

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT
on
MACHINE LEARNING

Submitted by

NITHIN C (1BM19CS106)

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 **NITHIN C (1BM19CS106)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Machine Learning - (20CS6PCMAL)** work prescribed for the said degree.

Dr G R Asha
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Find-S	
2	Candidate Elimination	
3	Decision Tree	
4	Naive Bayes	
5	Linear Regression	
6	Bayesian network	
7	k-means algorithm	
8	EM algorithm	
9	k-nearest neighbour algorithm	
10	Non-Parametric Locally Weighted Regression algorithm	

Course Outcome

--	--

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

```
In [1]: import numpy as np
import pandas as pd
```

```
In [2]: data = pd.read_csv("mydata.csv")
print(data, "\n")
```

	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

```
In [3]: d = np.array(data)[:, :-1]
print("\n The attributes are: ", d)
target = np.array(data)[:, -1]
print("\n The target is: ", target)
```

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']

```
In [4]: def finds(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis

print("\n The final hypothesis is:", finds(d,target))
```

The final hypothesis is: ['?', 'Sunny', '?', 'Yes', '?', '?']

```

In [1]: import numpy as np
import pandas as pd

In [2]: print("Enter features separated by space")
features = input().split()
print("Features ", features)
num_samples = int(input("enter number of samples: "))

Enter features separated by space
Time Weather Temperature Company Humidity Wind
Features ['Time', 'Weather', 'Temperature', 'Company', 'Humidity', 'Wind']
enter number of samples: 4

In [11]: def finds():
    specific_hypothesis = ["n"]*len(features)
    for a in range(num_samples):
        print("sample", a)

        temp_features = input("Enter features: ").split()
        target = input("Enter outcome: ")
        if target == "Yes":
            for x in range(len(specific_hypothesis)):
                if specific_hypothesis[x] == "n":
                    specific_hypothesis[x] = temp_features[x]
                elif temp_features[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
        print("Specific hypothesis: ", specific_hypothesis)
    return specific_hypothesis

In [12]: print("\n The final hypothesis is:",finds())

sample 0
Enter features: Morning Sunny Warm Yes Mild Strong
Enter outcome: Yes
Specific hypothesis: ['Morning', 'Sunny', 'Warm', 'Yes', 'Mild', 'Strong']
sample 1
Enter features: Evening Rainy Cold No Mild Normal
Enter outcome: No
Specific hypothesis: ['Morning', 'Sunny', 'Warm', 'Yes', 'Mild', 'Strong']
sample 2
Enter features: Morning Sunny Moderate Yes Normal Normal
Enter outcome: Yes
Specific hypothesis: ['Morning', 'Sunny', '?', 'Yes', '?', '?']
sample 3
Enter features: Evening Sunny Cold Yes High Strong
Enter outcome: Yes
Specific hypothesis: ['?', 'Sunny', '?', 'Yes', '?', '?']

The final hypothesis is: ['?', 'Sunny', '?', 'Yes', '?', '?']

```

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.


```
[ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?']]  
Instance 1 is ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']  
Instance is Positive  
Specific Boundary 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 Boundary 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 Boundary after 3 Instance is ['sunny' 'warm' '?' 'strong' 'warm' 'same']  
Generic Boundary after 3 Instance is [[ 'sunny', '?', '?', '?', '?', '?'], [ '?', 'warm', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?']]
```

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

```
In [ ]:
```

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.

```
In [1]: import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
data = load_iris()

In [2]: df = pd.DataFrame(data.data, columns = data.feature_names)

In [3]: df.head()

Out[3]:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0              5.1          3.5            1.4           0.2
1              4.9          3.0            1.4           0.2
2              4.7          3.2            1.3           0.2
3              4.6          3.1            1.5           0.2
4              5.0          3.6            1.4           0.2

In [4]: 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)

In [5]: x = df.drop(columns="Species")
```

```

In [5]: x = df.drop(columns="Species")
y = df["Species"]

In [6]: feature_names = x.columns
labels = y.unique()

In [7]: from sklearn.model_selection import train_test_split
X_train, test_x, y_train, test_lab = train_test_split(x,y,test_size = 0.4,random_state = 42)

In [10]: from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state = 42, criterion="entropy")

In [11]: clf.fit(X_train, y_train)

Out[11]: DecisionTreeClassifier(criterion='entropy', random_state=42)

In [12]: test_pred = clf.predict(test_x)

In [13]: from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
confusion_matrix = metrics.confusion_matrix(test_lab,test_pred)

In [14]: confusion_matrix

Out[14]: array([[23,  0,  0],
   [ 0, 19,  0],
   [ 0,  1, 17]], dtype=int64)

In [15]: matrix_df = pd.DataFrame(confusion_matrix)

In [14]: confusion_matrix

Out[14]: array([[23,  0,  0],
   [ 0, 19,  0],
   [ 0,  1, 17]], dtype=int64)

In [15]: matrix_df = pd.DataFrame(confusion_matrix)
ax = plt.axes()
sns.set(font_scale=1.3)
plt.figure(figsize=(10,7))
sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
ax.set_title('Confusion Matrix - Decision Tree')
ax.set_xlabel("Predicted label", fontsize=15)
ax.set_xticklabels([''+labels])
ax.set_ylabel("True Label", fontsize=15)
ax.set_yticklabels(list(labels), rotation = 0)
plt.show()




|            |            | setosa | versicolor | virginica |
|------------|------------|--------|------------|-----------|
| True Label | setosa     | 23     | 0          | 0         |
|            | versicolor | 0      | 19         | 0         |
|            | virginica  | 0      | 1          | 17        |

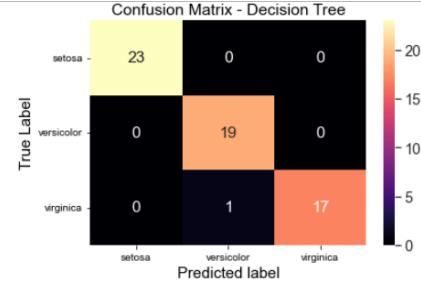


<Figure size 720x504 with 0 Axes>

In [16]: clf.score(test_x,test_lab)

Out[16]: 0.9833333333333333

```

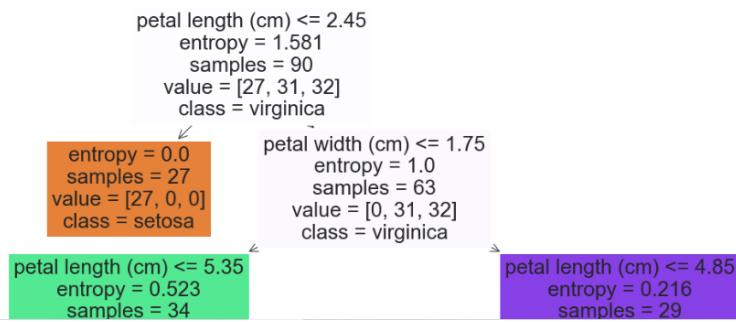


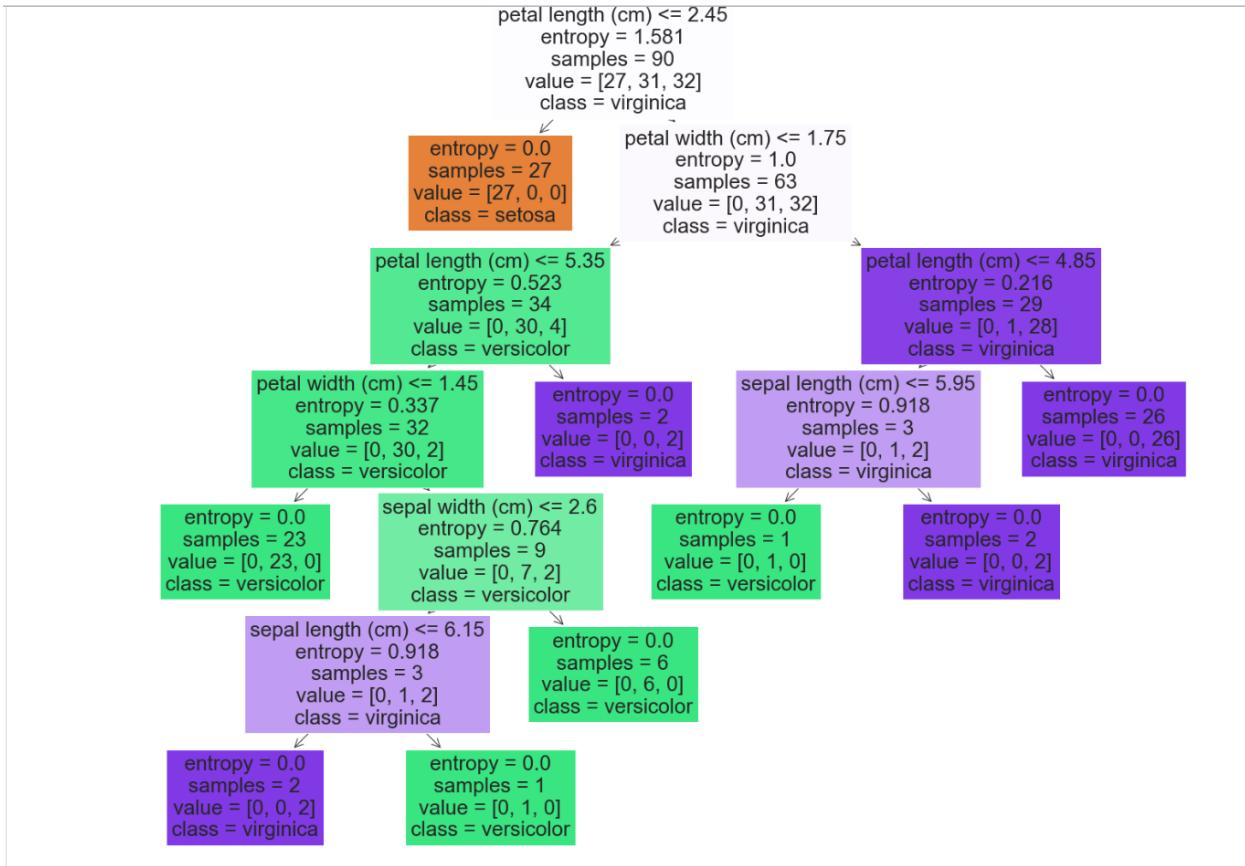
<Figure size 720x504 with 0 Axes>

```
In [16]: clf.score(test_x,test_lab)
```

```
Out[16]: 0.9833333333333333
```

```
In [17]: from sklearn import tree
fig = plt.figure(figsize=(25,20))
_=tree.plot_tree(clf,
                 feature_names=data.feature_names,
                 class_names=data.target_names,
                 filled=True)
```





```
In [1]: import pandas as pd
import math
import numpy as np
```

```
In [2]: data = pd.read_csv("dataset.csv")
features = [feat for feat in data]
features.remove("answer")
```

```
In [7]: features
```

```
Out[7]: ['outlook', 'temperature', 'humidity', 'wind']
```

```
In [4]: data
```

```
Out[4]: outlook temperature humidity wind answer
0 sunny hot high weak no
1 sunny hot high strong yes
2 overcast hot high weak yes
3 rain mild high weak yes
4 rain cool normal weak yes
5 rain cool normal strong no
6 overcast cool normal strong yes
7 sunny mild high weak no
8 sunny cool normal weak yes
9 rain mild normal weak yes
10 sunny mild normal strong yes
11 overcast mild high strong yes
12 overcast hot normal weak yes
```

```

11 overcast    mieu    malign    strong    yes
12 overcast    hot     normal    weak     yes
13 rain       mild     high     strong    no

```

```
In [3]:
class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
```

```
In [5]:
def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
```

```
In [6]:
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain
```

```
In [8]: def ID3(examples, attrs):
```

```
In [8]:
def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            newAttrs = attrs.copy()
            newAttrs.remove(max_feat)
            child = ID3(subdata, newAttrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root
```

```
In [9]: def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
```

```
        newAttrs = attrs.copy()
        newAttrs.remove(maxFeat)
        child = ID3(subData, newAttrs)
        dummyNode.children.append(child)
        root.children.append(dummyNode)
    return root
```

```
In [9]: def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)
```

```
In [10]: root = ID3(data, features)
printTree(root)

outlook
    overcast -> ['yes']

    rain
        wind
            strong -> ['no']
            weak -> ['yes']

    sunny
        humidity
            high -> ['no']
            normal -> ['yes']
```

```
In [ ]:
```

4. 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.

```
In [1]: import numpy as np
import pandas as pd

In [2]: data = pd.read_csv('/content/dataset.csv')
data.head()

Out[2]: PlayTennis Outlook Temperature Humidity Wind
0 No Sunny Hot High Weak
1 No Sunny Hot High Strong
2 Yes Overcast Hot High Weak
3 Yes Rain Mild High Weak
4 Yes Rain Cool Normal Weak
```

```
In [3]: y = list(data['PlayTennis'].values)
X = data.iloc[:,1:].values
print(f'Target Values: {y}')
print(f'Features: \n{X}')


Target Values: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
Features:
[['Sunny' 'Hot' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Strong']
 ['Overcast' 'Hot' 'High' 'Weak']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Cool' 'Normal' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Mild' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
```

```
In [4]: y_train = y[:8]
y_val = y[8:]
X_train = X[:8]
X_val = X[8:]
print("Number of instances in training set: " + str(len(X_train)))
print("Number of instances in testing set: " + str(len(X_val)))
```

Number of instances in training set:
Number of instances in testing set:

```
[5]: 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
```

```

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

```

In [6]:

```

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)

```

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

In [18]:

```

import numpy as np
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
y = df[predicted_class_names].values
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)

```

In [19]:

```

df.head()

```

Out[19]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [29]:

```

clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])

```

In [30]:

```

metrics.confusion_matrix(ytest,predicted)

```

Out[30]:

```

array([[139, 26],
       [33, 56]], dtype=int64)

```

In [28]:

```

print('\nConfusion matrix')
print(metrics.plot_confusion_matrix(clf,ytest,predicted))

```

```
In [30]: metrics.confusion_matrix(ytest,predicted)
Out[30]: array([[139,  26],
   [ 33,  56]], dtype=int64)

In [28]: print('\nConfusion matrix')
print(metrics.plot_confusion_matrix(clf,ytest,predicted))

Confusion matrix
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x00000190E55B3670>


```

```
In [31]: print(metrics.classification_report(ytest,predicted))

      precision    recall  f1-score   support

          0       0.81      0.84      0.82      165
          1       0.68      0.63      0.65      89

  accuracy                           0.77      254
  macro avg       0.75      0.74      0.74      254
weighted avg       0.76      0.77      0.77      254
```

```
In [8]: print("Predicted Value for individual Test Data:", predictTestData)
Predicted Value for individual Test Data: [1]
```

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

```

In [1]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

In [2]: dataset = pd.read_csv('salary_dataset.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

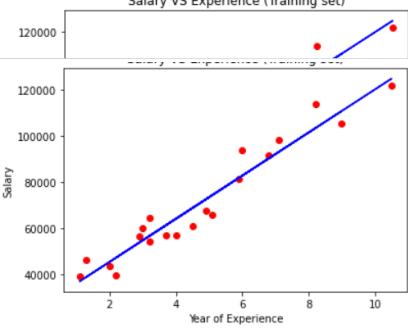
In [3]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)

In [4]: # Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

Out[4]: LinearRegression()

In [5]: # Predicting the Test set results
y_pred = regressor.predict(X_test)

In [6]: # 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()

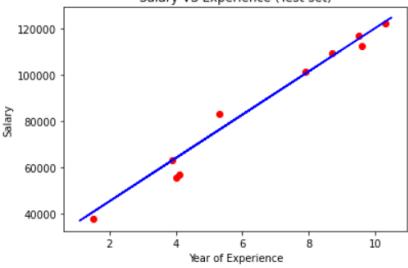


```



```

In [7]: # 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()



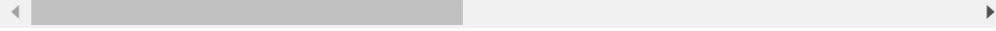
```

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

```

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

/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
    import pandas.util.testing as tm

 ▶

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)

The sample instances from the dataset are:
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope \
0   63    1    1      145    233     1      2      150      0       2.3      3
1   67    1    4      160    286     0      2      108      1       1.5      2
2   67    1    4      120    229     0      2      129      1       2.6      2
3   37    1    3      130    250     0      0      187      0       3.5      3
4   41    0    2      130    204     0      2      172      0       1.4      1

   ca  thal  heartdisease
0   0    6            0
1   3    3            2
2   2    7            1
3   0    3            0
4   0    3            0

Attributes and datatypes:
age          int64
sex          int64
cp           int64
trestbps    int64
chol         int64
fbs          int64
restecg     int64
thalach     int64
exang        int64
oldpeak     float64
slope        int64
ca           object
thal          object
heartdisease int64
dtype: object

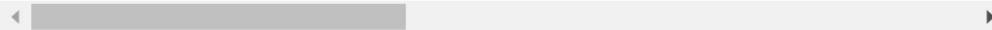
model = BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdiseas
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(trainingData,estimator=MaximumLikelihoodEstimator)
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

```

```
/usr/local/lib/python3.7/dist-packages/pgmpy/models/BayesianModel.py:10: FutureWarning:  
FutureWarning,
```

```
Learning CPD using Maximum likelihood estimators
```

```
Inferencing with Bayesian Network:
```



```
print('\n 1. Probability of HeartDisease given evidence = restecg (Rest ECG): 1')  
q1 = HeartDiseasetest_infer.query(variables = ['heartdisease'], evidence={'restecg':1})  
print(q1)
```

```
1. Probability of HeartDisease given evidence = restecg (Rest ECG): 1
```

```
Finding Elimination Order: : 4/4 [00:00<00:00,
```

```
100% 8.00it/s]
```

```
Eliminating: age: 100% 4/4 [00:00<00:00, 4.88it/s]
```

heartdisease	phi(heartdisease)
heartdisease(0)	0.1012
heartdisease(1)	0.0000
heartdisease(2)	0.2392
heartdisease(3)	0.2015
heartdisease(4)	0.4581

```
print('\n 2. Probability of HeartDisease given evidence = chol (Cholestorol): 100 ')  
q2 = HeartDiseasetest_infer.query(variables = ['heartdisease'], evidence={'chol':100})  
print(q2)
```

```
2. Probability of HeartDisease given evidence = chol (Cholestorol): 100
/usr/local/lib/python3.7/dist-packages/pgmpy/factors/discrete/DiscreteFactor.py:537:
UserWarning,
Finding Elimination Order: : 4/4 [00:00<00:00,
100% 7.39it/s]

Eliminating: age: 100% 4/4 [00:00<00:00, 3.70it/s]

+-----+
| heartdisease | phi(heartdisease) |
+=====+
| heartdisease(0) | 1.0000 |
+-----+
| heartdisease(1) | 0.0000 |
+-----+
| heartdisease(2) | 0.0000 |
+-----+
| heartdisease(3) | 0.0000 |
+-----+
```

7. Apply k-Means algorithm to cluster a set of data stored in a .CSV file. (using built in)

▼ K-Means Clustering

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline

from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.
```

```
df = pd.read_csv('/content/drive/MyDrive/income.csv')
df.head(10)
```

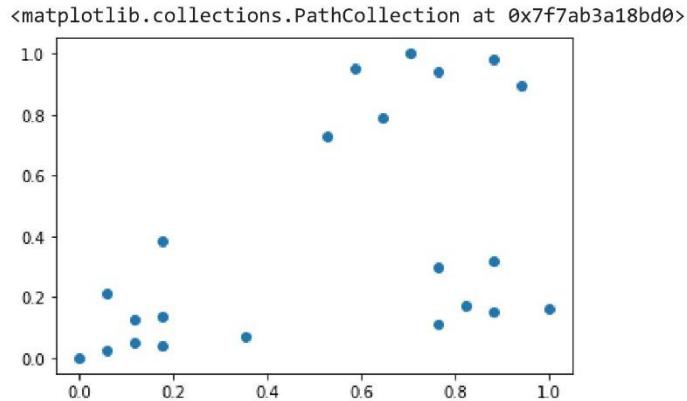
	Name	Age	Income(\$)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000
5	Gautam	39	155000
6	David	41	160000
7	Andrea	38	162000
8	Brad	36	156000
9	Angelina	35	130000

```
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)
```

	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	0.705992	1.000000

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



▼ Finding Elbow Point

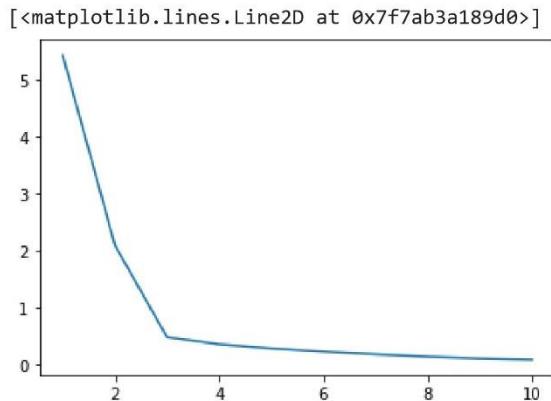
```
k_range = range(1, 11)
sse = []
for k in k_range:
    kmc = KMeans(n_clusters=k)
    kmc.fit(df[['Age', 'Income($)']])
    sse.append(kmc.inertia_)

sse
[5.434011511988178,
 2.091136388699078,
 0.4750783498553096,
 0.3491047094419566,
 0.27768187154369994,
 0.22020960864009398,
 0.1735559655531264,
 0.1327661931978319,
 0.10188787724979426,
 0.08026197041664467]
```

7/10/22, 11:03 PM

K-MeansClustering(using libraries).ipynb - Colaboratory

```
plt.xlabel = 'Number of Clusters'  
plt.ylabel = 'Sum of Squared Errors'  
plt.plot(k_range, sse)
```



- ▼ Therefore, the elbow point is 3

```
km = KMeans(n_clusters=3)  
km  
  
KMeans(n_clusters=3)  
  
y_predict = km.fit_predict(df[['Age', 'Income($)']])  
y_predict  
  
array([1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2],  
      dtype=int32)
```

```
df['cluster'] = y_predict  
df.head()
```

	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	0

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

	Name	Age	Income(\$)	cluster
4	Kory	0.941176	0.897436	0
5	Gautam	0.764706	0.940171	0
6	David	0.882353	0.982906	0
7	Andrea	0.705882	1.000000	0
8	Brad	0.588235	0.948718	0
9	Angelina	0.529412	0.726496	0

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

	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
11	Tom	0.000000	0.000000	1
12	Arnold	0.058824	0.025641	1
13	Jared	0.117647	0.051282	1
14	Stark	0.176471	0.038462	1
15	Ranbir	0.352941	0.068376	1

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

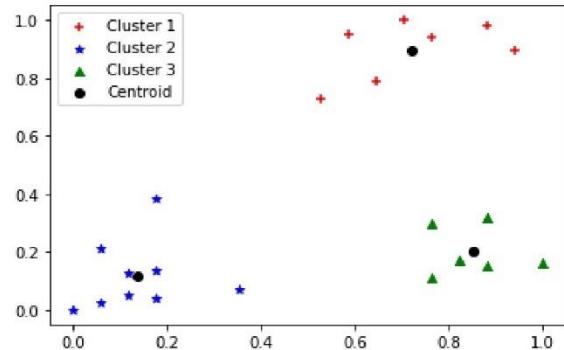
	Name	Age	Income(\$)	cluster
16	Dipika	0.823529	0.170940	2
17	Priyanka	0.882353	0.153846	2
18	Nick	1.000000	0.162393	2
19	Alia	0.764706	0.299145	2
20	Sid	0.882353	0.316239	2
21	Abdul	0.764706	0.111111	2

km.cluster_centers_

```
array([[0.72268908, 0.8974359 ],
       [0.1372549 , 0.11633428],
       [0.85294118, 0.2022792 ]])
```

```
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.legend((p1, p2, p3, c),
           ('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroid'))
```

```
<matplotlib.legend.Legend at 0x7f7aad7cf610>
```



```
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):
    https://colab.research.google.com/github/niths1271/MACHINE-LEARNING-LAB/blob/main/LAB 6/K_MeansClustering_hardcoded.ipynb#editMod... 1/5
```

7/10/22, 11:04 PM K-MeansClustering_hardcoded.ipynb - Colaboratory

```
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)
```

7/10/22, 11:04 PM K-MeansClustering_hardcoded.ipynb - Colaboratory

```

        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)

```

```

7/10/22, 11:04 PM K-MeansClustering_hardcoded.ipynb - Colaboratory

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()

[[6.3, 1.8], [5.1, 1.9], [1.9, 0.4], [1.6, 0.4], [4.9, 1.8], [1.7, 0.4], [4.2, 1.2],
Means = [[5.583, 2.032], [1.462, 0.257], [4.258, 1.341]]

```

The figure is a scatter plot with two axes ranging from 1 to 7. It shows data points clustered into three distinct groups: a green cluster at low x-values (1-2), a cyan cluster in the middle (x=3-5), and a blue cluster at higher x-values (x=5-7). Three red dots represent the cluster centers (means). A horizontal scrollbar is visible below the plot area.

https://colab.research.google.com/github/niths1271/MACHINE-LEARNING-LAB/blob/main/LAB%206/K_MeansClustering_hardcoded.ipynb#printMod... 4/5

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.

```
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)
#v cluster gmm
```

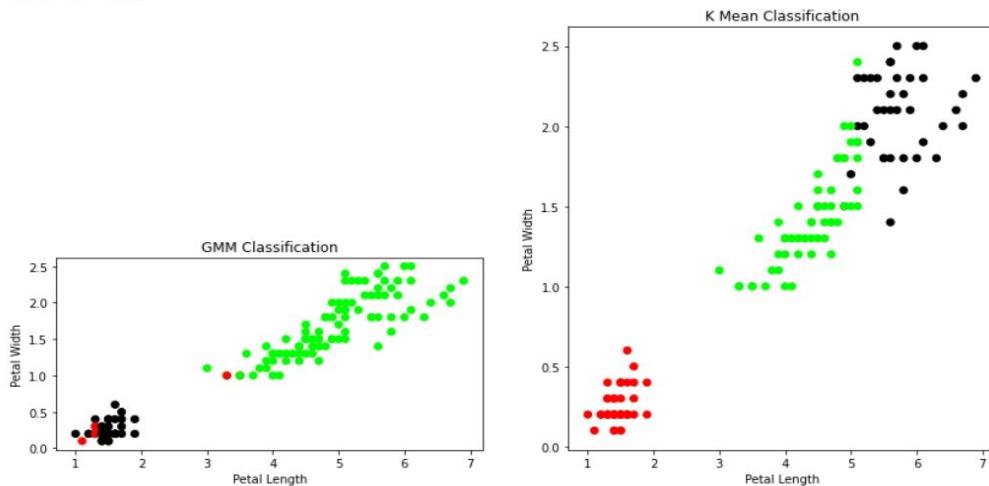
```

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-Means: 0.8933333333333333
 The Confusion matrix of K-Means: [[50 0 0]
 [0 48 2]
 [0 14 36]]
 The accuracy score of EM: 0.3533333333333333
 The Confusion matrix of EM: [[5 0 45]
 [2 48 0]
 [0 50 0]]



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

```

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

iris=datasets.load_iris()

x = iris.data
y = iris.target

# print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
# # print(x)
# print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
# # print(y)

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

#To Training the model and Nearest neighbors 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	
[[13 0 0]	
[0 13 2]	
[0 1 16]]	

Accuracy Metrics				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	13
1	0.93	0.87	0.90	15
2	0.89	0.94	0.91	17
accuracy			0.93	45
macro avg	0.94	0.94	0.94	45
weighted avg	0.93	0.93	0.93	45

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

(using built in)

```

import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook

def local_regression(x0, X, Y, tau):# add bias term
    x0 = np.r_[1, x0] # Add one to avoid the loss in information
    X = np.c_[np.ones(len(X)), X]

    # fit model: normal equations with kernel
    xw = X.T * radial_kernel(x0, X, tau) # XTranspose * W

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

    # predict value
    return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction

def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
# Weight or Radial Kernel 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 :

7/10/22, 11:23 PM

Copy of LocallyWeightedRegression.ipynb - Colaboratory

```
[-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]
```



```
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

# load data points
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)
```

7/10/22, 11:25 PM

WeightedRegression.ipynb - Colaboratory

```
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()
```

