**Q1. Implement and demonstrate the FIND-S algorithm for findingthe most specific hypothesis based on a given set of training data samples. Read thetraining data**

from a .CSV file. (use enjoysport.csv)

import pandas as pd

import numpy as np

df = pd.read\_excel('enjoysport.xlsx')

df

def find\_s(data):

hypothesis = ['0'] \* (len(data.columns) - 1) # Initialize the

hypothesis as the most general hypothesis

for \_, example in data.iterrows():

if example.iloc[-1] == 'yes':

for i in range(len(example) - 1):

if hypothesis[i] == '0':

hypothesis[i] = example[i]

elif hypothesis[i] != example[i]:

hypothesis[i] = '?'

return hypothesis

hypothesis = find\_s(df)

print('Final hypothesis:', hypothesis)

**Q2. Implement email spam classification using naive Bayes algorithm.**

import pandas as pd

df = pd.read\_csv('spam.csv')

df

df.groupby('Category').describe()

df['spam'] = df['Category'].apply(lambda x: 1 if x == 'spam' else 0)

df.head()

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

X\_train, X\_test, y\_train, y\_test =

train\_test\_split(df.Message,df.spam,test\_size=0.20)

from sklearn.feature\_extraction.text import CountVectorizer

v= CountVectorizer()

X\_train\_count = v.fit\_transform(X\_train.values)

X\_train\_count.toarray()

from sklearn.naive\_bayes import MultinomialNB

model = MultinomialNB()

model.fit(X\_train\_count,y\_train)

emails = [

'Hey mohan, can we get together to watch football game

tomorrow?',

'Upto 20% discount on parking, exclusive offer just for you. Dont

miss this reward!'

]

emails\_count = v.transform(emails)

emails\_count.toarray()

model.predict(emails\_count)

x\_test\_count = v.transform(X\_test)

model.score(x\_test\_count,y\_test)

from sklearn.pipeline import Pipeline

clf = Pipeline([

('vectorizer', CountVectorizer()),

('nb', MultinomialNB())

])

clf.fit(X\_train,y\_train)

clf.score(X\_test,y\_test)

clf.predict(emails)

**Q3. Implement Linear regression to predict house prices using (i)Least squared method (ii) Normal equations.**

(use homeprice\_uni.csv)

import pandas as pd

import numpy as np

df = pd.read\_excel('homeprices\_uni.xlsx')

df

from matplotlib import pyplot as plt

from sklearn import linear\_model

%matplotlib inline

plt.xlabel('area')

plt.ylabel('price')

plt.scatter(df.area, df.price, color="blue",marker='+')

new\_df = df.drop('price', axis='columns')

model = linear\_model.LinearRegression()

model.fit(new\_df, df.price)

model.predict([[3300]])

model.coef\_

model.intercept\_

price = model.intercept\_ + model.coef\_\*3300

price

**Q3. Implement Linear regression to predict house prices usinggradient descent algorithm. (use homeprice\_uni.csv) LINEAR REGRESSION MULTIPLE VARIABLE**

import pandas as pd

import numpy as np

df = pd.read\_excel('homeprices\_multivariate.xlsx')

df

df.tail()

df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())

df

from sklearn.linear\_model import LinearRegression

mulreg = LinearRegression()

new\_df = df.drop('price', axis='columns')

new\_df

mulreg.fit(new\_df, df.price)

print("Weight: ",mulreg.coef\_," Intercept: ",mulreg.intercept\_)

mulreg.predict([[3000, 3, 40]])

price = mulreg.intercept\_ + mulreg.coef\_[0]\*3000 +

mulreg.coef\_[1]\*3 + mulreg.coef\_[2]\*40

print("the predicted price is: ",price)

**Q4. Implement Linear regression to predict house prices based onmultiple variables. homeprice\_multivariate. csv)**

from sklearn.datasets import fetch\_openml

boston = fetch\_openml('boston',version=1)

boston

boston.feature\_names

boston.target\_names

import pandas as pd

df = pd.DataFrame(boston.data, columns=boston.feature\_names)

import matplotlib.pyplot as plt

%matplotlib inline

plt.scatter(df['RM'], boston.target)

plt.xlabel('RM')

plt.ylabel('Price')

plt.title('Price vs RM')

df

df.drop(['CHAS'], axis=1, inplace=True)

df

df[boston.target\_names[0]] = boston.target

df

df.isna().any()

X = df.drop(boston.target\_names[0], axis=1)

y = df[boston.target\_names[0]]

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,

random\_state=0)

x\_train = x\_train.apply(pd.to\_numeric, errors='coerce')

y\_train = y\_train.apply(pd.to\_numeric, errors='coerce')

x\_test = x\_test.apply(pd.to\_numeric, errors='coerce')

y\_test = y\_test.apply(pd.to\_numeric, errors='coerce')

x\_train = x\_train.fillna(0)

y\_train = y\_train.fillna(0)

x\_test = x\_test.fillna(0)

y\_test = y\_test.fillna(0)

L1 Regularized: Lasso And L2 Regularized: Ridge

from sklearn.linear\_model import Lasso, Ridge

model\_lasso = Lasso(alpha=0.1,max\_iter=100,tol=0.1)

model\_ridge = Ridge(alpha=0.1,max\_iter=100,tol=0.1)

model\_lasso.fit(x\_train, y\_train)

model\_ridge.fit(x\_train, y\_train)

model\_lasso.score(x\_train, y\_train)

model\_lasso.score(x\_test, y\_test)

model\_ridge.score(x\_train, y\_train)

model\_ridge.score(x\_test, y\_test)

Normal Linear Regression

from sklearn.linear\_model import LinearRegression

model\_lr = LinearRegression().fit(x\_train, y\_train)

model\_lr.score(x\_train, y\_train)

model\_lr.score(x\_test, y\_test)

**Q5. Implement Linear regression to predict house prices based on multiple variables regularization techniques and explain how regularisation overcome overfitting problem. (use inbuilt dataset boston- from sklearn.datasets import load\_boston)**

from sklearn.naive\_bayes import MultinomialNB, GaussianNB

import pandas as pd

import numpy as np

df = pd.read\_csv('titanic.csv')

df

df.drop(['Name', 'SibSp', 'Parch', 'Ticket', 'Cabin',

'Embarked','PassengerId','Fare'], axis='columns', inplace=True)

df

df.Age = df.Age.fillna(df.Age.mean())

df

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

df['female'] = df['Sex'].apply(lambda x: 1 if x == "female" else 0)

df

df.drop(['Sex'], axis='columns', inplace=True)

df

model = MultinomialNB()

model2 = GaussianNB()

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(df.drop(['Survived'],

axis='columns'),df['Survived'],test\_size = 0.20)

model.fit(X\_train, Y\_train)

model2.fit(X\_train, Y\_train)

model.score(X\_test, Y\_test)

model2.score(X\_test, Y\_test)

model.predict(X\_test)

model2.predict(X\_test)

X\_test

**Q6. Implement titanic survival prediction using Naive Bayes algorithm.**

import pandas as pd

import numpy as np

df = pd.read\_excel('insurance\_data.xlsx')

df

import matplotlib.pyplot as plt

%matplotlib inline

plt.scatter(df['age'], df['bought\_insurance'], color='blue',marker='+')

plt.xlabel('Age')

plt.ylabel('Insurance')

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df[['age']],

df.bought\_insurance, test\_size=0.2)

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(X\_train, y\_train)

model.score(X\_test, y\_test)

Y\_pred = model.predict(X\_test)

X\_test

y\_test

Y\_pred

**Q7. Predict a person would buy life insurance based on his age using**

**logistic regression. (insurance\_data.csv)**

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.datasets import mnist

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train\_flat = x\_train.reshape(x\_train.shape[0], -1)

x\_test\_flat = x\_test.reshape(x\_test.shape[0], -1)

scaler = StandardScaler()

x\_train\_scaled = scaler.fit\_transform(x\_train\_flat)

# Logistic Regression model for classification

lr\_model = LogisticRegression(multi\_class='multinomial',

solver='lbfgs')

lr\_model.fit(x\_train\_scaled, y\_train)

# Define neural network architecture

nn\_model = Sequential()

nn\_model.add(Dense(128, activation='relu', input\_shape=(784,))) #

Input layer with 784 features

nn\_model.add(Dense(64, activation='relu')) # Hidden layer with 64

neurons

nn\_model.add(Dense(10, activation='softmax')) # Output layer with

10 neurons (one for each digit)

# Compile the model

nn\_model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

nn\_model.fit(x\_train\_flat, y\_train, epochs=10, batch\_size=32,

validation\_data=(x\_test\_flat, y\_test))

# Evaluate Logistic Regression model

lr\_score = lr\_model.score(x\_test\_flat, y\_test)

print("Logistic Regression Accuracy:", lr\_score)

# Evaluate Neural Network model

nn\_loss, nn\_accuracy = nn\_model.evaluate(x\_test\_flat, y\_test)

print("Neural Network Accuracy:", nn\_accuracy)

**Q8. Implement neural network or Logistic regression to recognise hand writen digits.**

import pandas as pd

import numpy as np

df = pd.read\_csv('income.csv')

df

import matplotlib.pyplot as plt

%matplotlib inline

plt.scatter(df['Age'], df['Income($)'])

xlabel = 'Age'

ylabel = 'Income($)'

plt.xlabel(xlabel)

plt.ylabel(ylabel)

from sklearn.cluster import KMeans

from sklearn.preprocessing import MinMaxScaler

km = KMeans(n\_clusters=3)

y\_predicted = km.fit\_predict(df[['Age', 'Income($)']])

y\_predicted

df['Cluster'] = y\_predicted

df.head()

km.cluster\_centers\_

df1 = df[df.Cluster==1]

df2 = df[df.Cluster==2]

df0 = df[df.Cluster==0]

plt.scatter(df0.Age, df0['Income($)'], label = 'Cluster 0',color='black')

plt.scatter(df1.Age, df1['Income($)'], color='green', label = 'Cluster

1')

plt.scatter(df2.Age, df2['Income($)'], color='red', label = 'Cluster 2')

plt.scatter(km.cluster\_centers\_[:,0],km.cluster\_centers\_[:,1],color='pu

rple',marker='\*',label='centroid')

plt.xlabel(xlabel)

plt.ylabel(ylabel)

plt.legend()

Scaler = MinMaxScaler()

Scaler.fit(df[['Income($)']])

df['Income($)'] = Scaler.transform(df[['Income($)']])

Scaler.fit(df[['Age']])

df['Age'] = Scaler.transform(df[['Age']])

df.head()

plt.scatter(df.Age,df['Income($)'])

km = KMeans(n\_clusters=3)

y\_predicted = km.fit\_predict(df[['Age','Income($)']])

y\_predicted

df['Cluster']=y\_predicted

df.head()

km.cluster\_centers\_

df1 = df[df.Cluster==0]

df2 = df[df.Cluster==1]

df3 = df[df.Cluster==2]

plt.scatter(df1.Age,df1['Income($)'],color='green')

plt.scatter(df2.Age,df2['Income($)'],color='red')

plt.scatter(df3.Age,df3['Income($)'],color='black')

plt.scatter(km.cluster\_centers\_[:,0],km.cluster\_centers\_[:,1],color='pu

rple',marker='\*',label='centroid')

plt.legend()

sse = []

k\_rng = range(1,10)

for k in k\_rng:

km = KMeans(n\_clusters=k)

km.fit(df[['Age','Income($)']])

sse.append(km.inertia\_)

plt.xlabel('K')

plt.ylabel('Sum of squared error')

plt.plot(k\_rng,sse)

**Q9. Implement K-means clustring.**

import pandas as pd

import numpy as np

def entropy(target\_col):

elements,counts = np.unique(target\_col,return\_counts = True)

entropy = np.sum([(-

counts[i]/np.sum(counts))\*np.log2(counts[i]/np.sum(counts)) for i in

range(len(elements))])

return entropy

def InfoGain(data,split\_attribute\_name,target\_name="class"):

total\_entropy = entropy(data[target\_name])

vals,counts=

np.unique(data[split\_attribute\_name],return\_counts=True)

Weighted\_Entropy =

np.sum([(counts[i]/np.sum(counts))\*entropy(data.where(data[split\_attri

bute\_name]==vals[i]).dropna()[target\_name]) for i in range(len(vals))])

Information\_Gain = total\_entropy - Weighted\_Entropy

return Information\_Gain

def

ID3(data,originaldata,features,target\_attribute\_name="class",parent\_no

de\_class = None):

if len(np.unique(data[target\_attribute\_name])) <= 1:

return np.unique(data[target\_attribute\_name])[0]

elif len(data)==0:

return

np.unique(originaldata[target\_attribute\_name])[np.argmax(np.unique(o

riginaldata[target\_attribute\_name],return\_counts=True)[1])]

elif len(features) ==0:

return parent\_node\_class

else:

parent\_node\_class =

np.unique(data[target\_attribute\_name])[np.argmax(np.unique(data[targ

et\_attribute\_name],return\_counts=True)[1])]

item\_values = [InfoGain(data,feature,target\_attribute\_name) for

feature in features]

best\_feature\_index = np.argmax(item\_values)

best\_feature = features[best\_feature\_index]

tree = {best\_feature:{}}

features = [i for i in features if i != best\_feature]

for value in np.unique(data[best\_feature]):

value = value

sub\_data = data.where(data[best\_feature] == value).dropna()

subtree =

ID3(sub\_data,originaldata,features,target\_attribute\_name,parent\_node\_

class)

tree[best\_feature][value] = subtree

return(tree)

def predict(query,tree,default = 1):

for key in list(query.keys()):

if key in list(tree.keys()):

try:

result = tree[key][query[key]]

except:

return default

result = tree[key][query[key]]

if isinstance(result,dict):

return predict(query,result)

else:

return result

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

from sklearn import datasets

iris = datasets.load\_iris()

iris

df = pd.DataFrame(data = np.c\_[iris['data'], iris['target']],columns=

iris['feature\_names'] + ['target'])

df

train, test = train\_test\_split(df, test\_size = 0.2)

features = train.columns[:-1]

features

tree = ID3(train,train,features,'target')

query = test.iloc[0,:].to\_dict()

query.pop('target')

prediction = predict(query,tree,1)

print('The predicted class is:', prediction)

print('The actual class is:', test.iloc[0, -1])

query

**Q10. Implement decision tree using ID3 algorithm.**

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

import numpy as np

from sklearn.datasets import load\_digits

import matplotlib.pyplot as plt

digits = load\_digits()

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(digits.data,

digits.target, test\_size=0.3)

lrModel = LogisticRegression()

lrModel.fit(x\_train, y\_train)

lrModel.score(x\_test, y\_test)

svm = SVC()

svm.fit(x\_train, y\_train)

svm.score(x\_test, y\_test)

rfmodel = RandomForestClassifier(n\_estimators=40)

rfmodel.fit(x\_train, y\_train)

rfmodel.score(x\_test, y\_test)

from sklearn.model\_selection import KFold

KFold(n\_splits=10,random\_state=None,shuffle=False)

array = np.array([1,2,3,4,5,6,7,8,9])

for train\_index, test\_index in KFold(n\_splits=3).split(array):

print(array[train\_index],array[test\_index])

def get\_score(model,x\_train,x\_test,y\_train,y\_test):

model.fit(x\_train,y\_train)

return model.score(x\_test,y\_test)

scores\_logistic = []

scores\_svm = []

scores\_rf = []

for train\_index, test\_index in KFold(n\_splits=3,shuffle=

False,random\_state=None).split(digits.data,digits.target):

X\_train, X\_test, Y\_train, Y\_test = digits.data[train\_index],

digits.data[test\_index], digits.target[train\_index],

digits.target[test\_index]

scores\_logistic.append(get\_score(LogisticRegression(),X\_train,X\_test,

Y\_train,Y\_test))

scores\_svm.append(get\_score(SVC(),X\_train,X\_test,Y\_train,Y\_test))

scores\_rf.append(get\_score(RandomForestClassifier(),X\_train,X\_test,Y

\_train,Y\_test))

scores\_logistic

scores\_rf

scores\_svm

from sklearn.model\_selection import cross\_val\_score

cross\_val\_score(LogisticRegression(solver='liblinear',multi\_class='ovr'),

digits.data, digits.target,cv=3)

cross\_val\_score(SVC(gamma='auto'), digits.data, digits.target,cv=3)

cross\_val\_score(RandomForestClassifier(n\_estimators=40),digits.data,

digits.target,cv=3)

**Q11. Implement  kfold cross valudation technique.**

**12. Implement KNN classification technique.**

import pandas as pd

from sklearn.datasets import load\_iris

iris = load\_iris()

iris.feature\_names

iris.target\_names

df = pd.DataFrame(iris.data, columns=iris.feature\_names)

df.head()

df['target'] = iris.target

df.head()

df[df.target==1].head()

df[df.target==2].head()

df['flower\_name'] = df.target.apply(lambda x:

iris.target\_names[x])

df.head()

df1 = df[df.flower\_name=='setosa']

df2 = df[df.flower\_name=='versicolor']

df3 = df[df.flower\_name=='virginica']

df1

df2

df3

import matplotlib.pyplot as plt

%matplotlib inline

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.scatter(df1['sepal length (cm)'], df1['sepal width

(cm)'],color="green",marker='+')

plt.scatter(df2['sepal length (cm)'], df2['sepal width

(cm)'],color="blue",marker='.')

plt.scatter(df3['sepal length (cm)'], df3['sepal width

(cm)'],color="red",marker='\*')

plt.legend(['setosa', 'versicolor','virginica'])

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

plt.scatter(df1['petal length (cm)'], df1['petal width

(cm)'],color="green",marker='+')

plt.scatter(df2['petal length (cm)'], df2['petal width

(cm)'],color="blue",marker='.')

plt.scatter(df3['petal length (cm)'], df3['petal width

(cm)'],color="red",marker='\*')

plt.legend(['setosa', 'versicolor','virginica'])

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

X = df.drop(['target','flower\_name'], axis='columns')

Y = df.target

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2)

len(X\_train)

len(X\_test)

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(X\_train, Y\_train)

knn.score(X\_test, Y\_test)

ypred = knn.predict(X\_test)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(Y\_test, ypred)

cm

%matplotlib inline

import seaborn as sn

plt.figure(figsize = (10,7))

sn.heatmap(cm, annot=True)

plt.xlabel('Predicted')

plt.ylabel('Truth')

from sklearn.metrics import classification\_report

print (classification\_report(Y\_test, ypred))