VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LERANING LAB

Submitted by

RAHUL RAJ (1BM20CS120)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019

May-2023 to July-2023

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 LAB" carried out by **RAHUL RAJ (1BM20CS120),** 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 2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **RAHUL RAJ - (20CS6PCMAL)** work prescribed for the said degree.

Saritha N	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

`

Index Sheet

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.

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

```
import csv
a = []
with open('enjoysport.csv', 'r') as csvfile:
    for row in csv.reader(csvfile):
        a.append(row)
    print(a)
print("\n The total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\n The initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)
for i in range(0, len(a)):
    if a[i][num_attribute] == 'yes':
        for j in range(0, num_attribute):
            if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
                hypothesis[j] = a[i][j]
            else:
                hypothesis[j] = '?'
    print("\n The hypothesis for the training instance {} is : \n" .format(i+1),hypothesis)
print("\n The Maximally specific hypothesis for the training instance is ")
print(hypothesis)
```

Output:

```
 The total number of training instances are :  5

 The initial hypothesis is :
['0', '0', '0', '0', '0', '0']

 The hypothesis for the training instance 1 is :
 ['0', '0', '0', '0', '0', '0']

 The hypothesis for the training instance 2 is :
 ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

 The hypothesis for the training instance 3 is :
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

 The hypothesis for the training instance 4 is :
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

 The hypothesis for the training instance 5 is :
 ['sunny', 'warm', '?', 'strong', '?', '?']

 The Maximally specific hypothesis for the training instance is
['sunny', 'warm', '?', 'strong', '?', '?']
```

Program 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.

```
import numpy as np
import pandas as pd
data = pd.read_csv('enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)
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):
        print("For Loop Starts")
        if target[i] == "yes":
            print("If 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("If 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(" steps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
        print("\n")
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

Output:

```
[['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']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
```

```
For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '



For Loop Starts
If instance is Negative
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],



For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],



Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Program 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

```
import math
import csv
def load_csv(filename):
    lines=csv.reader(open(filename,"r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset,headers
class Node:
    def __init__(self,attribute):
        self.attribute=attribute
        self.children=[]
        self.answer=""
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
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)
'''Main program'''
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_test.csv")
for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:",end="   ")
    classify(node1,xtest,features)
```

<u>Output:</u>

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

Program 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

```
import numpy as np
import math
import csv
import pdb
def read_data(filename):
    with open(filename,'r') as csvfile:
        datareader = csv.reader(csvfile)
        metadata = next(datareader)
        traindata=[]
        for row in datareader:
            traindata.append(row)
    return (metadata, traindata)
def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    testset = list(dataset)
    i=0
    while len(trainSet) < trainSize:
        trainSet.append(testset.pop(i))
    return [trainSet, testset]
def classify(data,test):
    total_size = data.shape[0]
    print("\n")
    print("training data size=",total_size)
    print("test data size=",test.shape[0])
    countYes = 0
```

```
countNo = 0
probYes = 0
probNo = 0
print("\n")
print("target    count    probability")
for x in range(data.shape[0]):
    if data[x,data.shape[1]-1] == 'yes':
        countYes +=1
    if data[x,data.shape[1]-1] == 'no':
        countNo +=1
probYes=countYes/total_size
probNo= countNo / total_size
print('Yes',"\t",countYes,"\t",probYes)
print('No',"\t",countNo,"\t",probNo)
prob0 =np.zeros((test.shape[1]-1))
prob1 =np.zeros((test.shape[1]-1))
accuracy=0
print("\n")
print("instance prediction  target")
for t in range(test.shape[0]):
    for k in range (test.shape[1]-1):
        count1=count0=0
        for j in range (data.shape[0]):
            #how many times appeared with no
            if test[t,k] == data[j,k] and data[j,data.shape[1]-1]=='no':
                count0+=1
            #how many times appeared with yes
            if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='yes':
```

```
                count1+=1
            if countNo != 0:
                prob0[k] = count0 / countNo
            else:
                prob0[k] = 0
            prob1[k] = count1 / countYes
        probno=probNo
        probyes=probYes
        for i in range(test.shape[1]-1):
            probno=probno*prob0[i]
            probyes=probyes*prob1[i]
        if probno>probyes:
            predict='no'
        else:
            predict='yes'
        print(t+1,"\t",predict,"\t    ",test[t,test.shape[1]-1])
        if predict == test[t,test.shape[1]-1]:
            accuracy+=1
    final_accuracy=(accuracy/test.shape[0])*100
    print("accuracy",final_accuracy,"%")
    return
metadata,traindata= read_data("naive.csv")
splitRatio=0.6
trainingset, testset=splitDataset(traindata, splitRatio)
training=np.array(trainingset)
print("\n The Training data set are:")
for x in trainingset:
    print(x)
```

```
testing=np.array(testset)
print("\n The Test data set are:")
for x in testing:
    print(x)
classify(training,testing)
```

Output:

```
 The Training data set are:
['sunny', 'hot', 'high', 'FALSE', 'no']
['sunny', 'hot', 'high', 'TRUE', 'no']
['overcast', 'hot', 'high', 'FALSE', 'yes']
['rainy', 'mild', 'high', 'FALSE', 'yes']
['rainy', 'cool', 'normal', 'FALSE', 'yes']
['rainy', 'cool', 'normal', 'TRUE', 'no']

 The Test data set are:
['overcast' 'cool' 'normal' 'TRUE' 'yes']
['sunny' 'mild' 'high' 'FALSE' 'no']
['sunny' 'cool' 'normal' 'FALSE' 'yes']
['rainy' 'mild' 'normal' 'FALSE' 'yes']
['sunny' 'mild' 'normal' 'TRUE' 'yes']


training data size= 6
test data size= 5


target    count    probability
Yes       3        0.5
No        3        0.5


instance prediction  target
1        yes         yes
2        yes         no
3        no          yes
4        yes         yes
5        yes         yes
accuracy 60.0 %
```

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

```
import numpy as np

import pandas as pd

import csv

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

heartDisease = pd.read_csv('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)

model=
BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdiseas
e'),('heartdisease','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:

```
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
```

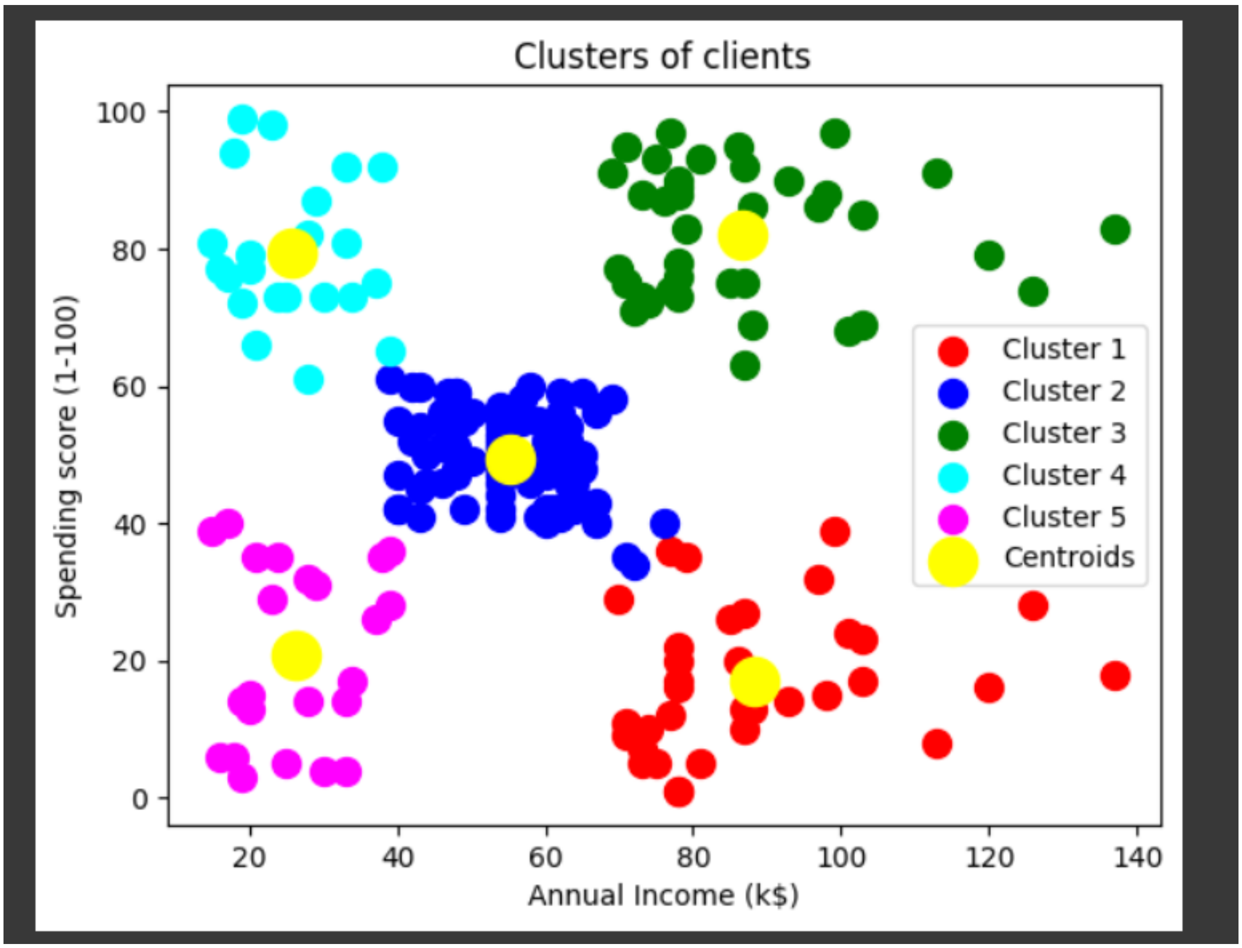
```
Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

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

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

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('/content/drive/MyDrive/ML Lab/Mall_Customers.csv')
X = dataset.iloc[:,[3,4]].values
# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss =[]
for i in range (1,11):
   kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter =300, n_init = 10, random_state
= 0)
   kmeans.fit(X)
   wcss.append(kmeans.inertia_)
# Plot the graph to visualize the Elbow Method to find the optimal number of cluster
"""plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()"""
# Applying KMeans to the dataset with the optimal number of cluster
kmeans=KMeans(n_clusters= 5, init = 'k-means++', max_iter = 300, n_init = 10, random_state =
0)
y_kmeans = kmeans.fit_predict(X)
# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0,1],s = 100, c='red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1,1],s = 100, c='blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2,1],s = 100, c='green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3,1],s = 100, c='cyan', label = 'Cluster 4')
```

```
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4,1],s = 100, c='magenta', label = 'Cluster 5')

plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s = 300, c = 'yellow', label
= 'Centroids')

plt.title('Clusters of clients')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending score (1-100)')

plt.legend()

plt.show()
```

Output:



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

```
import math
days = [70, 65, 90, 55, 100, 75, 80, 55, 30, 80]
std = 10
variance = std**2
k = 2 # sunny and cloudy days
avg_sunny = 80
avg_cloudy = 55
for n in range(1000):
    # estimator step
    E_sunny = []
    E_cloudy = []
    for i in range(len(days)):
        val_sunny = math.pow(math.e, (-0.5)/(variance) *
                        math.pow(days[i]-avg_sunny, 2))
        val_cloudy = math.pow(math.e, (-0.5)/(variance)
                        * math.pow(days[i]-avg_cloudy, 2))
        E_sunny.append(val_sunny/(val_sunny + val_cloudy))
        E_cloudy.append(val_cloudy/(val_sunny + val_cloudy))
    # maximization step
    sunny_numerator = 0
    sunny_denominator = sum(E_sunny)
    cloudy_numerator = 0
    cloudy_denominator = sum(E_cloudy)
    for i in range(len(days)):
        sunny_numerator += E_sunny[i]*days[i]
        cloudy_numerator += E_cloudy[i]*days[i]
    if (sunny_numerator/sunny_denominator) - avg_sunny <= 0.1 and (cloudy_numerator/cloudy_denominator) - avg_cloudy <= 0.1:
        break
    avg_sunny = sunny_numerator/sunny_denominator
    avg_cloudy = cloudy_numerator/cloudy_denominator
    print(avg_sunny, avg_cloudy)
```

<u>Output:</u>

```
82.00761549152422 52.981059926333586
82.12894336998127 52.562823852049945
```

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

```
from csv import reader
from sys import exit
from math import sqrt
from operator import itemgetter
def load_data_set(filename):
    try:
        with open(filename, newline='') as iris:
            return list(reader(iris, delimiter=','))
    except FileNotFoundError as e:
        raise e
def convert_to_float(data_set, mode):
    new_set = []
    try:
        if mode == 'training':
            for data in data_set:
                new_set.append([float(x) for x in data[:len(data)-1]] + [data[len(data)-1]])
        elif mode == 'test':
            for data in data_set:
                new_set.append([float(x) for x in data])
        else:
            print('Invalid mode, program will exit.')
            exit()
        return new_set
    except ValueError as v:
        print(v)
        print('Invalid data set format, program will exit.')
        exit()
```

```
def get_classes(training_set):
    return list(set([c[-1] for c in training_set]))
def find_neighbors(distances, k):
    return distances[0:k]
def find_response(neighbors, classes):
    votes = [0] * len(classes)
    for instance in neighbors:
        for ctr, c in enumerate(classes):
            if instance[-2] == c:
                votes[ctr] += 1
    return max(enumerate(votes), key=itemgetter(1))
def knn(training_set, test_set, k):
    distances = []
    dist = 0
    limit = len(training_set[0]) - 1
    # generate response classes from training data
    classes = get_classes(training_set)
    try:
        for test_instance in test_set:
            for row in training_set:
                for x, y in zip(row[:limit], test_instance):
                    dist += (x-y) * (x-y)
                distances.append(row + [sqrt(dist)])
                dist = 0
            distances.sort(key=itemgetter(len(distances[0])-1))
            # find k nearest neighbors
            neighbors = find_neighbors(distances, k)
            # get the class with maximum votes
```

```
            index, value = find_response(neighbors, classes)
            # Display prediction
            print('The predicted class for sample ' + str(test_instance) + ' is : ' + classes[index])
            print('Number of votes : ' + str(value) + ' out of ' + str(k))
            # empty the distance list
            distances.clear()
    except Exception as e:
        print(e)
def main():
    try:
        # get value of k
        k = int(input('Enter the value of k : '))
        # load the training and test data set
        training_file = input('Enter name of training data file : ')
        test_file = input('Enter name of test data file : ')
        training_set = convert_to_float(load_data_set(training_file), 'training')
        test_set = convert_to_float(load_data_set(test_file), 'test')
        if not training_set:
            print('Empty training set')
        elif not test_set:
            print('Empty test set')
        elif k > len(training_set):
            print('Expected number of neighbors is higher than number of training data instances')
        else:
            knn(training_set, test_set, k)
    except ValueError as v:
        print(v)
    except FileNotFoundError:
```

```
        print('File not found')
if __name__ == '__main__':
    main()
```

Output:

```
Enter the value of k : 5
Enter name of training data file : iris-dataset.csv
Enter name of test data file : iris-test.csv
The predicted class for sample [4.3, 2.9, 1.7, 0.3] is : Iris-setosa
Number of votes : 5 out of 5
The predicted class for sample [4.6, 2.7, 1.5, 0.2] is : Iris-setosa
Number of votes : 5 out of 5
The predicted class for sample [5.3, 3.4, 1.6, 0.2] is : Iris-setosa
Number of votes : 5 out of 5
The predicted class for sample [5.2, 4.1, 1.5, 0.1] is : Iris-setosa
Number of votes : 5 out of 5
The predicted class for sample [6.0, 2.2, 4.2, 1.0] is : Iris-versicolor
Number of votes : 5 out of 5
The predicted class for sample [6.2, 2.3, 4.5, 1.5] is : Iris-versicolor
Number of votes : 4 out of 5
The predicted class for sample [5.0, 2.1, 3.6, 1.2] is : Iris-versicolor
Number of votes : 5 out of 5
The predicted class for sample [6.6, 2.8, 5.4, 2.0] is : Iris-virginica
Number of votes : 5 out of 5
The predicted class for sample [6.4, 3.2, 5.3, 2.3] is : Iris-virginica
Number of votes : 5 out of 5
The predicted class for sample [7.0, 3.1, 5.5, 1.8] is : Iris-virginica
Number of votes : 5 out of 5
The predicted class for sample [6.2, 3.3, 5.9, 2.1] is : Iris-virginica
Number of votes : 5 out of 5
The predicted class for sample [6.6, 2.9, 5.3, 2.3] is : Iris-virginica
Number of votes : 5 out of 5
```

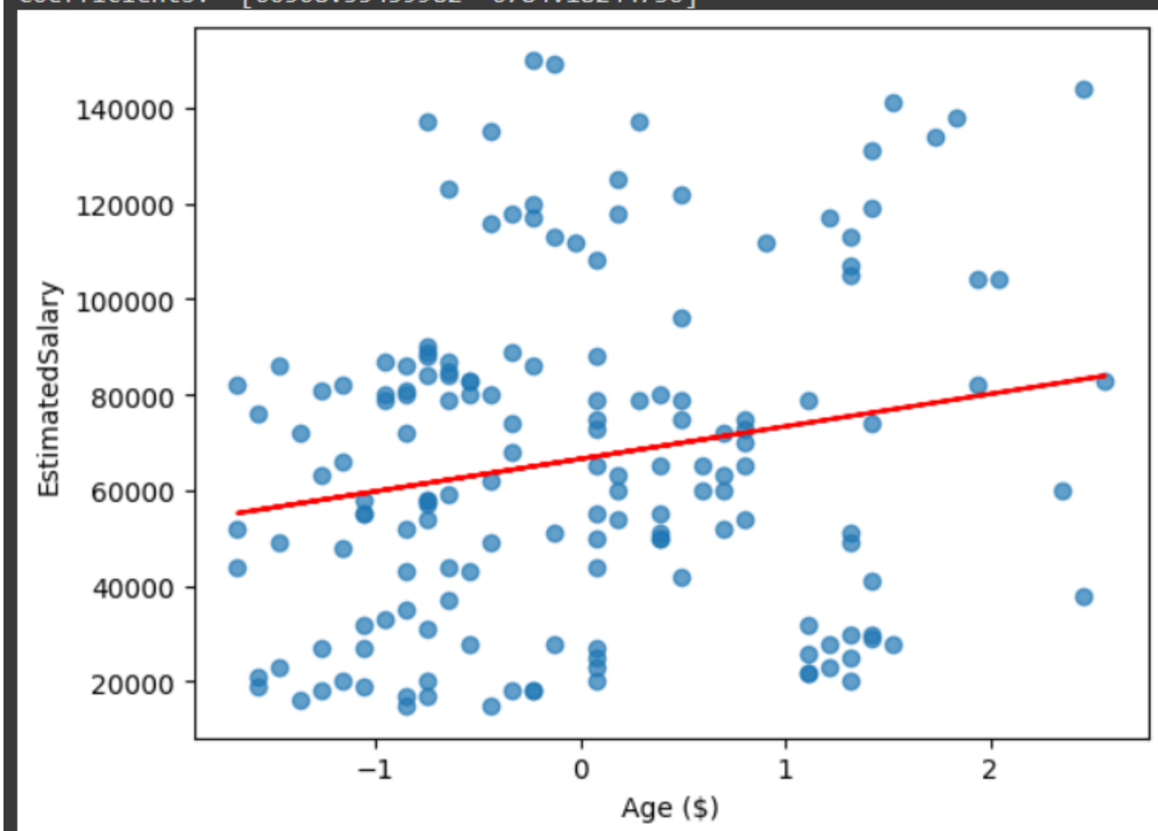
Program 9: Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Load the dataset

data = pd.read_csv("/content/drive/MyDrive/ML Lab/new1.csv")

# Select the feature (Age) and target (EstimatedSalary) variables

X = data["Age"].values.reshape(-1, 1)

y = data["EstimatedSalary"].values

X_mean = np.mean(X)

X_std = np.std(X)

X = (X-X_mean)/X_std

# Add a column of ones to X for the intercept term

X = np.c_[np.ones(X.shape[0]), X]

# Initialize the coefficients to zeros

theta = np.zeros(X.shape[1])

# Set the learning rate and number of iterations

alpha = 0.01

num_iterations = 1000

# Loop over the specified number of iterations

for i in range(num_iterations):

    # Calculate the predicted values

    y_pred = X.dot(theta)

    # Calculate the error between the predicted values and the true values

    error = y_pred - y

    # Update the coefficients using the LMS algorithm

    theta -= alpha * X.T.dot(error) / X.shape[0]

# Print the coefficients
```

```
print("Coefficients: ", theta)
# Plot the data points and the line of best fit
plt.scatter(X[:, 1], y, alpha=0.7)
plt.plot(X[:, 1], X.dot(theta), color='red')
plt.xlabel("Age ($)")
plt.ylabel("EstimatedSalary")
plt.show()
```

Output:

```
Coefficients:  [66568.55459982  6784.18244736]
```



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

```
from numpy import *
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
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))
    W = np1.linalg.pinv(X.T * wei * X) * X.T * wei * ymat.T
    return W
def localWeightRegression(xmat,ymat,k):
    m,n = np1.shape(xmat)
    ypred = np1.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred
# load data points
```

```
data = pd.read_csv('tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)
#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimensional array form
m= np1.shape(mbill)[1]
# print(m) 244 data is stored in m
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
#print(X)
#set k here
ypred = localWeightRegression(X,mtip,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: