VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT on

Machine Learning

Submitted by

HARSHA V N(1BM20CS406)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING BENGALURU-560019 May-2022 to July-2022

(Autonomous Institution under VTU)

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 **HARSHA V N(1BM20CS406),** 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	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

`

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Find-S	
2	Candidate Elimination	
3	Decision Tree	
4	Naïve Bayes	
5	Linear Regression	

Course Outcome

CO1	Ability to apply the different learning algorithms.
CO2	Ability to analyze the learning techniques for 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.

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

```
In [14]: import numpy as np
         import pandas as pd
```

```
In [15]: data = pd.read_csv("finddata.csv")
         print(data,"\n")
```

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

```
In [19]: d = np.array(data)[:,:-1]
         print("\n The attributes are: ",d)
         target = np.array(data)[:,-1]
         print("\n The target is: ",target)
```

```
 The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High']]

 The target is:  ['Yes' 'No' 'Yes' 'Yes']
```

```
In [17]: 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
```

```
In [18]: print("\n The final hypothesis is:",findS(d,target))
```

```
 The final hypothesis is: ['?' '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 [4]: import numpy as np
        import pandas as pd

        #to read the data in the csv file
        data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
        print(data,"\n")

        #making an array of all the attributes
        concepts = np.array(data.iloc[:,0:-1])
        print("The attributes are: ",concepts)

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

        #training function to implement candidate_elimination algorithm
        def learn(concepts, target):
         specific_h = concepts[0].copy()
         print("\n 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)
             if target[i] == "no":
                 for x in range(len(specific_h)):
                     if h[x]!= specific_h[x]:
```

```
        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
s_final, g_final = learn(concepts, target)

#obtaining the final hypothesis
print("\nFinal Specific_h:", s_final, sep="\n")
print("\nFinal General_h:", g_final, sep="\n")
```

```
     sky  temp humidity    wind water forcast enjoysport
0  sunny  warm   normal  strong  warm    same        yes
1  sunny  warm     high  strong  warm    same        yes
2  rainy  cold     high  strong  warm  change         no
3  sunny  warm     high  strong  cool  change        yes

The attributes 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']]

 The target is:  ['yes' 'yes' 'no' 'yes']

 Initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

 Steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

 Steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

 Steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

 Steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

 Steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['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 [24]:
```
import pandas as pd
import math
import numpy as np
```

In [34]:
```
data = pd.read_csv("data.csv")
features = [feat for feat in data]
features.remove("answer")
```

In [37]:
```
class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
```

In [38]:
```
def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
```

In [39]:
```
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain
```

```
In [40]: def ID3(examples, attrs):
             root = Node()

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

```
In [41]: def printTree(root: Node, depth=0):
             for i in range(depth):
                 print("\t", end="")
             print(root.value, end="")
             if root.isLeaf:
                 print(" -> ", root.pred)
             print()
             for child in root.children:
                 printTree(child, depth + 1)
```

```
In [42]: root = ID3(data, features)
         printTree(root)

outlook
	overcast ->  ['yes']

	rain
		wind
			strong ->  ['no']

			weak ->  ['yes']

	sunny
		humidity
			high ->  ['no']

			normal ->  ['yes']
```

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

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

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

Out[2]:

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

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

```
In [4]: 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
```

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

In [17]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import r2_score
```

In [9]:
```
dataset = pd.read_csv('salary_dataset.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
```

In [10]:
```
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 [11]:
```
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[11]:
```
LinearRegression()
```

In [15]:
```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
y_pred
```

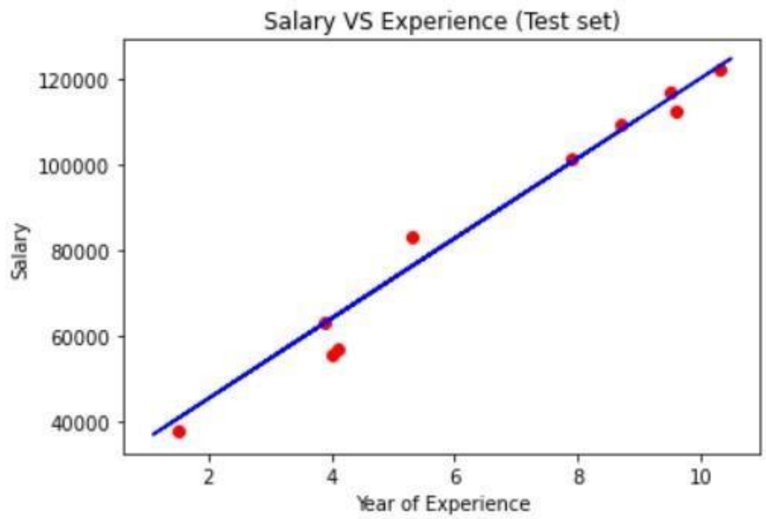
Out[15]:
```
array([ 40835.10590871, 123079.39940819,  65134.55626083,  63265.36777221,
       115602.64545369, 108125.8914992 , 116537.23969801,  64199.96201652,
        76349.68719258, 100649.1375447 ])
```

In [18]:
```
r2_score(y_test,y_pred)
```

Out[18]:
```
0.9749154407708353
```

In [19]:
```
# 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 [14]:
```
# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```



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

a) Using built-in:

```
!pip install pgmpy
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
heartDisease = pd.read_csv('heart_disease.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)
model=
BayesianModel([('age','Heartdisease'),('sex','Heartdisease'),('exang','Heartdisease'),('cp','Heartdisease'),('Heart
disease','restecg'),('Heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

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

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

Output:

```
Inferencing with Bayesian Network:

1. Probability of HeartDisease given evidence= restecg
```

Finding Elimination Order: : 100% 4/4 [00:00<00:00, 100.26it/s]

Eliminating: exang: 100% 4/4 [00:00<00:00, 190.96it/s]

```
+-----------------+---------------------+
| 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: : 100% 3/3 [00:00<00:00, 60.16it/s]

Eliminating: exang: 100% 3/3 [00:00<00:00, 91.15it/s]

```
+-----------------+---------------------+
| Heartdisease    |   phi(Heartdisease) |
+=================+=====================+
| Heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| Heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| Heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| Heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| Heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

b) Without using built-in:

```
import bayespy as bp
import numpy as np
import csv
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}
import pandas as pd
data = pd.read_csv("heart_disease_data.csv")
data =np.array(data, dtype='int8')
N = len(data)
# 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])
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()
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 FamilyHistory: ' + str(familyHistoryEnum))), int(input('Enter dietEnum: ' + str(
       dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter Cholesterol: ' +
str(cholesterolEnum)))], bp.nodes.Categorical, p_heartdisease).get_moments()[0][heartDiseaseEnum['Yes']]
   print("Probability(HeartDisease) = " + str(res))

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

Output:

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

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

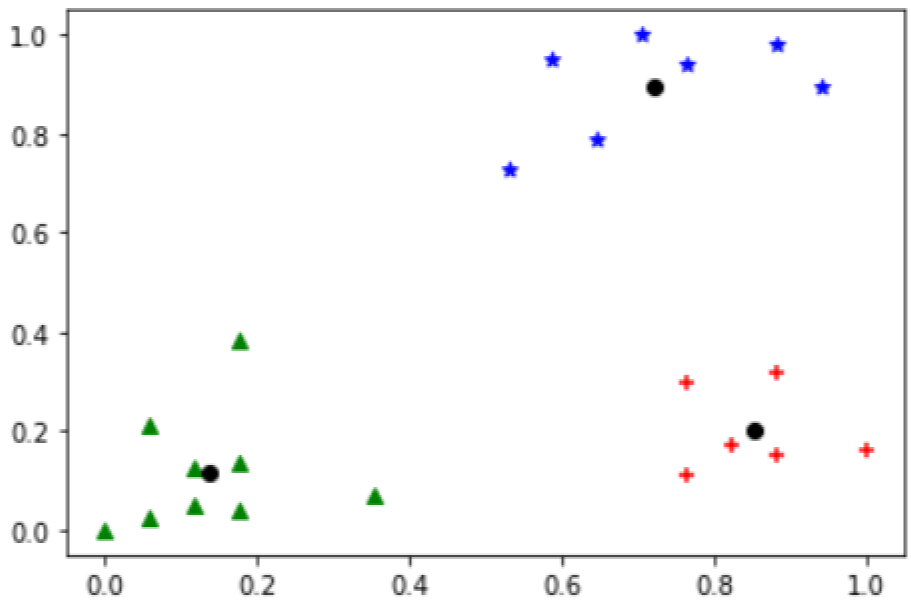
a) Using built-in:

```
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)
scaler = MinMaxScaler()
scaler.fit(df[['Age']])
df[['Age']] = scaler.transform(df[['Age']])

scaler.fit(df[['Income($)']])
df[['Income($)']] = scaler.transform(df[['Income($)']])
df.head(10)
plt.scatter(df['Age'], df['Income($)'])

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

km = KMeans(n_clusters=3)
km
df0 = df[df.cluster == 0]
df0
df1 = df[df.cluster == 1]
df1
df2 = df[df.cluster == 2]
df2
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'))
```

Output:

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



b) Without using built-in:

```
import math;
import sys;
import pandas as pd
import numpy as np
from random import choice
from matplotlib import pyplot
from random import shuffle, uniform;
def ReadData(fileName):
    f = open(fileName,'r')
    lines = f.read().splitlines()
    f.close()

    items = []
    for i in range(1,len(lines)):
        line = lines[i].split(',')
        itemFeatures = []
        for j in range(len(line)-1):
            v = float(line[j])
            itemFeatures.append(v)
        items.append(itemFeatures)
    shuffle(items)
    return items
def FindColMinMax(items):
    n = len(items[0])
    minima = [float('inf') for i in range(n)]
    maxima = [float('-inf') -1 for i in range(n)]
    for item in items:
        for f in range(len(item)):
            if(item[f] < minima[f]):
                minima[f] = item[f]
            if(item[f] > maxima[f]):
                maxima[f] = item[f]
    return minima,maxima

def EuclideanDistance(x,y):
    S = 0
    for i in range(len(x)):
        S += math.pow(x[i]-y[i],2)
    return math.sqrt(S)
def InitializeMeans(items,k,cMin,cMax):
```

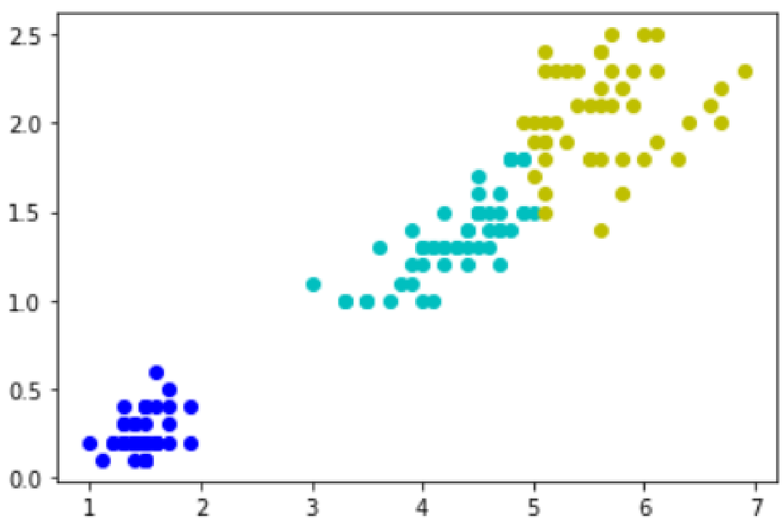
```
    f = len(items[0])
    means = [[0 for i in range(f)] for j in range(k)]
    for mean in means:
        for i in range(len(mean)):
            mean[i] = uniform(cMin[i]+1,cMax[i]-1)

    return means

def UpdateMean(n,mean,item):
    for i in range(len(mean)):
        m = mean[i]
        m = (m*(n-1)+item[i])/float(n)
        mean[i] = round(m,3)
    return mean
def FindClusters(means,items):
    clusters = [[] for i in range(len(means))]
    for item in items:
        index = Classify(means,item)
        clusters[index].append(item)
    return clusters

def Classify(means,item):
        minimum = float('inf');
    index = -1
    for i in range(len(means)):
        dis = EuclideanDistance(item,means[i])
        if(dis < minimum):
            minimum = dis
            index = i
        return index
def CalculateMeans(k,items,maxIterations=100000):
    cMin, cMax = FindColMinMax(items)
    means = InitializeMeans(items,k,cMin,cMax)
    clusterSizes = [0 for i in range(len(means))]
    belongsTo = [0 for i in range(len(items))]
    for e in range(maxIterations):
        noChange = True;
        for i in range(len(items)):
            item = items[i];
            index = Classify(means,item)
            clusterSizes[index] += 1
```

```
            cSize = clusterSizes[index]
            means[index] = UpdateMean(cSize,means[index],item)
            if(index != belongsTo[i]):
                noChange = False
            belongsTo[i] = index

        if (noChange):
            break
    return means

def CutToTwoFeatures(items,indexA,indexB):
    n = len(items)
    X = []
    for i in range(n):
        item = items[i]
        newItem = [item[indexA],item[indexB]]
        X.append(newItem)
    return X

def PlotClusters(clusters):
    n = len(clusters)
    X = [[] for i in range(n)]
    for i in range(n):
        cluster = clusters[i]
        for item in cluster:
            X[i].append(item)
    colors = ['r','b','g','c','m','y']
    for x in X:
        c = choice(colors)
        colors.remove(c)
        Xa = []
        Xb = []
        for item in x:
            Xa.append(item[0])
            Xb.append(item[1])
        pyplot.plot(Xa,Xb,'o',color=c)
    pyplot.show()



def main():
    items = ReadData('data.txt')
```

```
    k = 3
    items = CutToTwoFeatures(items,2,3)
    print(items)
    means = CalculateMeans(k,items)
    print("\nMeans = ", means)
    clusters = FindClusters(means,items)
    PlotClusters(clusters)
    newItem = [1.5,0.2]
    print(Classify(means,newItem))

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

Output:

```
Means =  [[4.308, 1.372], [5.639, 2.059], [1.465, 0.255]]
```




```
2
```

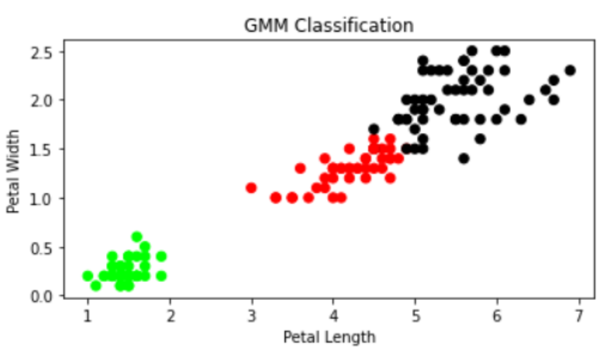
8) Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

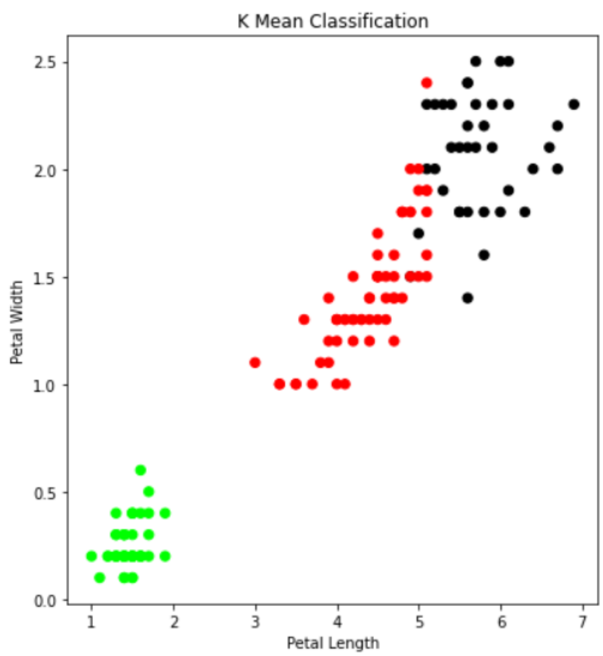
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
model = KMeans(n_clusters=3)
model.fit(X)
plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))

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

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

```
gmm.fit(xs)

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

Output:

```
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
The accuracy score of EM:  0.3333333333333333
The Confusion matrix of EM:  [[ 0 50  0]
 [45  0  5]
 [ 0  0 50]]
```





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

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

iris=datasets.load_iris()

x = iris.data
y = iris.target
print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)
#To Training the model and Nearest 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:

```
 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
[[14  0  0]
 [ 0 14  0]
 [ 0  2 15]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.88      1.00      0.93        14
           2       1.00      0.88      0.94        17

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

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

a) Using built-in:

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

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

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

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


 # predict value
 return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction
def radial_kernel(x0, X, tau):
 return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
# Weight or Radial 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)]]))
```

Output:

```
The Data Set ( 10 Samples) X :
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y :
 [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
 2.11015444 2.10584249 2.10152068]
Normalised (10 Samples) X :
 [-3.08663662 -2.79327673 -3.13292877 -3.03726639 -3.0967025  -2.9652877
 -3.00708877 -2.94234969 -2.79405157]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```

b) Without using built-in:

```
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)))
    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
data = pd.read_csv('10-dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

#preparing and add 1 in bill
mbill = np.mat(bill)
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))
ypred = localWeightRegression(X,mtip,0.5)
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: