VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING

Submitted by

ANITEJ PRASAD
(1BM19CS194)

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
April-2022 to August-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 **ANITEJ PRASAD (1BM19CS194),** 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.

Prof Saritha A.N.
Assistant professor
Department of CSE
BMSCE, Bengaluru

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

`

Index Sheet

Course Outcome

CO1	Ability to apply the different learning algorithms.
CO2	Ability to analyse the learning techniques for a given dataset.
CO3	Ability to design a model using machine learning to solve a problem.
CO4	Ability to conduct practical experiments to solve problems using appropriate machine learning techniques.

LAB1- FIND S ALGORITHM

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 csv

def findS(dataset, hypothesis):
    for i in range(len(dataset)):
        if dataset[i][-1] == 'yes':
            print('The tuple', i+1, 'is a positive instance.')
            for j in range(len(hypothesis)):
                if hypothesis[j] == '0'  or dataset[i][j] == hypothesis[j]:
                    hypothesis[j] = dataset[i][j]
                else:
                    hypothesis[j] = '?'
            print('The hypothesis for traning tuple',i+1,'and instance',j+1, 'is:', hypothesis)
        elif dataset[i][-1] == 'no':
            print('The tuple', i+1, 'is a negative instance.')
            print('The hypothesis for traning tuple',i+1, 'is:', hypothesis)
    return hypothesis



def main():

    dataset = []
    with open('FindS-CSV.csv', 'r') as csvfile:
        next(csvfile)
        for row in csv.reader(csvfile):
            dataset.append(row)
        print(dataset)
    hypothesis = ['0']*len(dataset[0])
    print('The Initial hypothesis:', hypothesis)
    hypothesis = findS(dataset, hypothesis)
    print('Final Hypothesis: ', hypothesis)

if __name__ == "__main__":
    main()
```

```
[['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'], ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'], ['rainy', 'cold', 'high', 's
trong', 'warm', 'change', 'no'], ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']]
The Initial hypothesis: ['0', '0', '0', '0', '0', '0', '0']
The tuple 1 is a positive instance.
The hypothesis for traning tuple 1 and instance 7 is: ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
The tuple 2 is a positive instance.
The hypothesis for traning tuple 2 and instance 7 is: ['sunny', 'warm', '?', 'strong', 'warm', 'same', 'yes']
The tuple 3 is a negative instance.
The hypothesis for traning tuple 3 is: ['sunny', 'warm', '?', 'strong', 'warm', 'same', 'yes']
The tuple 4 is a positive instance.
The hypothesis for traning tuple 4 and instance 7 is: ['sunny', 'warm', '?', 'strong', '?', '?', 'yes']
Final Hypothesis:  ['sunny', 'warm', '?', 'strong', '?', '?', 'yes']
```

In [ ]:

LAB2- Candidate Elimination

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 pandas as pd
import numpy as np
import csv


data = pd.read_csv('Candidate-Elimination.csv')
d = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",d)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
```

```
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']
```

In [2]:
```
def learn(d, target):
    specific_h = d[0].copy()
    print("\nSpecific Hypothesis: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Hypothesis: ",general_h)

    for i, h in enumerate(d):
        print("\nIteration", 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 Hypothesis after ", i+1, "Instance is ", specific_h)
        print("Generic Hypothesis 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
```

In [3]:

```
specific, general = learn(d, target)

print("Final Specific Hypothesis: ", '<', ', '.join(specific),'>')
print("Final General Hypothesis: ")
for i in general:
    print('<', ', '.join(i),'>, ')
```

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

Generic Hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

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



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



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



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

Final Specific Hypothesis:  < sunny, warm, ?, strong, ?, ? >
Final General Hypothesis:
< sunny, ?, ?, ?, ?, ? >,
< ?, warm, ?, ?, ?, ? >,
```

In [ ]:

LAB3- Decision Tree

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

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
```

In [4]:

```
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)
```

In [5]:

```
dataset,features=load_csv("id3.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("id3.csv")

for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:",end="   ")
    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
```

```
The test instance: ['sunny', 'hot', 'high', 'weak', 'no']
The label for test instance:   no
The test instance: ['sunny', 'hot', 'high', 'strong', 'no']
The label for test instance:   no
The test instance: ['overcast', 'hot', 'high', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'mild', 'high', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'cool', 'normal', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'cool', 'normal', 'strong', 'no']
The label for test instance:   no
The test instance: ['overcast', 'cool', 'normal', 'strong', 'yes']
The label for test instance:   yes
The test instance: ['sunny', 'mild', 'high', 'weak', 'no']
The label for test instance:   no
The test instance: ['sunny', 'cool', 'normal', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'mild', 'normal', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['sunny', 'mild', 'normal', 'strong', 'yes']
The label for test instance:   yes
The test instance: ['overcast', 'mild', 'high', 'strong', 'yes']
The label for test instance:   yes
The test instance: ['overcast', 'hot', 'normal', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'mild', 'high', 'strong', 'no']
The label for test instance:   no
```

In [ ]:

LAB4- Naïve Bayes Classifier

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]:
```python
import numpy as np
import pandas as pd
```

In [2]:
```python
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]:
```python
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', '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' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
```

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

```
Number of instances in training set: 8
Number of instances in testing set: 6
```

In [5]:

```python
class NaiveBayesClassifier:
    def __init__(self, X, y):
        self.X, self.y = X, y
        self.N = len(self.X)
        self.dim = len(self.X[0])
        self.attrs = [[] for _ in range(self.dim)]
        self.output_dom = {}
        self.data = []
        for i in range(len(self.X)):
            for j in range(self.dim):
                if not self.X[i][j] in self.attrs[j]:
                    self.attrs[j].append(self.X[i][j])
            if not self.y[i] in self.output_dom.keys():
                self.output_dom[self.y[i]] = 1
            else:
                self.output_dom[self.y[i]] += 1
            self.data.append([self.X[i], self.y[i]])
    def classify(self, entry):
        solve = None
        max_arg = -1
        for y in self.output_dom.keys():
            prob = self.output_dom[y]/self.N
            for i in range(self.dim):
                cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y]
                n = len(cases)
                prob *= n/self.N
            if prob > max_arg:
                max_arg = prob
                solve = y
        return solve
```

In [6]:

```python
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
```

LAB5- Bayesian Network

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

In [1]:

```
!pip install bayespy
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting bayespy
  Downloading bayespy-0.5.22.tar.gz (490 kB)
Requirement already satisfied: numpy>=1.10.0 in c:\programdata\anaconda3\lib\site-packages (from bayespy) (1.21.5)
Requirement already satisfied: scipy>=0.13.0 in c:\programdata\anaconda3\lib\site-packages (from bayespy) (1.7.3)
Requirement already satisfied: h5py in c:\programdata\anaconda3\lib\site-packages (from bayespy) (3.6.0)
Building wheels for collected packages: bayespy
  Building wheel for bayespy (setup.py): started
  Building wheel for bayespy (setup.py): finished with status 'done'
  Created wheel for bayespy: filename=bayespy-0.5.22-py3-none-any.whl size=379454 sha256=5e83889d5cd79371d5456950bc6e50be36b085b60b7c4a71b4e5e1fe991698
53
  Stored in directory: c:\users\admin\appdata\local\pip\cache\wheels\71\1f\01\0bf4461db21a3ce88a441a08de5f3618151f25bdf85c297753
Successfully built bayespy
Installing collected packages: bayespy
Successfully installed bayespy-0.5.22
```

In [2]:

```
import bayespy as bp
import numpy as np
import csv
!pip3 install colorama
!pip3 install colorama
from colorama import init
from colorama import Fore, Back, Style
init()

# Define Parameter Enum values
# Age
ageEnum = {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1,
           'MiddleAged': 2, 'Youth': 3, 'Teen': 4}
# Gender
genderEnum = {'Male': 0, 'Female': 1}
# FamilyHistory
familyHistoryEnum = {'Yes': 0, 'No': 1}
# Diet(Calorie Intake)
dietEnum = {'High': 0, 'Medium': 1, 'Low': 2}
# LifeStyle
lifeStyleEnum = {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}
# Cholesterol
cholesterolEnum = {'High': 0, 'BorderLine': 1, 'Normal': 2}
# HeartDisease
heartDiseaseEnum = {'Yes': 0, 'No': 1}
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-packages (0.4.4)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-packages (0.4.4)
```

In [9]:

```
import pandas as pd
```

In [10]:

```
data = pd.read_csv(r"C:\Users\Admin\OneDrive\Desktop\6th sem\ML\lab-ml\Lab 6\Bayesian network- heart_disease\heart_disease_data.csv")
```

In [11]:

```
data =np.array(data, dtype='int8')
N = len(data)
```

In [12]:

```
# Input data column assignment
p_age = bp.nodes.Dirichlet(1.0*np.ones(5))
age = bp.nodes.Categorical(p_age, plates=(N,))
age.observe(data[:, 0])

p_gender = bp.nodes.Dirichlet(1.0*np.ones(2))
gender = bp.nodes.Categorical(p_gender, plates=(N,))
gender.observe(data[:, 1])

p_familyhistory = bp.nodes.Dirichlet(1.0*np.ones(2))
familyhistory = bp.nodes.Categorical(p_familyhistory, plates=(N,))
familyhistory.observe(data[:, 2])

p_diet = bp.nodes.Dirichlet(1.0*np.ones(3))
diet = bp.nodes.Categorical(p_diet, plates=(N,))
diet.observe(data[:, 3])

p_lifestyle = bp.nodes.Dirichlet(1.0*np.ones(4))
lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N,))
lifestyle.observe(data[:, 4])

p_cholesterol = bp.nodes.Dirichlet(1.0*np.ones(3))
cholesterol = bp.nodes.Categorical(p_cholesterol, plates=(N,))
cholesterol.observe(data[:, 5])
```

In [13]:

```
# Prepare nodes and establish edges
# np.ones(2) -> HeartDisease has 2 options Yes/No
# plates(5, 2, 2, 3, 4, 3) -> corresponds to options present for domain values
p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))
heartdisease = bp.nodes.MultiMixture(
    [age, gender, familyhistory, diet, lifestyle, cholesterol], bp.nodes.Categorical, p_heartdisease)
heartdisease.observe(data[:, 6])
p_heartdisease.update()
```

In [ ]:

```
#print("Sample Probability")
#print("Probability(HeartDisease|Age=SuperSeniorCitizen, Gender=Female, FamilyHistory=Yes, DietIntake=Medium, LifeStyle=Sedetary, Cholesterol=High)")
#print(bp.nodes.MultiMixture([ageEnum['SuperSeniorCitizen'], genderEnum['Female'], familyHistoryEnum['Yes'], dietEnum['Medium'], lifeStyleEnum['Sedeta

# Interactive Test
m = 0
while m == 0:
    print("\n")
    res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))), int(input('Enter Gender: ' + str(genderEnum))), int(input('Enter FamilyHist
        dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter Cholesterol: ' + str(cholesterolEnum)))], bp.nodes.Categor
    print("Probability(HeartDisease) = " + str(res))

# print(Style.RESET_ALL)
    m = int(input("Enter for Continue:0, Exit :1 "))
```

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

In [ ]:

LAB6- K-Means Clustering

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

In [2]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
from itertools import cycle, islice
```

In [3]:
```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

In [4]:
```
data = pd.read_csv('/content/drive/MyDrive/minute_weather.csv')
```

In [5]:
```
data.shape
```

Out[5]:
```
(1587257, 13)
```

In [6]:
```
data.head()
```

Out[6]:

	rowID	hpwren_timestamp	air_pressure	air_temp	avg_wind_direction	avg_wind_speed	max_wind_direction	max_wind_speed	min_wind_direction	min_wind_speed	rain_accumul
0	0	2011-09-10 00:00:49	912.3	64.76	97.0	1.2	106.0	1.6	85.0	1.0	
1	1	2011-09-10 00:01:49	912.3	63.86	161.0	0.8	215.0	1.5	43.0	0.2	
2	2	2011-09-10 00:02:49	912.3	64.22	77.0	0.7	143.0	1.2	324.0	0.3	
3	3	2011-09-10 00:03:49	912.3	64.40	89.0	1.2	112.0	1.6	12.0	0.7	
4	4	2011-09-10 00:04:49	912.3	64.40	185.0	0.4	260.0	1.0	100.0	0.1	

In [7]:
```
#data sampling
sampled_df = data[(data['rowID'] % 10) == 0]
sampled_df.shape
```

In [7]:

```
#data sampling
sampled_df = data[(data['rowID'] % 10) == 0]
sampled_df.shape
```

Out[7]:

```
(158726, 13)
```

In [8]:

```
sampled_df.describe().transpose()
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
rowID	158726.0	793625.000000	458203.937509	0.00	396812.5	793625.00	1190437.50	1587250.00
air_pressure	158726.0	916.830161	3.051717	905.00	914.8	916.70	918.70	929.50
air_temp	158726.0	61.851589	11.833569	31.64	52.7	62.24	70.88	99.50
avg_wind_direction	158680.0	162.156100	95.278201	0.00	62.0	182.00	217.00	359.00
avg_wind_speed	158680.0	2.775215	2.057624	0.00	1.3	2.20	3.80	31.90
max_wind_direction	158680.0	163.462144	92.452139	0.00	68.0	187.00	223.00	359.00
max_wind_speed	158680.0	3.400558	2.418802	0.10	1.6	2.70	4.60	36.00
min_wind_direction	158680.0	166.774017	97.441109	0.00	76.0	180.00	212.00	359.00
min_wind_speed	158680.0	2.134664	1.742113	0.00	0.8	1.60	3.00	31.60
rain_accumulation	158725.0	0.000318	0.011236	0.00	0.0	0.00	0.00	3.12
rain_duration	158725.0	0.409627	8.665523	0.00	0.0	0.00	0.00	2960.00
relative_humidity	158726.0	47.609470	26.214409	0.90	24.7	44.70	68.00	93.00

In [9]:

```
sampled_df[sampled_df['rain_accumulation'] == 0].shape
```

Out[9]:

```
(157812, 13)
```

In [10]:

```
sampled_df[sampled_df['rain_duration'] == 0].shape
```

Out[10]:

```
(157237, 13)
```

In [11]:

```
del sampled_df['rain_accumulation']
del sampled_df['rain_duration']
```

In [12]:

```
rows_before = sampled_df.shape[0]
sampled_df = sampled_df.dropna()
rows_after = sampled_df.shape[0]
```

In [13]:

```
rows_before - rows_after
```

Out[13]:

```
46
```

In [14]:

```
sampled_df.columns
```

Out[14]:

```
Index(['rowID', 'hpwren_timestamp', 'air_pressure', 'air_temp',
       'avg_wind_direction', 'avg_wind_speed', 'max_wind_direction',
       'max_wind_speed', 'min_wind_direction', 'min_wind_speed',
       'relative_humidity'],
      dtype='object')
```

In [15]:

```
features = ['air_pressure', 'air_temp', 'avg_wind_direction', 'avg_wind_speed', 'max_wind_direction', 'max_wind_speed','relative_humidity']
```

In [16]:

```
select_df = sampled_df[features]
```

In [17]:

```
select_df.columns
```

Out[17]:

```
Index(['air_pressure', 'air_temp', 'avg_wind_direction', 'avg_wind_speed',
       'max_wind_direction', 'max_wind_speed', 'relative_humidity'],
      dtype='object')
```

In [18]:

```
select_df
```

Out[18]:

	air_pressure	air_temp	avg_wind_direction	avg_wind_speed	max_wind_direction	max_wind_speed	relative_humidity
0	912.3	64.76	97.0	1.2	106.0	1.6	60.5
10	912.3	62.24	144.0	1.2	167.0	1.8	38.5
20	912.2	63.32	100.0	2.0	122.0	2.5	58.3
30	912.2	62.60	91.0	2.0	103.0	2.4	57.9
40	912.2	64.04	81.0	2.6	88.0	2.9	57.4
...	...	...	...	...	...	...	...
1587210	915.9	75.56	330.0	1.0	341.0	1.3	47.8
1587220	915.9	75.56	330.0	1.1	341.0	1.4	48.0
1587230	915.9	75.56	344.0	1.4	352.0	1.7	48.0
1587240	915.9	75.20	359.0	1.3	9.0	1.6	46.3
1587250	915.9	74.84	6.0	1.5	20.0	1.9	46.1

158680 rows × 7 columns

In [19]:

```
X = StandardScaler().fit_transform(select_df)
X
```

Out[19]:

```
array([[-1.48456281,  0.24544455, -0.68385323, ..., -0.62153592,
        -0.74440309,  0.49233835],
       [-1.48456281,  0.03247142, -0.19055941, ...,  0.03826701,
        -0.66171726, -0.34710804],
       [-1.51733167,  0.12374562, -0.65236639, ..., -0.44847286,
        -0.37231683,  0.40839371],
       ...,
       [-0.30488381,  1.15818654,  1.90856325, ...,  2.0393087 ,
        -0.70306017,  0.01538018],
       [-0.30488381,  1.12776181,  2.06599745, ..., -1.67073075,
        -0.74440309, -0.04948614],
       [-0.30488381,  1.09733708, -1.63895404, ..., -1.55174989,
        -0.62037434, -0.05711747]])
```

In [20]:

```
#Using kmeans clustering
kmeans = KMeans(n_clusters=12)
model = kmeans.fit(X)
print("model\n", model)
```

```
model
 KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=12, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [21]:

```
centers = model.cluster_centers_
centers
```

Out[21]:

```
array([[ 0.06360158, -0.79106984, -1.19865111, -0.57036444, -1.04474114,
        -0.58494759,  0.88013875],
       [-0.7245782 ,  0.51194369,  0.17191124, -0.58229578,  0.34128394,
        -0.59563459, -0.08826078],
       [-0.20840982,  0.63345726,  0.40875435,  0.73440934,  0.51698859,
         0.67243274, -0.15328594],
       [ 1.19040416, -0.25450331, -1.1549009 ,  2.12106046, -1.05336487,
         2.23776343, -1.13465193],
       [-1.1831215 , -0.87028195,  0.44681341,  1.98322667,  0.53830956,
         1.9442532 ,  0.90866143],
       [ 0.13574177,  0.83434575,  1.41344862, -0.63899948,  1.67791749,
        -0.59005644, -0.71379529],
       [-0.16372869,  0.86324348, -1.31172732, -0.58942801, -1.16773268,
        -0.6047306 , -0.64119682],
       [ 0.25182364, -0.99684608,  0.65839645, -0.54672097,  0.84872262,
        -0.52936112,  1.15827623],
       [ 0.23422959,  0.32038874,  1.88815273, -0.65179307, -1.55172536,
        -0.57665647, -0.28363417],
       [ 0.68752537,  0.48036551,  0.28249096, -0.53878886,  0.46892137,
        -0.54507948, -0.76332259],
       [-0.83542211, -1.20432314,  0.37675641,  0.37001863,  0.47501918,
         0.3578328 ,  1.36568446],
       [ 1.36796382, -0.08175169, -1.20532878, -0.05333267, -1.073853  ,
        -0.03317495, -0.97765783]])
```

LAB7- EM Vs K-Means

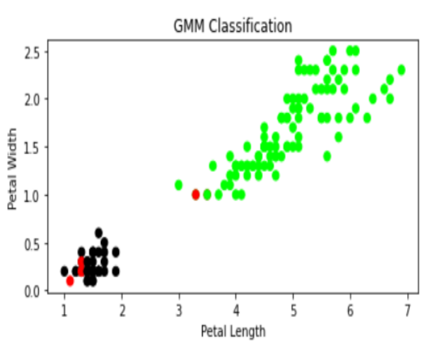
Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

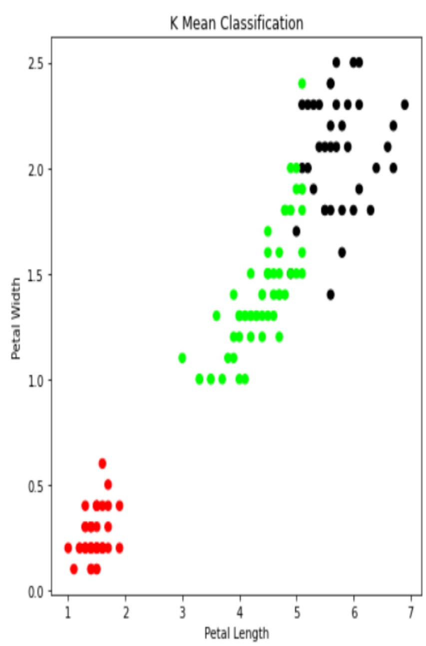
In [1]:

```
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)
```

```python
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:  0.8933333333333333
The Confusion matrixof K-Mean:  [[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
The accuracy score of EM:  0.35333333333333333
The Confusion matrix of EM:  [[ 5  0 45]
 [ 2 48  0]
 [ 0 50  0]]
```





LAB8- K-Nearest Neighbour

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

In [2]:
```
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]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
```

```
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
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
[[15  0  0]
 [ 0 13  0]
 [ 0  1 16]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       0.93      1.00      0.96        13
           2       1.00      0.94      0.97        17

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

LAB9- Linear Regression

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 [3]:
```
dataset = pd.read_csv('salary_dataset.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
```

In [4]:
```
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 [5]:
```
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[5]:
```
LinearRegression()
```

In [6]:
```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
```

In [7]:
```
# 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 [8]:

```
# 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()
```

LAB10- Logically Weighted Regression

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

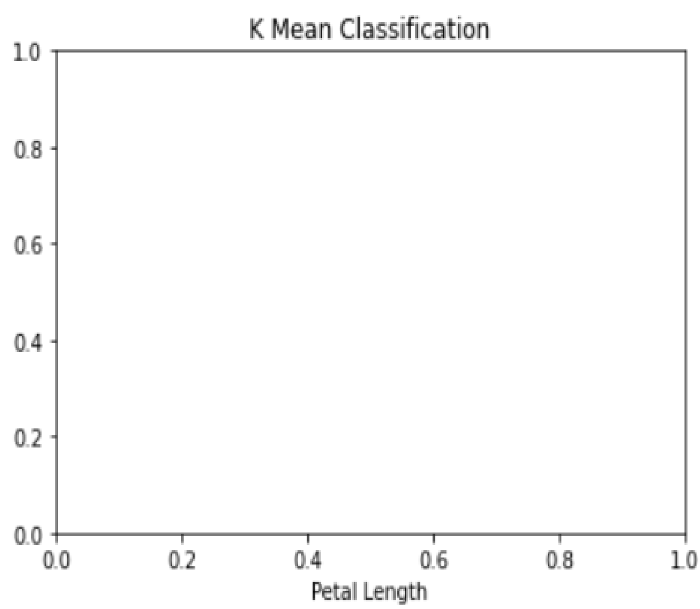
In [2]:

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

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)]]))
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
```

```
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 :
 [-2.88440998 -2.97461063 -2.97639127 -2.9042727  -3.1194782  -3.06506157
 -2.8349021  -2.90676221 -2.92454458]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```

```
Out[2]: Text(0.5, 0, 'Petal Length')
```



In [3]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m))) # eye - identity matrix
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.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 = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill,tip, color='green')
    ax.plot(xsort[:,1],ypred[sortindex], color = 'red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show();
```

```
    plt.show();

# load data points
data = pd.read_csv('tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips data
tip = np.array(data.tip)

mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols

# increase k to get smooth curves
ypred = localWeightRegression(X,mtip,3)
graphPlot(X,ypred)
```