# PRACTICAL 1

1. **Design a simple machine learning model to train the training instances and test the same.**

## CODE:

from sklearn.linear\_model import LinearRegression from random import randint

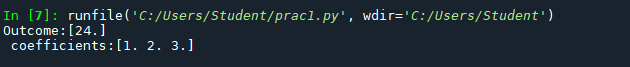
TRAIN\_limit=1000 train\_set\_count=100 train\_input=list() train\_output=list()

for i in range(train\_set\_count): a=randint(0,TRAIN\_limit) b=randint(0,TRAIN\_limit) c=randint(0,TRAIN\_limit) op=a+(2\*b)+(3\*c) train\_input.append([a,b,c]) train\_output.append(op)

predictor=LinearRegression(n\_jobs=-1) predictor.fit(X=train\_input,y=train\_output) X\_test=[[3,3,5]] outcome=predictor.predict(X=X\_test) coefficients=predictor.coef\_

print('Outcome:{}\n coefficients:{}'.format(outcome, coefficients))

## OUTPUT:



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

## CODE:

import pandas as pd import numpy as np

data=pd.read\_csv("D:/Prakash/find-s.csv") print(data,"n")

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

def train(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]:

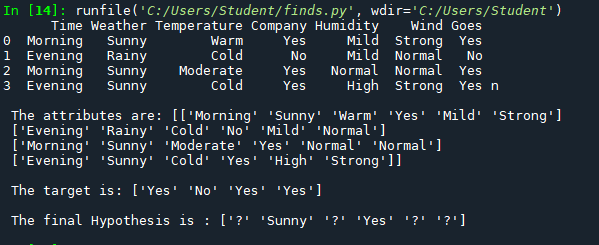
specific\_hypothesis[x]="?" else:

pass

return specific\_hypothesis

print("\n The final Hypothesis is :",train(d,target))

**OUTPUT:**



# PRACTICAL 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.

**CODE:**

from operator import ge,le import re

import pandas as pd import numpy as np

data=pd.DataFrame(data=pd.read\_csv("D:/Prakash/candidate.csv")) print(data)

concepts=np.array(data.iloc[:,0:-1]) target=np.array(data.iloc[:,-1]) print("\n",target) print("\n",concepts)

def learn(concepts,target): specific\_h=concepts[0].copy()

print("\n Initialization of Specific\_H and general\_h") print("\n specific\_h:",specific\_h)

general\_h=[["?"for i in range(len(specific\_h))]for i in range(len(specific\_h))] print("\ngeneral\_h:",general\_h)

print("\nconcepts",concepts) 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]=="?"

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("\n steps of candidate elimination algorithm:",i+1) print("\n Specific\_h:",i+1)

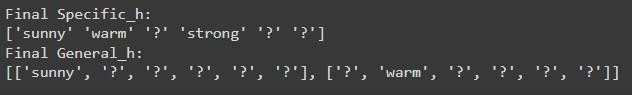
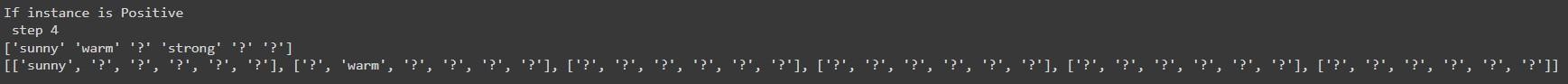
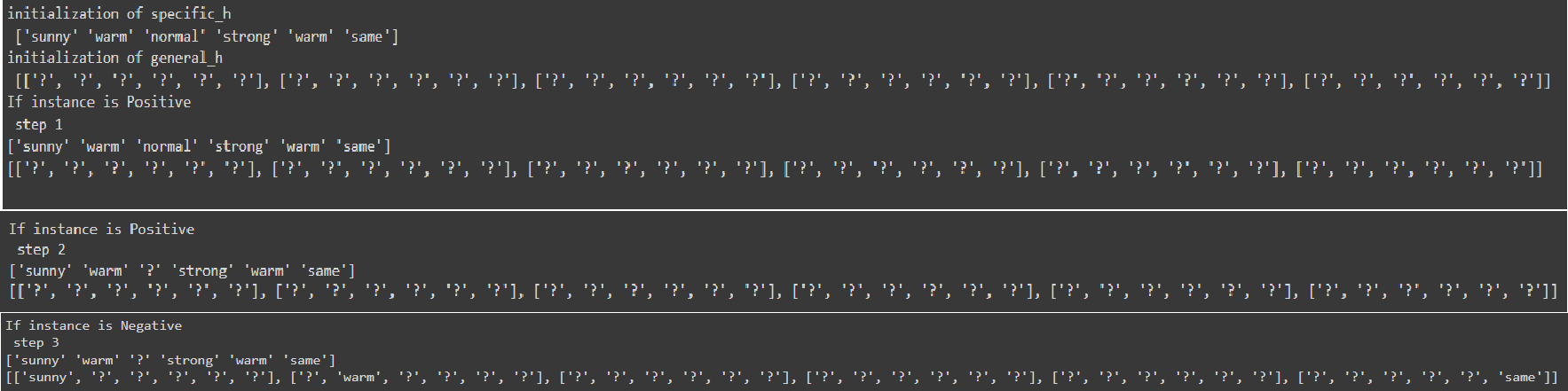
print(specific\_h,"\n") print("general\_h:",i+1) print(general\_h)

indices=[i for i, val in enumerate(general\_h) if val==["?","?","?","?","?","?"]] print("\n Indices",indices)

for i in indices: general\_h.remove(["?","?","?","?","?","?"])

return specific\_h,general\_h s\_final,g\_final=learn(concepts, target) print("\n Final specific\_h",s\_final,sep="\n")

**OUTPUT:**



**PRACTICAL 3**

# 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 numpy as nm

import matplotlib.pyplot as mtp import pandas as pd

dataset = pd.read\_csv('D:/Prakash/user\_data.csv') x = dataset.iloc[:, [1, 2]].values

y = dataset.iloc[:, 3].values

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0) from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train) x\_test = sc.transform(x\_test)

from sklearn.naive\_bayes import GaussianNB classifier = GaussianNB() classifier.fit(x\_train, y\_train)

y\_pred = classifier.predict(x\_test)

from sklearn.metrics import confusion\_matrix cm = confusion\_matrix(y\_test, y\_pred) print(cm)

from matplotlib.colors import ListedColormap x\_set, y\_set = x\_train, y\_train

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),alpha = 0.75, cmap = ListedColormap(('purple', 'green'))) mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)): mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('purple', 'green'))(i), label = j) mtp.title('Naive Bayes (Training set)')

mtp.xlabel('Age') mtp.ylabel('Estimated Salary') mtp.legend()

mtp.show()

x\_set, y\_set = x\_test, y\_test

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01), nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('purple', 'green'))) mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

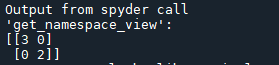
for i, j in enumerate(nm.unique(y\_set)): mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

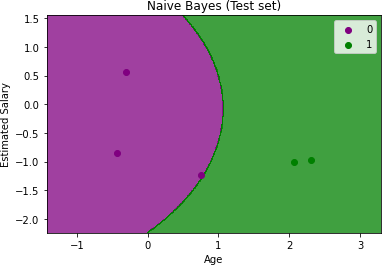
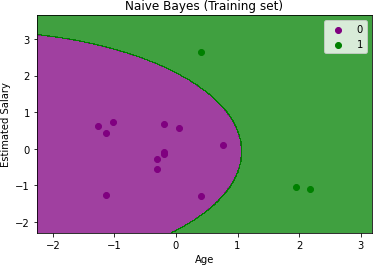
c = ListedColormap(('purple', 'green'))(i), label = j) mtp.title('Naive Bayes (Test set)')

mtp.xlabel('Age') mtp.ylabel('Estimated Salary') mtp.legend()

mtp.show()

**OUTPUT:**





# Write A Program to Implement a Decision Tree with Prediction, Test Score And Confusion Matrix.

## CODE:

import numpy as nm

import matplotlib.pyplot as mtp import pandas as pd

dataset = pd.read\_csv('D:/Prakash/user\_data.csv') x = dataset.iloc[:, [1, 2]].values

y = dataset.iloc[:, 3].values

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train) x\_test = sc.transform(x\_test)

from sklearn.tree import DecisionTreeClassifier classifier =DecisionTreeClassifier(criterion='entropy') classifier.fit(x\_train, y\_train)

DecisionTreeClassifier(class\_weight=None,criterion='entropy',max\_depth=None,max\_featur es=None,max\_leaf\_nodes=None,min\_impurity\_decrease=0.0,min\_samples\_leaf=1,min\_samp les\_split=2,min\_weight\_fraction\_leaf=0.0,random\_state=0,splitter='best')

y\_pred = classifier.predict(x\_test)

from sklearn.metrics import confusion\_matrix cm = confusion\_matrix(y\_test, y\_pred) print(cm)

from sklearn.metrics import accuracy\_score sc1=accuracy\_score(y\_test, y\_pred)\*100 print("Accuracy",sc1)

from matplotlib.colors import ListedColormap x\_set, y\_set = x\_train, y\_train

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),alpha = 0.75, cmap = ListedColormap(('purple', 'green'))) mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)): mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('purple', 'green'))(i), label = j) mtp.title('Decision Tree Algorithm (Training set)') mtp.xlabel('Age')

mtp.ylabel('Estimated Salary') mtp.legend()

mtp.show()

x\_set, y\_set = x\_test, y\_test

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('purple', 'green'))) mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)): mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

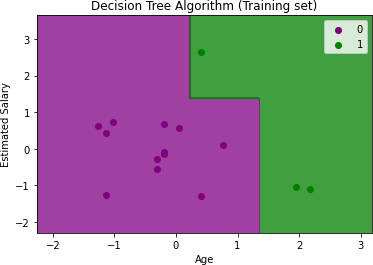
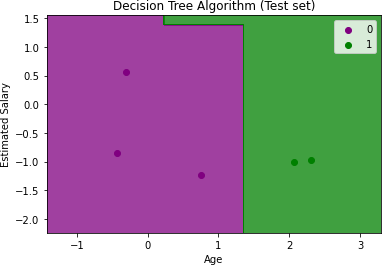
c = ListedColormap(('purple', 'green'))(i), label = j) mtp.title('Decision Tree Algorithm (Test set)') mtp.xlabel('Age')

mtp.ylabel('Estimated Salary') mtp.legend()

mtp.show()

## OUTPUT:



# Write a Program to Implement Random Forest with Prediction, Test Score and Confusion Matrix.

## CODE:

import numpy as nm import pandas as pd

import matplotlib.pyplot as mtp data\_set=pd.read\_csv('D:/Prakash/Social\_Network\_Ads.csv') x=data\_set.iloc[:,[2,3]].values

y = data\_set.iloc[:, 4].values

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler st\_x = StandardScaler()

x\_train = st\_x.fit\_transform(x\_train) x\_test = st\_x.transform(x\_test)

from sklearn.ensemble import RandomForestClassifier

classifier =RandomForestClassifier(n\_estimators=10,criterion='entropy') classifier.fit(x\_train, y\_train)

y\_pred=classifier.predict(x\_test) print(y\_pred)

from sklearn.metrics import confusion\_matrix cm = confusion\_matrix(y\_test, y\_pred) print(cm)

from sklearn.metrics import accuracy\_score sc1=accuracy\_score(y\_test, y\_pred)\*100 print("Accuracy",sc1)

from matplotlib.colors import ListedColormap x\_set, y\_set = x\_train, y\_train

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),alpha = 0.75, cmap = ListedColormap(('purple', 'green'))) mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)): mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('purple', 'green'))(i), label = j) mtp.title('RandomForest Algorithm (Training set)') mtp.xlabel('Age')

mtp.ylabel('Estimated Salary') mtp.legend()

mtp.show()

x\_set, y\_set = x\_test, y\_test

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('purple', 'green'))) mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

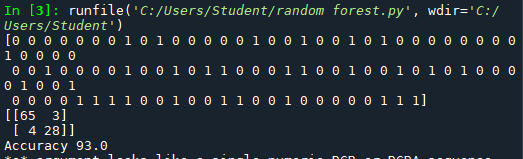
for i, j in enumerate(nm.unique(y\_set)): mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

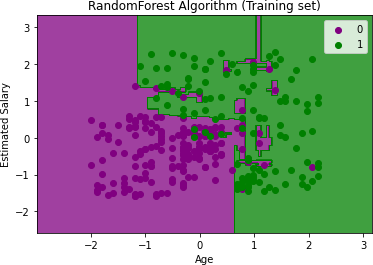
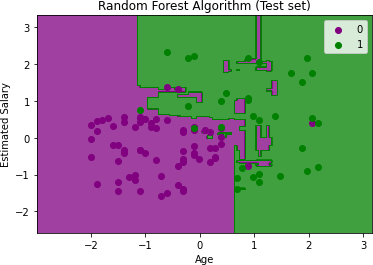
c = ListedColormap(('purple', 'green'))(i), label = j) mtp.title('Random Forest Algorithm (Test set)') mtp.xlabel('Age')

mtp.ylabel('Estimated Salary') mtp.legend()

mtp.show()

## OUTPUT:



# PRACTICAL 4

1. **For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm.**

## CODE:

import numpy as np import pandas as pd

import matplotlib.pyplot as plt data=pd.read\_csv('D:/Prakash/sale.csv') print(data.shape)

(6,2)

print(data.head())

X=data['Price of T-shirts in dollars(x)'].values Y=data['# of T - Shirts Sold(y)'].values mean\_x=np.mean(X)

mean\_y=np.mean(Y) n=len(X)

numer=0 denom=0

for i in range(n):

numer+=(X[i]-mean\_x)\*(Y[i]-mean\_y) denom+=(X[i]-mean\_x)\*\*2 m=numer/denom

c=mean\_y-(m\*mean\_x) print('Coefficients') print(m,c) max\_x=np.max(X)+100 min\_x=np.min(X)-100

x=np.linspace(min\_x,max\_x,1000) y=m\*x+c

plt.plot(x,y,color='#58b970',label="Regression Line") plt.scatter(X, Y, c='#ef5423',label='Scatter Plot') plt.xlabel('Price of T-shirts in Dollars(x)') plt.ylabel("# of T-shirts Sold(y)")

plt.legend() plt.show() rmse=0

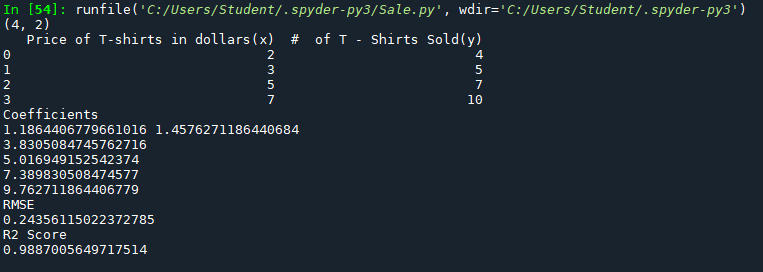
for i in range(n): y\_pred=c+m\*X[i] print(y\_pred) rmse+=(Y[i]-y\_pred)\*\*2

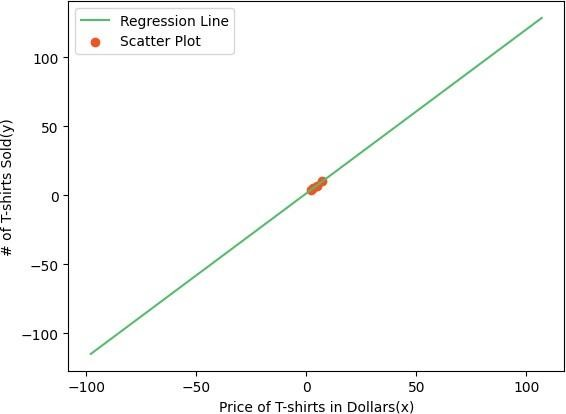
rmse=np.sqrt(rmse/n) print(rmse)

ss\_tot=0 ss\_res=0

for i in range(n): y\_pred=c+m\*X[i] ss\_tot+=(Y[i]-mean\_y)\*\*2 ss\_res+=(Y[i]-y\_pred)\*\*2

r2=1-(ss\_res/ss\_tot) print("R2 SCORE :",r2)

**OUTPUT:**



# For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm.

## CODE:

import numpy as nm

import matplotlib.pyplot as mtp import pandas as pd

data\_set=pd.read\_csv("D:/Vinish/Social\_Network\_Ads.csv") x=data\_set.iloc[:,[2,3]].values

y=data\_set.iloc[:,4].values

from sklearn.model\_selection import train\_test\_split x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.20,random\_state=0) from sklearn.preprocessing import StandardScaler

sc=StandardScaler() x\_train=sc.fit\_transform(x\_train) x\_test=sc.transform(x\_test)

from sklearn.linear\_model import LogisticRegression classifier=LogisticRegression(random\_state=0) classifier.fit(x\_train,y\_train) y\_pred=classifier.predict(x\_test)

from sklearn.metrics import confusion\_matrix cm=confusion\_matrix(y\_test, y\_pred) print(cm)

from sklearn.metrics import accuracy\_score sc1=accuracy\_score(y\_test,y\_pred)\*100 print("Accuracy",sc1)

from sklearn.metrics import classification\_report creport=classification\_report(y\_test,y\_pred) print(creport)

from matplotlib.colors import ListedColormap x\_set, y\_set = x\_train, y\_train

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),alpha = 0.75, cmap = ListedColormap(('red', 'green'))) mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)): mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j) mtp.title('logistic Regression (Training set)')

mtp.xlabel('Age') mtp.ylabel('Estimated Salary') mtp.legend()

mtp.show()

x\_set, y\_set = x\_test, y\_test

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green'))) mtp.xlim(X1.min(), X1.max())

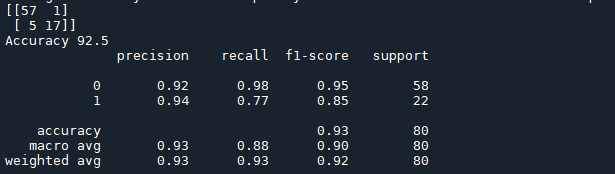
mtp.ylim(X2.min(), X2.max())

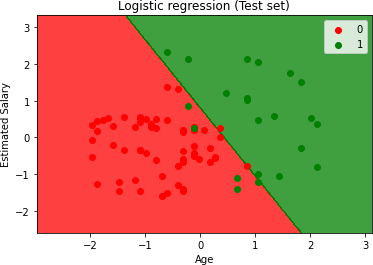
for i, j in enumerate(nm.unique(y\_set)): mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j) mtp.title('Logistic regression (Test set)') mtp.xlabel('Age')

mtp.ylabel('Estimated Salary') mtp.legend()

mtp.show()

**OUTPUT:**



# PRACTICAL 5

## 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 numpy as nm

import matplotlib.pyplot as mtp import pandas as pd

dataset = pd.read\_csv('D:/Prakash/user\_data.csv') x = dataset.iloc[:, [1, 2]].values

y = dataset.iloc[:, 3].values

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler st\_x = StandardScaler()

x\_train = st\_x.fit\_transform(x\_train) x\_test = st\_x.transform(x\_test)

from sklearn.tree import DecisionTreeClassifier classifier=DecisionTreeClassifier(criterion='entropy',random\_state=0) classifier.fit(x\_train,y\_train) DecisionTreeClassifier(class\_weight=None,criterion='entropy',max\_depth=None,max\_featur es=None,max\_leaf\_nodes=None,min\_impurity\_decrease=0.0,min\_samples\_leaf=1,min\_samp les\_split=2,min\_weight\_fraction\_leaf=0.0,random\_state=0,splitter='best') y\_pred=classifier.predict(x\_test)

from sklearn.metrics import confusion\_matrix cm=confusion\_matrix(y\_test,y\_pred) print(cm)

from sklearn.metrics import accuracy\_score sc1=accuracy\_score(y\_test,y\_pred)\*100 print("Acuracy Score: ",sc1)

from matplotlib.colors import ListedColormap x\_set, y\_set = x\_train, y\_train

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),alpha = 0.75, cmap = ListedColormap(('red', 'green'))) mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)): mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j) mtp.title('Decision Tree Algorithm (Training set)') mtp.xlabel('Age')

mtp.ylabel('Estimated Salary') mtp.legend()

mtp.show()

x\_set, y\_set = x\_test, y\_test

X1, X2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01)) mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green'))) mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)): mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j) mtp.title('Decision Tree Algorithm (Test set)') mtp.xlabel('Age')

mtp.ylabel('Estimated Salary') mtp.legend()

mtp.show()

## OUTPUT:

1. **Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set.**

## CODE:

import numpy as np import pandas as pd

import matplotlib.pyplot as mtp

import re as T data\_set=pd.read\_csv('D:/Vinish/Social\_Network\_Ads.csv') x=data\_set.iloc[:,[2,3]].values

y = data\_set.iloc[:, 4].values

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0) from sklearn.model\_selection import train\_test\_split x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.20,random\_state=0) print("len",len(y\_test))

from sklearn.preprocessing import StandardScaler st\_x = StandardScaler()

x\_train = st\_x.fit\_transform(x\_train) x\_test = st\_x.transform(x\_test) print(x\_train)

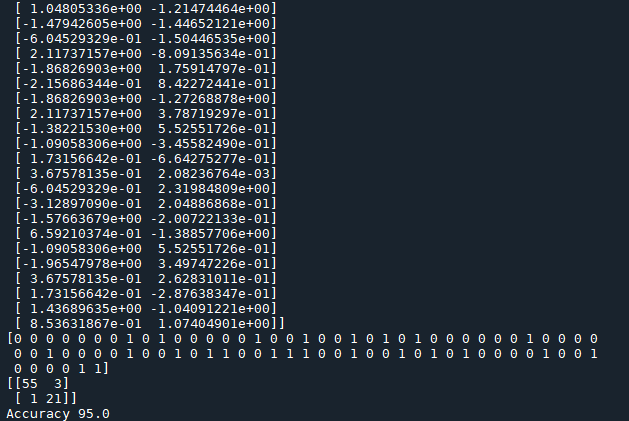
print(x\_test)

from sklearn.neighbors import KNeighborsClassifier classifier=KNeighborsClassifier(n\_neighbors=5,metric="minkowski",p=2) classifier.fit(x\_train,y\_train)

y\_pred=classifier.predict(x\_test) print(y\_pred)

from sklearn.metrics import confusion\_matrix cm = confusion\_matrix(y\_test, y\_pred) print(cm)

from sklearn.metrics import accuracy\_score sc1=accuracy\_score(y\_test, y\_pred)\*100 print("Accuracy",sc1)

**OUTPUT:**

**PRACTICAL 6**

# Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix.

## CODE:

import numpy as np

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix, accuracy\_score

wine = datasets.load\_wine()

X = wine.data y = wine.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42) knn = KNeighborsClassifier(n\_neighbors=5, metric='euclidean')

knn.fit(X\_train, y\_train) y\_pred = knn.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred) print(f"Accuracy: {accuracy:.2f}")

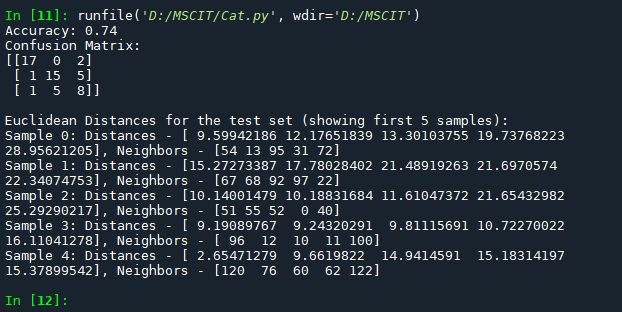
conf\_matrix = confusion\_matrix(y\_test, y\_pred) print("Confusion Matrix:")

print(conf\_matrix)

distances, indices = knn.kneighbors(X\_test)

print("\nEuclidean Distances for the test set (showing first 5 samples):") for i in range(5):

print(f"Sample {i}: Distances - {distances[i]}, Neighbors - {indices[i]}")

**OUTPUT:**

# Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix.

## CODE:

from sklearn.cluster import KMeans import matplotlib.pyplot as mtp

import pandas as pd

data\_set = pd.read\_csv('D:/Prakash/Mall\_Customers.csv') x = data\_set.iloc[:, [3, 4]].values

wcss\_list = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state=42) kmeans.fit(x)

wcss\_list.append(kmeans.inertia\_) mtp.plot(range(1, 11), wcss\_list) mtp.title("The Elbow Method Graph") mtp.xlabel("Number of Clusters(k)") mtp.ylabel('wcss\_list')

mtp.show()

kmeans = KMeans(n\_clusters=5, init='k-means++', random\_state=42) y\_predict = kmeans.fit\_predict(x)

print(y\_predict)

mtp.scatter(x[y\_predict == 0, 0], x[y\_predict == 0, 1],s=100, c='blue', label='Cluster 1')

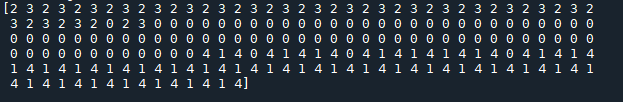
mtp.scatter(x[y\_predict == 1, 0], x[y\_predict == 1, 1],s=100, c='green', label='Cluster 2')

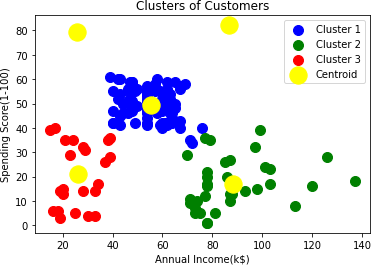
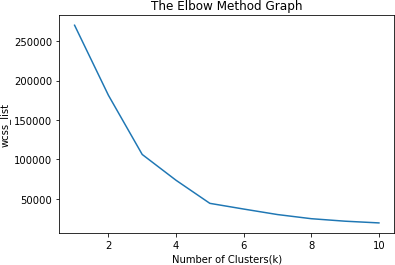
mtp.scatter(x[y\_predict == 2, 0], x[y\_predict == 2, 1],s=100, c='red', label='Cluster 3') mtp.scatter(kmeans.cluster\_centers\_[:,0],kmeans.cluster\_centers\_[:,1],s=300,c='yellow',label

="Centroid")

mtp.title('Clusters of Customers') mtp.xlabel('Annual Income(k$)') mtp.ylabel('Spending Score(1-100)') mtp.legend()

mtp.show()

**OUTPUT:**



# PRACTICAL 7

1. **Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix.**

## CODE:

import numpy as np

import matplotlib.pyplot as mtp import pandas as pd

dataset=pd.read\_csv('D:/Prakash/Mall\_Customers.csv') x = dataset.iloc[:, [3, 4]].values

import scipy.cluster.hierarchy as she dendro=she.dendrogram(she.linkage(x,method="ward")) mtp.title('Dendogram Plot')

mtp.ylabel('Euclidean Distances') mtp.xlabel('Customers') mtp.show()

from sklearn.cluster import AgglomerativeClustering hc=AgglomerativeClustering(n\_clusters=5,affinity='euclidean',linkage='ward') y\_pred=hc.fit\_predict(x)

print(y\_pred)

mtp.scatter(x[y\_pred == 0, 0], x[y\_pred == 0, 1],s=100, c='blue', label='Cluster 1')

mtp.scatter(x[y\_pred == 1, 0], x[y\_pred == 1, 1],s=100, c='green', label='Cluster 2')

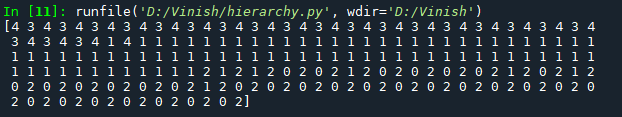
mtp.scatter(x[y\_pred == 2, 0], x[y\_pred == 2, 1],s=100, c='red', label='Cluster 3')

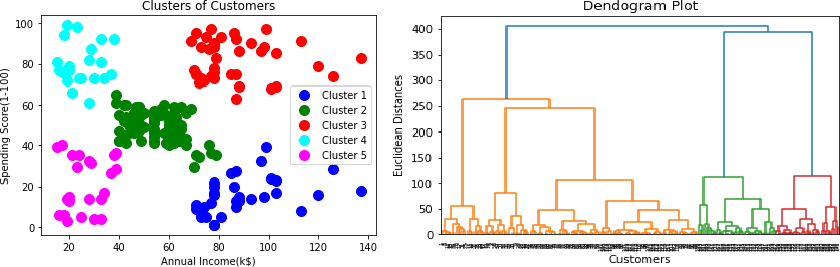
mtp.scatter(x[y\_pred == 3, 0], x[y\_pred == 3, 1],s=100, c='cyan', label='Cluster 4')

mtp.scatter(x[y\_pred == 4, 0], x[y\_pred == 4, 1],s=100, c='magenta', label='Cluster 5') mtp.title('Clusters of Customers')

mtp.xlabel('Annual Income(k$)') mtp.ylabel('Spending Score(1-100)') mtp.legend()

mtp.show()

**OUTPUT:**



# Implement the Rule based method and test the same.

## CODE:

import numpy as np import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score from sklearn.metrics import confusion\_matrix

mail\_data = pd.read\_csv(r"/content/mail\_data.csv") print(mail\_data)

data = mail\_data.where((pd.notnull(mail\_data)), '')

data.loc[data['Category'] == 'spam', 'Category',] = 0 data.loc[data['Category'] == 'ham', 'Category',] = 1 data.head()

X = data['Message'] y = data['Category'] print(X)

print(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 3) print(X.shape)

print(X\_train.shape) print(X\_test.shape) print(y.shape) print(y\_train.shape) print(y\_test.shape)

feature\_extraction = TfidfVectorizer(min\_df = 1, stop\_words = 'english', lowercase = True)

X\_train\_features = feature\_extraction.fit\_transform(X\_train) X\_test\_features = feature\_extraction.transform(X\_test)

# convert the targets into integers y\_train = y\_train.astype('int') y\_test = y\_test.astype('int') print(X\_train\_features)

model = LogisticRegression() model.fit(X\_train\_features, y\_train)

prediction\_on\_training\_data = model.predict(X\_train\_features) accuracy\_on\_training\_data = accuracy\_score(y\_train, prediction\_on\_training\_data) print('Accuracy on training data :', accuracy\_on\_training\_data)

prediction\_on\_test\_data = model.predict(X\_test\_features) accuracy\_on\_test\_data = accuracy\_score(y\_test, prediction\_on\_test\_data) print('Accuracy on test data :', accuracy\_on\_test\_data)

print('Confusion Matrix \n',confusion\_matrix(y\_test,prediction\_on\_test\_data ))

# building a predective system

input\_mail = ["XXXMobileMovieClub: To use your credit, click the WAP link in the next txt message or click here>> [http://wap](http://wap/). [xxxmobilemovieclub.com?n=QJKGIGHJJGCBL](http://xxxmobilemovieclub.com/?n=QJKGIGHJJGCBL)"]

# convert text into feature vectors

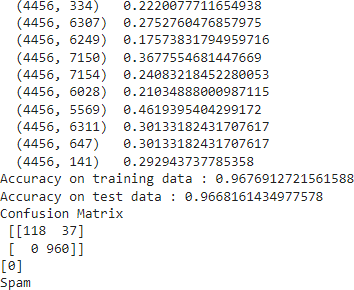
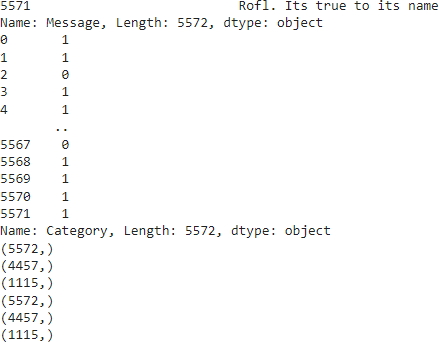
input\_data\_features = feature\_extraction.transform(input\_mail) # make predictions

prediction = model.predict(input\_data\_features) print(prediction)

if prediction[0] == 1 : print('Ham')

else :

print('Spam')

**OUTPUT:**

# PRACTICAL 8

1. **Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.**

## Code:

import numpy as np

x=np.array(([2,9],[1,5],[3,6]),dtype=float)

y=np.array(([92],[86],[89]),dtype=float) x=x/np.amax(x,axis=0)

y=y/100

def sigmoid(x):

return 1/(1+np.exp(-x)) def der\_sigmoid(x):

return x\*(1-x) epoch=5

lr=0.1 inputlayer\_neurons=2 hiddenlayer\_neurons=3 output\_neurons=1

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons)) bh=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons)) wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons)) bout=np.random.uniform(size=(output\_neurons))

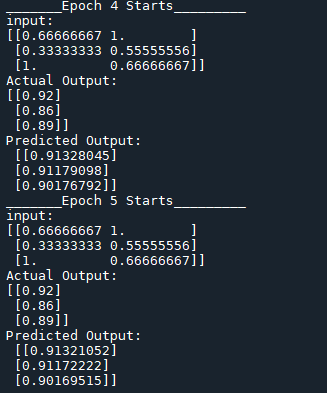
for i in range(epoch): hinp1=np.dot(x,wh) hinp=hinp1+bh hlayer\_act=sigmoid(hinp) outinp1=np.dot(hlayer\_act,wout) outinp=outinp1+bout output=sigmoid(outinp)

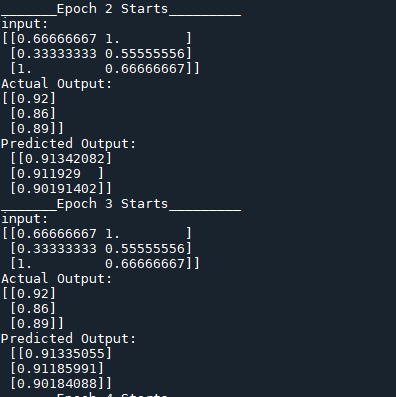
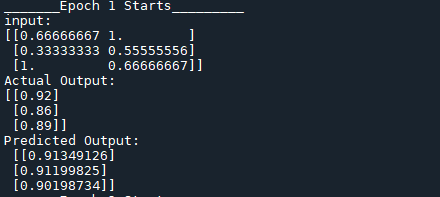
EO=y-output outgrad=der\_sigmoid(output) d\_output=EO\*outgrad EH=d\_output.dot(wout.T) hiddengrad=der\_sigmoid(hlayer\_act) d\_hiddenlayer=EH\*hiddengrad wout+=hlayer\_act.T.dot(d\_output)\*lr wh+=x.T.dot(d\_hiddenlayer)\*lr

print(" Epoch",i+1,"Starts ") print("input:\n"+str(x))

print("Actual Output:\n"+str(y)) print("Predicted Output:\n",output)

**OUTPUT:**





# Assuming a set of documents that need to be classified, use the naive Bayesian Classifier model to perform this task.

## CODE:

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB from sklearn import metrics

data=pd.read\_csv('D:/Prakash/naivebayes.csv',names=['text','label']) print("\n Dataset is :\n",data)

print("Dimension of the dataset",data.shape) data['labelnum']=data.label.map({'Positive':1,'Negative':0}) x=data.text

y=data.labelnum print(x)

print(y)

vectorizer=TfidfVectorizer() data=vectorizer.fit\_transform(x)

print('\n The tf-idf features of Dataset:\n') df=pd.DataFrame(data.toarray(),columns=vectorizer.get\_feature\_names()) df.head() x\_train,x\_test,y\_train,y\_test=train\_test\_split(data,y,test\_size=0.3,random\_state=2)

print("\n the total number of training Data:",y\_train.shape)

print("\n the total number of test Data:",y\_test.shape)

clf=MultinomialNB().fit(x\_train,y\_train) predicted=clf.predict(x\_test)

print("\n Accuracy of Classifier:",metrics.accuracy\_score(y\_test, predicted)) print("Confusion matrix:",metrics.confusion\_matrix(y\_test, predicted)) print("Classification report",metrics.classification\_report(y\_test,predicted)) print("\n the value of precision",metrics.precision\_score(y\_test,predicted)) print("\n The value of recall:",metrics.recall\_score(y\_test,predicted))

**OUTPUT:**

