
                      evidence=['Cancer'], evidence_card=[2])
cpd_dysp = TabularCPD(variable='Dyspnoea', variable_card=2,
                      values=[[0.65, 0.3], [0.35, 0.7]],
                      evidence=['Cancer'], evidence_card=[2])
```

In [6]:

```
cancer_model.add_cpds(cpd_poll, cpd_smoke, cpd_cancer, cpd_xray, cpd_dysp)
print('Model generated bt adding conditional probability distribution(cpds)')

# Checking if the cpds are valid for the model.
print('Checking for Correctness of model:', end='')
print(cancer_model.check_model())


'''print('All local dependencies are as follows')
cancer_model.get_independencies()
'''

print('Displaying CPDs')
print(cancer_model.get_cpds('Pollution'))
print(cancer_model.get_cpds('Smoker'))
print(cancer_model.get_cpds('Cancer'))
print(cancer_model.get_cpds('Xray'))
print(cancer_model.get_cpds('Dyspnoea'))
```

```
cancer_infer = VariableElimination(cancer_model)
print('\nInferencing with Bayesian Network')

print('\nProbability of Cancer given Smoker')
q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1})
print(q)

print('\nProbability of Cancer given Smoker, Pollution')
q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker':
1,'Pollution': 1})
print(q)
```

```
Displaying CPDs
+--------------+-----+
| Pollution(0) | 0.9 |
+--------------+-----+
| Pollution(1) | 0.1 |
+--------------+-----+
+-----------+-----+
| Smoker(0) | 0.3 |
+-----------+-----+
| Smoker(1) | 0.7 |
+-----------+-----+
+-----------+--------------+--------------+--------------+--------------+
| Smoker    | Smoker(0)    | Smoker(0)    | Smoker(1)    | Smoker(1)    |
+-----------+--------------+--------------+--------------+--------------+
| Pollution | Pollution(0) | Pollution(1) | Pollution(0) | Pollution(1) |
+-----------+--------------+--------------+--------------+--------------+
| Cancer(0) | 0.03         | 0.05         | 0.001        | 0.02         |
+-----------+--------------+--------------+--------------+--------------+
| Cancer(1) | 0.97         | 0.95         | 0.999        | 0.98         |
+-----------+--------------+--------------+--------------+--------------+
+---------+-----------+-----------+
| Cancer  | Cancer(0) | Cancer(1) |
+---------+-----------+-----------+
| Xray(0) | 0.9       | 0.2       |
+---------+-----------+-----------+
| Xray(1) | 0.1       | 0.8       |
+---------+-----------+-----------+
+-------------+-----------+-----------+
| Cancer      | Cancer(0) | Cancer(1) |
+-------------+-----------+-----------+
| Dyspnoea(0) | 0.65      | 0.3       |
+-------------+-----------+-----------+
| Dyspnoea(1) | 0.35      | 0.7       |
+-------------+-----------+-----------+
```

```
Inferencing with Bayesian Network

Probability of Cancer given Smoker
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0029 |
+-----------+---------------+
| Cancer(1) |        0.9971 |
+-----------+---------------+

Probability of Cancer given Smoker, Pollution
0it [00:00, ?it/s]
0it [00:00, ?it/s]
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0200 |
+-----------+---------------+
| Cancer(1) |        0.9800 |
+-----------+---------------+
```

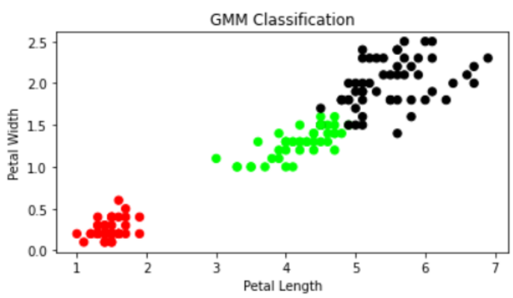
LAB7

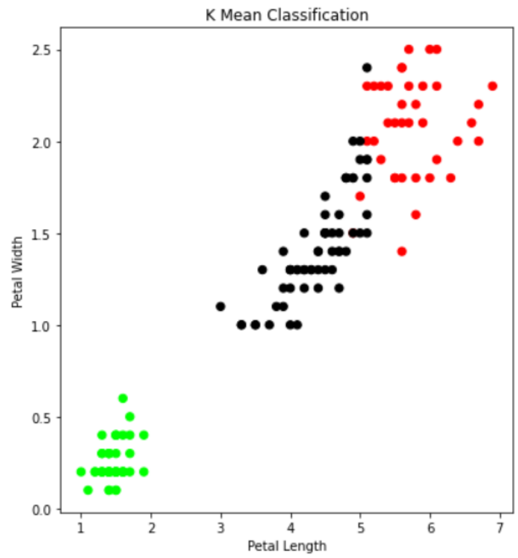
EM ALGORITHM

```
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))
```

```
The accuracy score of K-Mean:  0.09333333333333334
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [ 2  0 48]
 [36  0 14]]
The accuracy score of EM:  0.9666666666666667
The Confusion matrix of EM:  [[50  0  0]
 [ 0 45  5]
 [ 0  0 50]]
```





LAB8
K NEAREST NEIGHBOUR ALGORITHM

```
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

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))
```

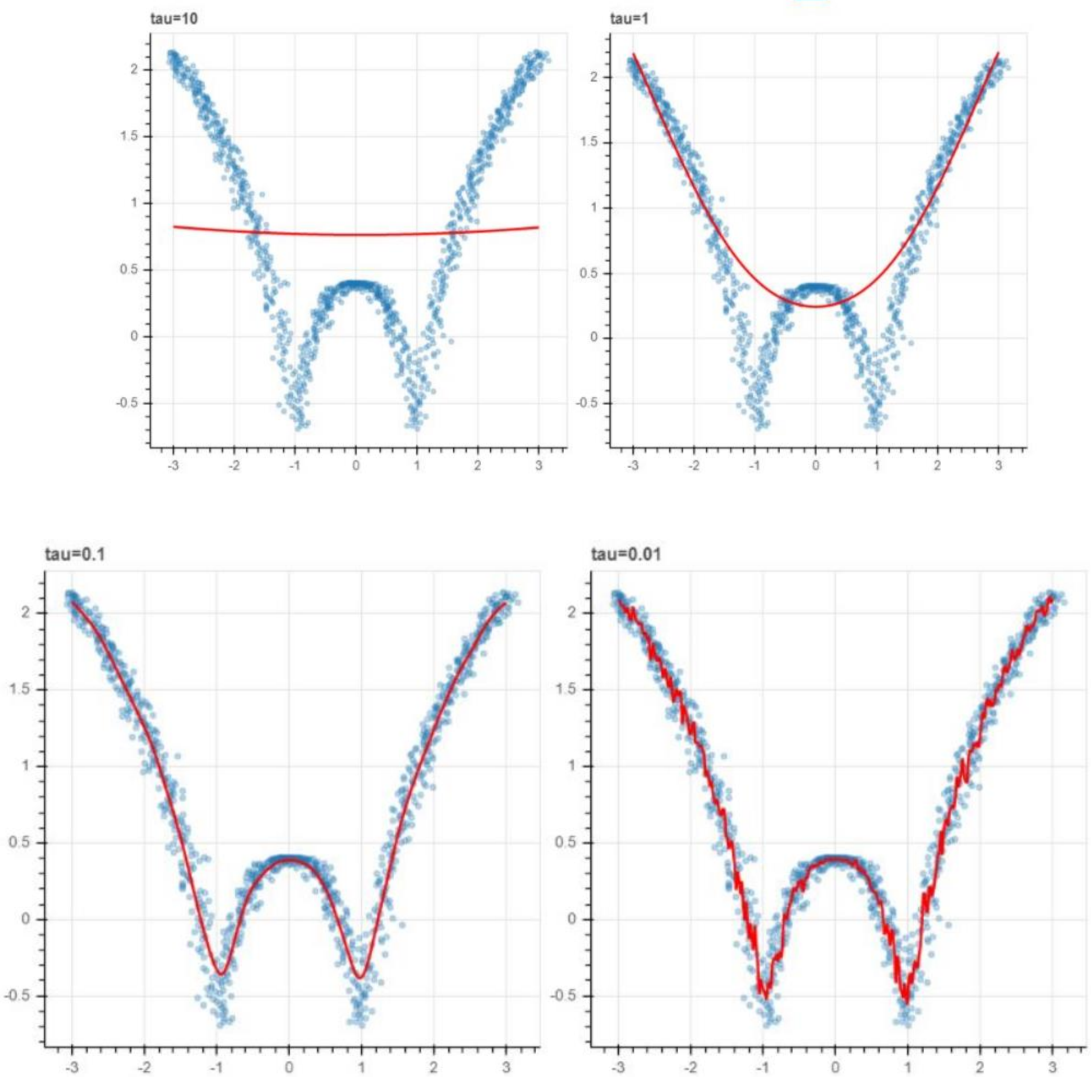
```
Confusion Matrix
[[14  0  0]
 [ 0 15  1]
 [ 0  1 14]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.94      0.94      0.94        16
           2       0.93      0.93      0.93        15

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```

LAB9
LOCALLY WEIGHTED REGRESSION

```
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 :
 [-3.04880282 -3.05548783 -2.9855117  -2.96076004 -2.97408012 -2.97887916
 -2.9302388  -3.05600455 -2.95179424]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```



LAB10

K MEANS ALGORITHM

```
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()
```

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

Means =  [[1.463, 0.258], [5.458, 1.962], [4.164, 1.285]]
```