B.M.S. COLLEGE OF ENGINEERING BENGALURU

Autonomous Institute, Affiliated to VTU

Lab Record

Machine Learning

Submitted in partial fulfillment for the 6th Semester Laboratory

Bachelor of Technology

in

Computer Science and Engineering

Submitted By

ABHISHIKAT KUMAR SONI (1BM18CS004)

Department of Computer Science and Engineering

B.M.S. College of Engineering

Bull Temple Road, Basavanagudi, Bangalore 560019

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the **Machine Learning(20CS6PCMAL)** laboratory has been carried out by ABHISHIKAT KUMAR SONI(1BM18CS004) during the 6th Semester Mar-June-2021.

Signature of the Faculty Incharge:

NAME OF THE FACULTY:
Dr. Asha GR
Assistant Professor
Department of Computer Science and Engineering
B.M.S. College of Engineering, Bangalore

LAB 1

Problem Statement:

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

CODE:

```
import pandas as pd
import numpy as np


data = pd.read_csv("./data.csv")
print(data,"\n")

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

target = np.array(data)[:,-1]
print("\n The target is: ",target)

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))
```

Output:

```
Data:
['GREEN', 'HARD', 'NO', 'WRINKLED', 'YES']
['GREEN', 'HARD', 'YES', 'SMOOTH', 'NO']
['BROWN', 'SOFT', 'NO', 'WRINKLED', 'NO']
['ORANGE', 'HARD', 'NO', 'WRINKLED', 'YES']
['GREEN', 'SOFT', 'NO', 'WRINKLED', 'YES']


Hypothesis:  ['?', '?', 'NO', 'WRINKLED']
```

LAB 2

Problem Statement:

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.

Code:

In [3]:
```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('/Users/abhishikatkumarsoni/Desktop/enjoysport.csv'))
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)
```

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

In [4]:
```
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)
    for i, h in enumerate(concepts):
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
                print(specific_h)
        print(specific_h)
        if target[i] == "no":
            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(" steps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
```

In [5]:
```
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

```
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?
', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?
', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

Output:

```
C:\SEM-6\ML\LAB-2>python cand_el.py
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], [
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], [
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
 steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], [
['sunny' 'warm' '?' 'strong' 'warm' 'same']
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' '?' 'same']
['sunny' 'warm' '?' 'strong' '?' '?']
['sunny' 'warm' '?' 'strong' '?' '?']
 steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Lab 3:

Problem Statement:

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.

Code:

```
import pandas as pd

import math
import numpy as np
import pprint



data=pd.read_csv("../input/dataset-id3/dataset.csv")
print("\n Input Data Set is:\n", data)
features = [f for f in data]
features.remove("answer")



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


def find_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))


def info_gain(examples, attr):
  uniq = np.unique(examples[attr])
  gain = find_entropy(examples)
  for u in uniq:
      subdata = examples[examples[attr] == u]
      sub_e = find_entropy(subdata)
      gain -= (float(len(subdata)) / float(len(examples))) * sub_e
  return gain



def id3(examples, attrs):
 root = Node()


 max_gain = 0
 max_feat = ""
 for feature in attrs:
     gain = info_gain(examples, feature)
     if gain > max_gain:
         max_gain = gain
         max_feat = feature
 root.value = max_feat
 uniq = np.unique(examples[max_feat])
 for u in uniq:
     subdata = examples[examples[max_feat] == u]
     if find_entropy(subdata) == 0.0:
         newNode = Node()
         newNode.isLeaf = True
         newNode.value = u
         newNode.pred = np.unique(subdata["answer"])
         root.children.append(newNode)
     else:
         tempNode = Node()
         tempNode.value = u
```

```
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = id3(subdata, new_attrs)
            tempNode.children.append(child)
            root.children.append(tempNode)
    return root



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)


root = id3(data, features)
print("Final decision tree:\n")
printTree(root)
```

Output:

```
 Input Data Set is:
     outlook temperature humidity    wind answer
0      sunny         hot     high    weak     no
1      sunny         hot     high  strong     no
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
13      rain        mild     high  strong     no
Final decision tree:

outlook
        overcast :  ['yes']

        rain
                wind
                        strong :  ['no']

                        weak :  ['yes']

        sunny
                humidity
                        high :  ['no']

                        normal :  ['yes']
```

Lab 4:

Problem Statement:

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

code:

```
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("/Users/suman/Downloads/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 # these are factors for the prediction
y = df[predicted_class_names].values # this is what we want to predict

#splitting the dataset into train and test data

xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)

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


# Training Naive Bayes (NB) classifier on training data.

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

#printing Confusion matrix, accuracy, Precision and Recall

print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))

print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))
```

```
print('\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)
```

Output:

```
 the total number of Training Data : (514, 1)

 the total number of Test Data : (254, 1)

 Confusion matrix
[[141  27]
 [ 29  57]]

 Accuracy of the classifier is 0.7795275590551181

 The value of Precision 0.6785714285714286

 The value of Recall 0.6627906976744186
Predicted Value for individual Test Data: [1]
```

Lab 5:

Problem Statement:

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

Code:

```
import numpy as np
import pandas as pd
import csv
!pip install pgmpy
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
```

```
Collecting pgmpy
  Downloading pgmpy-0.1.14-py3-none-any.whl (331 kB)
     |████████████████████████████████| 331 kB 3.0 MB/s
Requirement already satisfied: pandas in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.2.3)
Requirement already satisfied: scikit-learn in /opt/conda/lib/python3.7/site-packages (from pgmpy) (0.24.1)
Requirement already satisfied: scipy in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.5.4)
Requirement already satisfied: torch in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.7.0)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.7/site-packages (from pgmpy) (4.59.0)
Requirement already satisfied: joblib in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.0.1)
Requirement already satisfied: pyparsing in /opt/conda/lib/python3.7/site-packages (from pgmpy) (2.4.7)
Requirement already satisfied: statsmodels in /opt/conda/lib/python3.7/site-packages (from pgmpy) (0.12.2)
Requirement already satisfied: networkx in /opt/conda/lib/python3.7/site-packages (from pgmpy) (2.5)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.19.5)
Requirement already satisfied: decorator>=4.3.0 in /opt/conda/lib/python3.7/site-packages (from networkx->pgmpy) (4.4.2)
Requirement already satisfied: python-dateutil>=2.7.3 in /opt/conda/lib/python3.7/site-packages (from pandas->pgmpy) (2.8.1)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.7/site-packages (from pandas->pgmpy) (2021.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.7/site-packages (from python-dateutil>=2.7.3->pandas->pgmpy) (1.15.0)
```

Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from scikit-learn->pgmpy) (2.1.0)
Requirement already satisfied: patsy>=0.5 in /opt/conda/lib/python3.7/site-packages (from statsmodels->pgmpy) (0.5.1)
Requirement already satisfied: future in /opt/conda/lib/python3.7/site-packages (from torch->pgmpy) (0.18.2)
Requirement already satisfied: typing_extensions in /opt/conda/lib/python3.7/site-packages (from torch->pgmpy) (3.7.4.3)
Requirement already satisfied: dataclasses in /opt/conda/lib/python3.7/site-packages (from torch->pgmpy) (0.6)
Installing collected packages: pgmpy
Successfully installed pgmpy-0.1.14

In [2]:

```
heartDisease = pd.read_csv('../input/heartdisease/heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
```

```
Sample instances from the dataset are given below
   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
```

In [3]:

```
model =
BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisease
'),('heartdisease','restecg'),('heartdisease','chol')])
```

In [4]:

```
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\nInferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
```

```
Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:
```

In [5]:

```
print('\n1.Probability of HeartDisease given evidence = restecg :')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)

print('\n2.Probability of HeartDisease given evidence = cp :')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

Output:

```
Finding Elimination Order: :   0%|          | 0/5 [00:00<?, ?it/s]
  0%|          | 0/5 [00:00<?, ?it/s]
Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 480.98it/s]

Eliminating: age:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: sex:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: chol:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: exang: 100%|██████████| 5/5 [00:00<00:00, 92.85it/s]
1.Probability of HeartDisease given evidence = restecg :
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+

2.Probability of HeartDisease given evidence = cp :
Finding Elimination Order: :   0%|          | 0/5 [00:00<?, ?it/s]
  0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: age:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: sex:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: chol:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: exang:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: restecg: 100%|██████████| 5/5 [00:00<00:00, 227.41it/s]
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

Lab 6:

Problem Statement:

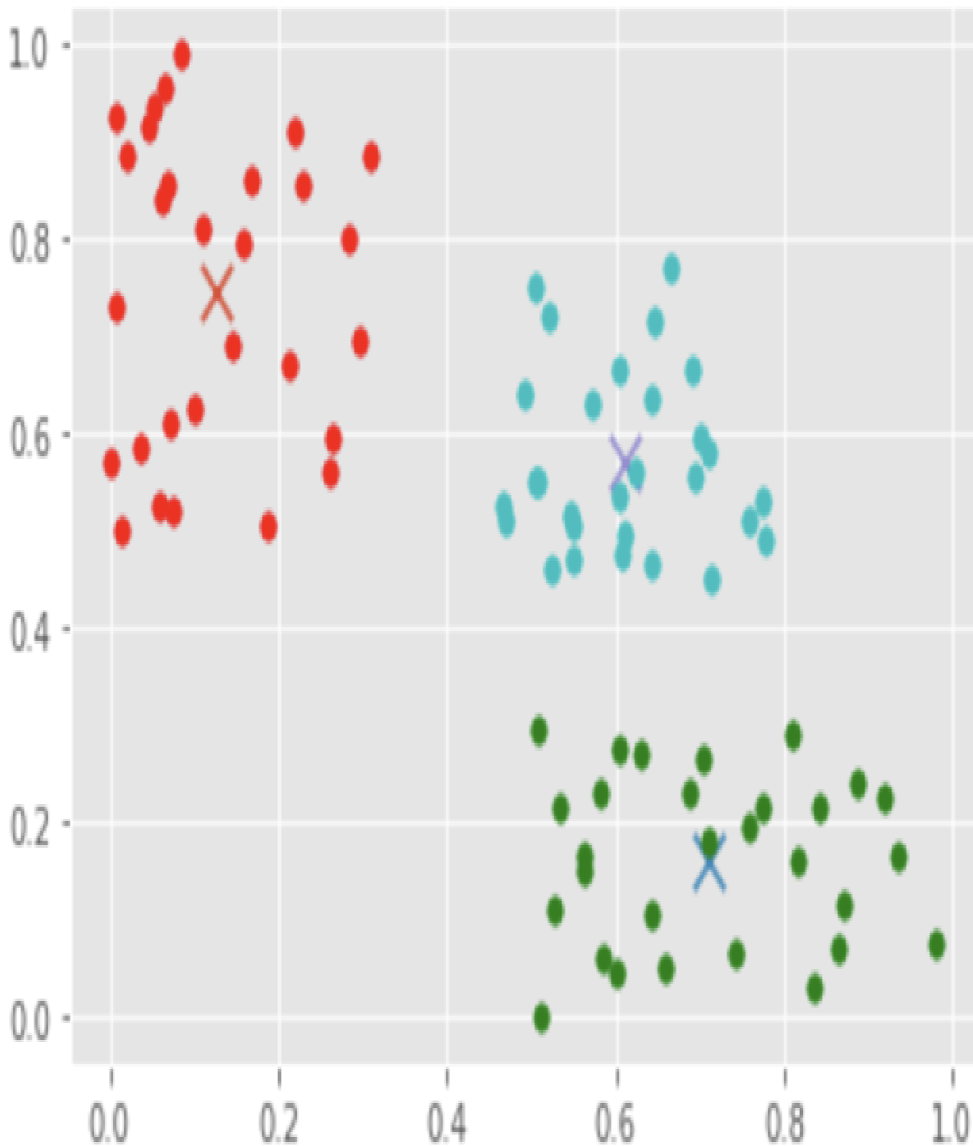
Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

Code:

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

style.use('ggplot')

class K_Means:
    def __init__(self, k =3, tolerance = 0.0001, max_iterations = 500):
        self.k = k
        self.tolerance = tolerance
        self.max_iterations = max_iterations

    def fit(self, data):

        self.centroids = {}

        #initialize the centroids, the first 'k' elements in the dataset will be our initial centroids
        for i in range(self.k):
            self.centroids[i] = data[i]
        #begin iterations
        for i in range(self.max_iterations):
            self.classes = {}
            for i in range(self.k):
                self.classes[i] = []

            #find the distance between the point and cluster; choose the nearest centroid
            for features in data:
                distances = [np.linalg.norm(features - self.centroids[centroid]) for centroid in self.centroids]
                classification = distances.index(min(distances))
                self.classes[classification].append(features)

            previous = dict(self.centroids)
            #average the cluster datapoints to re-calculate the centroids
            for classification in self.classes:
                self.centroids[classification] = np.average(self.classes[classification], axis = 0)
```

```
            isOptimal = True

            for centroid in self.centroids:

                original_centroid = previous[centroid]
                curr = self.centroids[centroid]

                if np.sum((curr - original_centroid)/original_centroid * 100.0) > self.tolerance:
                    isOptimal = False

            #break out of the main loop if the results are optimal, ie. the centroids don't change their
positions much(more than our tolerance)
            if isOptimal:
                break

    def pred(self, data):
        distances = [np.linalg.norm(data - self.centroids[centroid]) for centroid in self.centroids]
        classification = distances.index(min(distances))
        return classification

def main():

    df = pd.read_csv('data.csv')
    df = df[['one', 'two']]
    dataset = df.astype(float).values.tolist()

    X = df.values #returns a numpy array

    km = K_Means(3)
    km.fit(X)

        # Plotting starts here
    colors = 10*["r", "g", "c", "b", "k"]
    for centroid in km.centroids:
        plt.scatter(km.centroids[centroid][0], km.centroids[centroid][1], s = 130, marker = "x")

    for classification in km.classes:
        color = colors[classification]
        for features in km.classes[classification]:
            plt.scatter(features[0], features[1], color = color,s = 30)

    plt.show()


main()
```

Output:



Lab 7:

Problem Statement:

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

Code:

```
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
```

In [2]:

```
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']
```

In [3]:

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


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

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

<Figure size 1008x504 with 0 Axes>

In [4]:

```
# 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')
```

Out[4]:

```
Text(0, 0.5, 'Petal Width')
```

In [13]:

```
# 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')
```

Out[13]:

```
Text(0, 0.5, 'Petal Width')
```

In [14]:

```
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean:\n',sm.confusion_matrix(y, model.labels_))
```

```
The accuracy score of K-Mean:  0.8933333333333333
The Confusion matrixof K-Mean:
 [[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
```

In [7]:

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

In [8]:

```
from sklearn.mixture import GaussianMixture
```

```
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm = gmm.predict(xs)
#y_cluster_gmm
```

In [10]:

```
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')
```

Out[10]:

```
Text(0, 0.5, 'Petal Width')
```

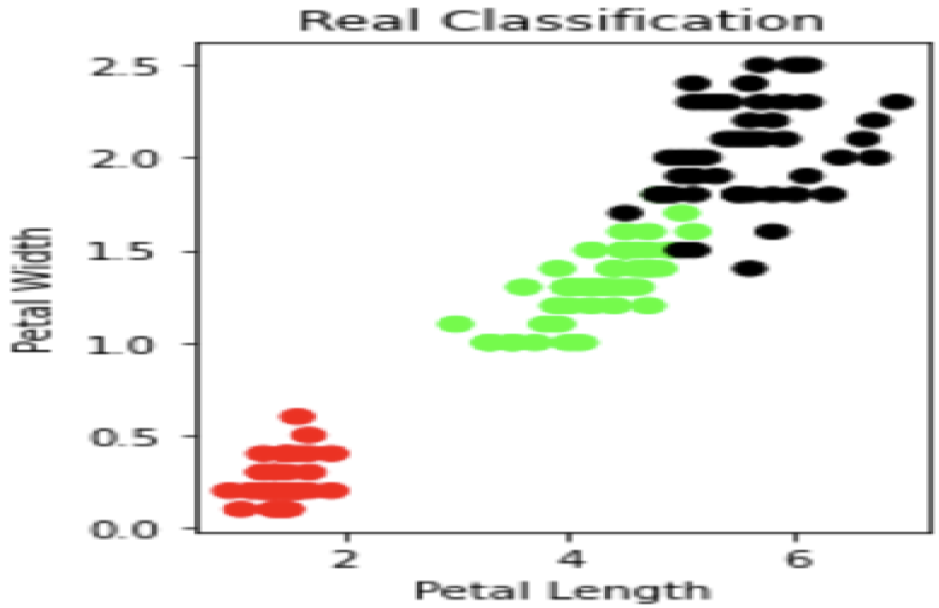
In [12]:

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

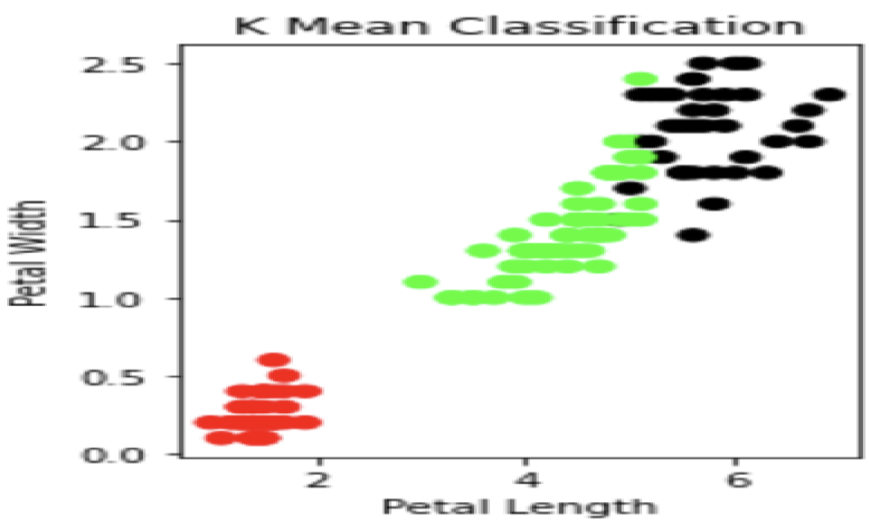
```
The accuracy score of EM:  0.36666666666666664
The Confusion matrix of EM:
 [[50  0  0]
 [ 0  5 45]
 [ 0 50  0]]
```

In [ ]:

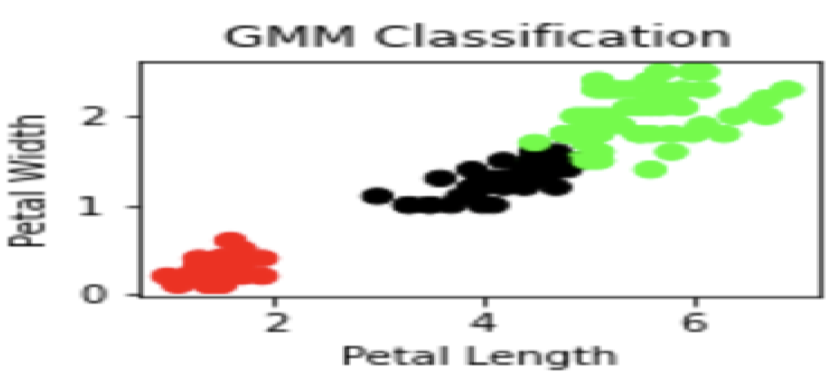
Output:



Real Classification
Petal Width
Petal Length
0.0
0.5
1.0
1.5
2.0
2.5
2
4
6



K Mean Classification
Petal Width
Petal Length
0.0
0.5
1.0
1.5
2.0
2.5
2
4
6



GMM Classification
Petal Width
Petal Length
0
1
2
2
4
6

LAB 8

Problem Statement:

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

Code:

```
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))
```

Output:

```
 [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
[[16  0  0]
 [ 0 11  0]
 [ 0  0 18]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        18

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

LAB 9

Problem Statement:

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

Code:

```
# Standalone simple linear regression example
from math import sqrt

# Calculate root mean squared error
def rmse_metric(actual, predicted):
	sum_error = 0.0
	for i in range(len(actual)):
		prediction_error = predicted[i] - actual[i]
		sum_error += (prediction_error ** 2)
	mean_error = sum_error / float(len(actual))
	return sqrt(mean_error)

# Evaluate regression algorithm on training dataset
def evaluate_algorithm(dataset, algorithm):
	test_set = list()
	for row in dataset:
		row_copy = list(row)
		row_copy[-1] = None
		test_set.append(row_copy)
	predicted = algorithm(dataset, test_set)
	print(predicted)
	actual = [row[-1] for row in dataset]
	rmse = rmse_metric(actual, predicted)
	return rmse

# Calculate the mean value of a list of numbers
def mean(values):
	return sum(values) / float(len(values))

# Calculate covariance between x and y
def covariance(x, mean_x, y, mean_y):
	covar = 0.0
	for i in range(len(x)):
		covar += (x[i] - mean_x) * (y[i] - mean_y)
	return covar

# Calculate the variance of a list of numbers
def variance(values, mean):
	return sum([(x-mean)**2 for x in values])

# Calculate coefficients
def coefficients(dataset):
	x = [row[0] for row in dataset]
	y = [row[1] for row in dataset]
	x_mean, y_mean = mean(x), mean(y)
	b1 = covariance(x, x_mean, y, y_mean) / variance(x, x_mean)
	b0 = y_mean - b1 * x_mean
	return [b0, b1]

# Simple linear regression algorithm
def simple_linear_regression(train, test):
	predictions = list()
	b0, b1 = coefficients(train)
	for row in test:
		yhat = b0 + b1 * row[0]
		predictions.append(yhat)
	return predictions

# Test simple linear regression
dataset = [[1, 1], [2, 3], [4, 3], [3, 2], [5, 5]]
rmse = evaluate_algorithm(dataset, simple_linear_regression)
print('RMSE: %.3f' % (rmse))
```

Output:

```
rmse = evaluate_algorithm(dataset, simple_linear_regression)
print('RMSE: %.3f' % (rmse))

[1.1999999999999995, 1.9999999999999996, 3.5999999999999996, 2.8, 4.3999999999999995]
RMSE: 0.693
```

LAB 10

Problem Statement:

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

Code:

```
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('/Users/abhishikatkumarsoni/Desktop/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,0.3)
SortIndex = X[:,1].argsort(0)
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();
```

Output: