VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning

Submitted by

Neha Cathrin A (1BM19CS099)

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 **Neha Cathrin (1BM19CS099),** 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 Head
Department of CSE
BMSCE, Bengaluru

`

Index Sheet

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

In [7]:

```
import pandas as pd
import numpy as np
data = pd.read_csv("data.csv")
print(data)
d = np.array(data)[:,:-1]
target = np.array(data)[:,-1]
def train(c,t):
    for i, val in enumerate(t):
        if val == "yes":
            sh = c[i].copy()
            break;
    for i, val in enumerate(c):
        if t[i] == "yes":
            for x in range(len(sh)):
                if val[x] != sh[x]:
                    sh[x] = "?"
                else:
                    pass
    return sh

print(train(d,target))
```

```
      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
['?' 'sunny' '?' 'yes' '?' '?']
```

In [ ]:

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.

In [1]:

```
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:
        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")
```

```
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', '?', '?', '?', '?']]
```

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

In [2]:

```
class Node:
    def __init__(self,attribute):
        self.attribute=attribute
        self.children=[]
        self.answer=""
```

In [3]:

```
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
```

In [4]:

```
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
```

In [6]:

```
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
```

In [7]:

```
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
```

In [8]:

```
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)
```

In [9]:

```
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)
```

In [10]:

```
dataset,features=load_csv("data.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("test.csv")

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

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

4.Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data samples

In [1]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
```

In [2]:

```
df = pd.read_csv("input_data.csv")
col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred', 'age']
predicted_class = ['diabetes']
```

In [3]:

```
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)
```

```
<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)
```

In [4]:

```
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)
```

```
 Confusion matrix
[[160  41]
 [ 47  60]]

 Accuracy of the classifier is 0.7142857142857143

 The value of Precision 0.594059405940594

 The value of Recall 0.5607476635514018
Predicted Value for individual Test Data: [1]
```

```
[ ]:
```

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

```
!pip install pgmpy
```

```
Requirement already satisfied: pgmpy in c:\programdata\anaconda3\lib\site-packages (0.1.18)
Requirement already satisfied: pyparsing in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (3.0.4)
Requirement already satisfied: statsmodels in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (0.12.2)
Requirement already satisfied: scikit-learn in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (0.24.2)
Requirement already satisfied: torch in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.11.0)
Requirement already satisfied: tqdm in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (4.62.3)
Requirement already satisfied: joblib in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.1.0)
Requirement already satisfied: pandas in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.3.4)
Requirement already satisfied: networkx in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (2.6.3)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.20.3)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from pgmpy) (1.7.1)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\programdata\anaconda3\lib\site-packages (from pandas->pgmpy) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\programdata\anaconda3\lib\site-packages (from pandas->pgmpy) (2021.3)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-dateutil>=2.7.3->pandas->pgmpy) (1.16.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn->pgmpy) (2.2.0)
Requirement already satisfied: patsy>=0.5 in c:\programdata\anaconda3\lib\site-packages (from statsmodels->pgmpy) (0.5.2)
Requirement already satisfied: typing-extensions in c:\programdata\anaconda3\lib\site-packages (from torch->pgmpy) (3.10.0.2)
Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-packages (from tqdm->pgmpy) (0.4.4)
```

```
from pgmpy.models import BayesianModel
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
```

```
cancer_model = BayesianNetwork([('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())
```

```python
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])
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())
```

```
Model generated bt adding conditional probability distribution(cpds)
Checking for Correctness of model:True
```

```python
'''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'))
```

```
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       |
+-------------+-----------+-----------+
```

```python
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)
```

```
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 |
+-----------+---------------+
```

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

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

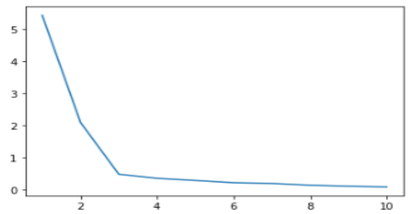
```
df = pd.read_csv('income.csv')
df.head(10)
```

	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)
```

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

```
[<matplotlib.lines.Line2D at 0x7f23cb541c10>]
```



Therefore, the elbow point is 3

```
km = KMeans(n_clusters=3)
km
```

```
KMeans(n_clusters=3)
```

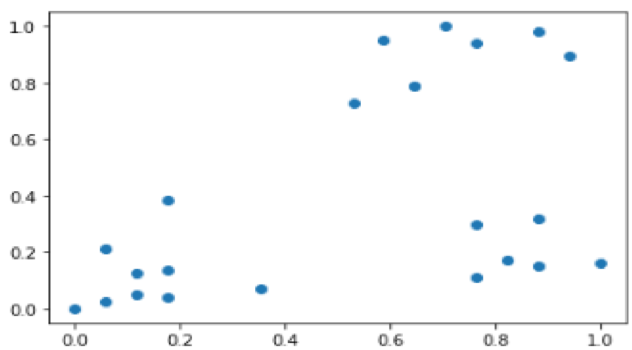
```
y_predict = km.fit_predict(df[['Age', 'Income($)']])
y_predict
```

```
array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2],
      dtype=int32)
```

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

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

```
<matplotlib.collections.PathCollection at 0x7f23ce044f10>
```



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
```

```
[5.434011511988179,
 2.091136388699078,
 0.4750783498553097,
 0.3491047094419566,
 0.2818479744366238,
 0.21055478995472496,
 0.18752738899206242,
 0.13265419827245162,
 0.10188787724979426,
 0.08026197041664467]
```

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

	Name	Age	Income($)	cluster
0	Rob	0.058824	0.213675	0
1	Michael	0.176471	0.384615	0
2	Mohan	0.176471	0.136752	0
3	Ismail	0.117647	0.128205	0
11	Tom	0.000000	0.000000	0
12	Arnold	0.058824	0.025641	0
13	Jared	0.117647	0.051282	0
14	Stark	0.176471	0.038462	0
15	Ranbir	0.352941	0.068376	0

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

	Name	Age	Income($)	cluster
4	Kory	0.941176	0.897436	1
5	Gautam	0.764706	0.940171	1
6	David	0.882353	0.982906	1
7	Andrea	0.705882	1.000000	1
8	Brad	0.588235	0.948718	1
9	Angelina	0.529412	0.726496	1
10	Donald	0.647059	0.786325	1

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

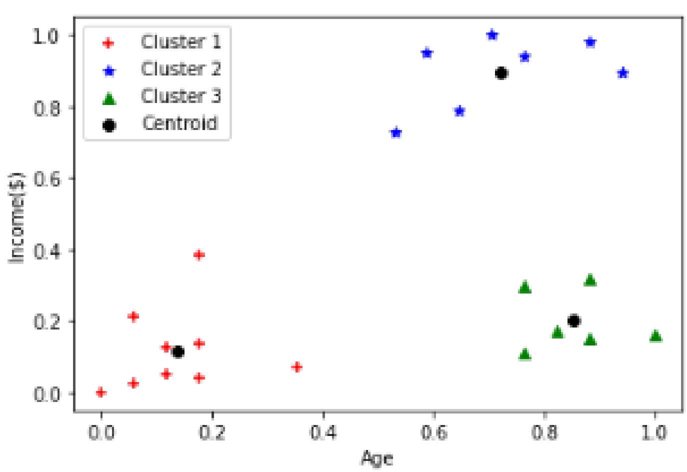
	Name	Age	Income($)	cluster
16	Dipika	0.823529	0.170940	2
17	Priyanka	0.882353	0.153846	2
18	Nick	1.000000	0.162393	2
19	Alia	0.764706	0.299145	2
20	Sid	0.882353	0.316239	2
21	Abdul	0.764706	0.111111	2

```
km.cluster_centers_
```

```
array([[0.1372549 , 0.11633428],
       [0.72268908, 0.8974359 ],
       [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($)'], marker='^', color='green')
c = plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], color='black')
plt.xlabel('Age')
plt.ylabel('Income($)')
plt.legend((p1, p2, p3, c),
           ('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroid'))
```

```
<matplotlib.legend.Legend at 0x7f23cb75d910>
```

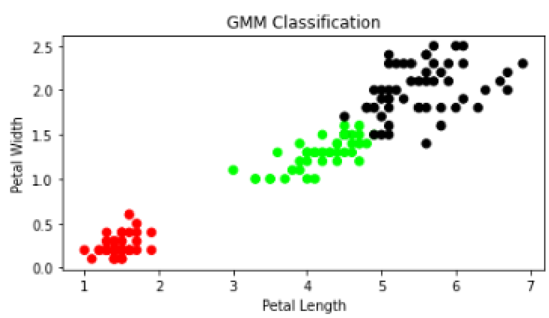


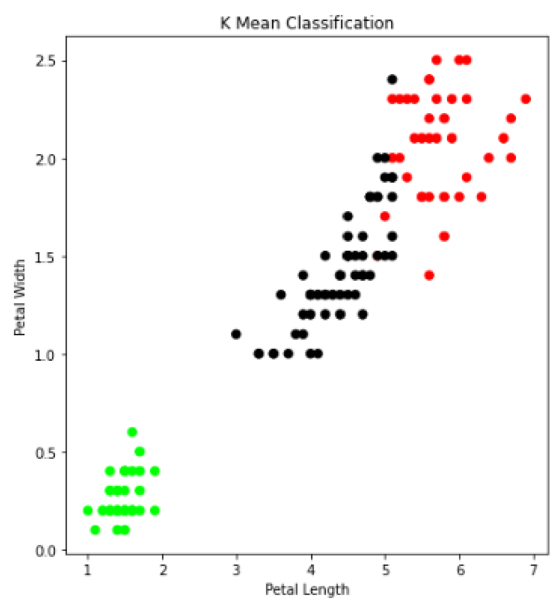
7.Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

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

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']

model = KMeans(n_clusters=3)
model.fit(X)


plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')


# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))
```

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm = gmm.predict(xs)
#y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))
```

```
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]]
```





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

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

iris=datasets.load_iris()

x = iris.data
y = iris.target

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

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

#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

#To make predictions on our test data
y_pred=classifier.predict(x_test)

print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

```
sepal-length sepal-width petal-length petal-width
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
```

```
class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Confusion Matrix
[[21  0  0]
 [ 0 10  1]
 [ 0  1 12]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       0.91      0.91      0.91        11
           2       0.92      0.92      0.92        13

    accuracy                           0.96        45
   macro avg       0.94      0.94      0.94        45
weighted avg       0.96      0.96      0.96        45
```

9.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_data.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()
```

In [ ]:

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.

```
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
```

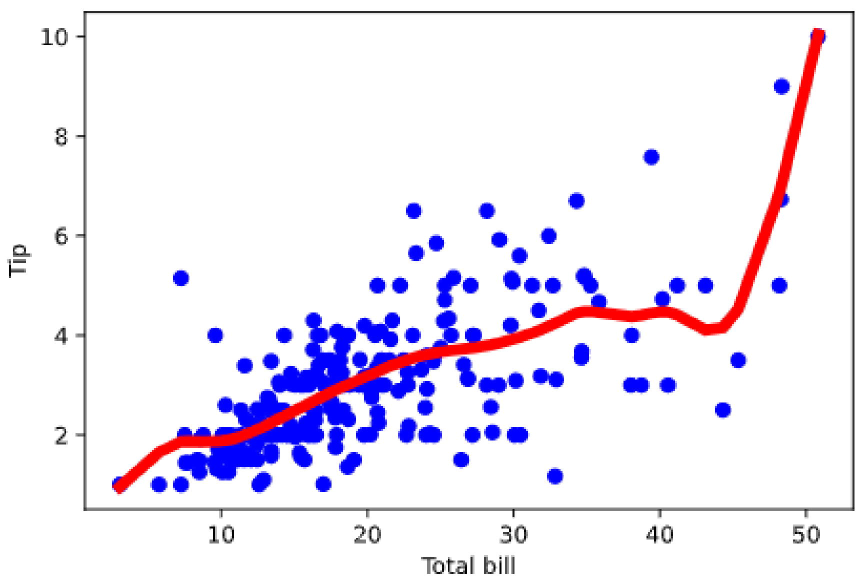
```
def localWeight(point,xmat,ymat,k):
 wei = kernel(point,xmat,k)
 W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
 return W
```

```
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('tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)
```

```
#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimensional array form
m= np1.shape(mbill)[1]
# print(m) 244 data is stored in m
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
#print(X)
#set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
```

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='blue')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
```



```python
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
```

```python
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
```

```python
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])
```

```
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 :
 [-2.95983905 -2.77699311 -3.06439147 -3.15903005 -3.19868861 -3.00406048
 -2.9445708  -2.87933746 -2.94253902]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```