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

Same.

Code:

from random import randint

TRAIN\_SET\_LIMIT=1000

TRAIN\_SET\_COUNT=100

TRAIN\_INPUT = list()

TRAIN\_OUTPUT = list()

for i in range (TRAIN\_SET\_COUNT) :

a = randint(0, TRAIN\_SET\_LIMIT )

b = randint(0, TRAIN\_SET\_LIMIT )

c = randint(0, TRAIN\_SET\_LIMIT )

op = a + (2 \* b) + (3 \* c)

TRAIN\_INPUT.append([a ,b, c])

TRAIN\_OUTPUT.append(op)

from sklearn.linear\_model import LinearRegression

predictor = LinearRegression(n\_jobs =-1)

predictor.fit(X = TRAIN\_INPUT , y = TRAIN\_OUTPUT)

X\_Test =[[20 ,40,60]]

outcome = predictor.predict(X = X\_Test)

coefficients = predictor.coef\_

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

B) 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 csv

num\_attributes = 6

a =[]

print("\n The Given Training Data Set \n")

with open('C:/Users/dsouz/OneDrive/Desktop/ML prac/enjoysport.csv' ,'r') as csvfile :

reader = csv.reader(csvfile)

for row in reader:

a.append (row)

print(row)

print("\n the initial value of hypothesis :")

hypothesis = ['0'] \* num\_attributes

print(hypothesis)

for j in range (0,num\_attributes):

hypothesis[j] = a[0][j]

print("\n Find S: Finding a Maximally Specific Hypothesis\n")

for i in range( 0, len(a)) :

if a[i] [num\_attributes] =='yes' :

for j in range (0, num\_attributes) :

if a[i][j]!= hypothesis[j] :

hypothesis[j] ='?'

else :

hypothesis[j] = a[i][j]

print(" for training instance No:{0} the Hypothesis is".format(1) ,hypothesis)

print("\n The Maximally Specific Hypothesis for a given Training Exam ples :\n")

print(hypothesis)

Practical 2  
  
A) Perform Data Loading, Feature selection (Principal Component analysis) and

Feature Scoring and Ranking.

Code:

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.decomposition import PCA

import pandas as pd

from sklearn.preprocessing import StandardScaler

plt.style.use('ggplot')

# Load the data

iris = datasets.load\_iris()

X = iris.data

y = iris.target

# Z-score the features

scaler = StandardScaler()

scaler.fit(X)

X = scaler.transform(X)# The PCA model

pca = PCA(n\_components=2) # estimate only 2 PCs

X\_new = pca.fit\_transform(X)

fig, axes = plt.subplots(1,2)

axes[0].scatter(X[:,0], X[:,1], c=y)

axes[0].set\_xlabel('x1')

axes[0].set\_ylabel('x2')

axes[0].set\_title('Before PCA')

axes[1].scatter(X\_new[:,0], X\_new[:,1], c=y)

axes[1].set\_xlabel('PC1')

axes[1].set\_ylabel('PC2')

axes[1].set\_title('After PCA')

plt.show()

1. 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 :

import csv

with open("C:/Users/dsouz/OneDrive/Desktop/ML prac/enjoysport.csv") as f:

csv\_file=csv.reader(f)

data=list(csv\_file)

s=data[1][:-1]

g=[['?' for i in range(len(s))] for j in range(len(s))]

for i in data:

if i[-1]=="Yes":

for j in range(len(s)):

if i[j]!=s[j]:

s[j]='?'

g[j][j]='?'

elif i[-1]=="No":

for j in range(len(s)):

if i[j]!=s[j]:

g[j][j]=s[j]

else:

g[j][j]="?"

print("\nSteps of Candidate Elimination Algorithm",data.index(i)+1)

print(s)

print(g)

gh=[]

for i in g:

for j in i:

if j!='?':

gh.append(i)

break

print("\nFinal specific hypothesis:\n",s)

print("\nFinal general hypothesis:\n",gh)

Practical No 3

1. 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 np

import pandas as pd

from sklearn import datasets

wine= datasets.load\_wine()

print(wine)

print("Feature:",wine.feature\_names)

print("Labels:",wine.target\_names)

X=pd.DataFrame(wine['data'])

print(X.head(0))

print(wine.data.shape)

y=print(wine.target)

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

X\_train, X\_test, y\_train,y\_test = train\_test\_split(wine.data, wine.target, test\_size=0.30, random\_state=109)

from sklearn.naive\_bayes import GaussianNB

gnb=GaussianNB()

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

y\_pred =gnb.predict(X\_test)

print(y\_pred)

from sklearn import metrics

print("Accuracy:",metrics.accuracy\_score(y\_test,y\_pred))

from sklearn.metrics import confusion\_matrix

cm=np.array(confusion\_matrix(y\_test ,y\_pred))

cm

1. Write a program to implement Decision Tree and Random forest with Prediction,

Test Score and Confusion Matrix.

Code:

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import load\_iris

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

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

import pandas as pd

import numpy as np

from sklearn.tree import DecisionTreeClassifier

# Load a sample dataset (Iris dataset)

data = load\_iris()

X = data.data

y = data.target

iris\_df = pd.DataFrame(data=np.c\_[data['data'], data['target']], columns=data['feature\_names']+['target'])

print(iris\_df.head())

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.9, random\_state=42)

# Create a RandomForestClassifier with custom parameters

clf = DecisionTreeClassifier(random\_state=42)

# Train the classifier on the training data

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

# Make predictions on the test data

dt\_y\_pred = clf.predict(X\_test)

# Calculate the accuracy of the classifier

accuracy = accuracy\_score(y\_test,dt\_y\_pred)

print(f'Accuracy: {accuracy \* 100:.2f}%')

clrf = RandomForestClassifier(n\_estimators= 52,random\_state=42)

# Train the classifier on the training data

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

# Make predictions on the test data

rf\_y\_pred = clrf.predict(X\_test)

# Calculate the accuracy of the classifier

accuracy = accuracy\_score(y\_test, rf\_y\_pred)

print(f'Accuracy: {accuracy \* 100:.2f}%')

dt\_confusion\_matrix= confusion\_matrix(y\_test, dt\_y\_pred)

print("confusion matrix for decision tree:")

print(dt\_confusion\_matrix)

rf\_confusion\_matrix= confusion\_matrix(y\_test, rf\_y\_pred)

print("confusion matrix for Random forest:")

print(rf\_confusion\_matrix)

Practical No 4  
  
A) 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("C:/Users/dsouz/OneDrive/Desktop/ML prac/headbrain.csv")

print(data.shape)

print(data.head())

X=data['Head Size(cm^3)'].values

Y=data['Brain Weight(grams)'].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=c + m \* x

plt.plot(x, y , color='#58b970', label='Regression Line')

plt.scatter(X,Y ,c ='#ef5423',label='Scatter plot')

plt.xlabel('Head Size in cm3')

plt.ylabel('Brain Weight in grams')

plt.legend()

plt.show()

B) For a given set of training data examples stored in a .CSV file implement Logistic

Regression algorithm

Code:

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('https://raw.githubusercontent.com/mk-gurucharan/Classification/master/DMVWrittenTests.csv')

X = dataset.iloc[:, [0, 1]].values

y = dataset.iloc[:, 2].values

dataset.head(5)

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.linear\_model import LogisticRegression

classifier = LogisticRegression()

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

LogisticRegression()

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

y\_pred

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

from sklearn.metrics import accuracy\_score

print ("Accuracy : ", accuracy\_score(y\_test, y\_pred))

Cm

Practical 5

1. 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 np

import math

import csv

def read\_data(filename):

with open(filename, 'r') as csvfile:

datareader = csv.reader(csvfile, delimiter=',')

headers = next(datareader)

metadata = []

traindata = []

for name in headers:

metadata.append(name)

for row in datareader:

traindata.append(row)

return (metadata, traindata)

class Node:

def \_\_init\_\_(self, attribute):

self.attribute = attribute

self.children = []

self.answer = ""

def \_\_str\_\_(self):

return self.attribute

def subtables(data, col, delete):

dict = {}

items = np.unique(data[:, col])

count = np.zeros((items.shape[0], 1), dtype=np.int32)

for x in range(items.shape[0]):

for y in range(data.shape[0]):

if data[y, col] == items[x]:

count[x] += 1

for x in range(items.shape[0]):

dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")

pos = 0

for y in range(data.shape[0]):

if data[y, col] == items[x]:

dict[items[x]][pos] = data[y]

pos += 1

if delete:

dict[items[x]] = np.delete(dict[items[x]], col, 1)

return items, dict

def entropy(S):

items = np.unique(S)

if items.size == 1:

return 0

counts = np.zeros((items.shape[0], 1))

sums = 0

for x in range(items.shape[0]):

counts[x] = sum(S == items[x]) / (S.size \* 1.0)

for count in counts:

sums += -1 \* count \* math.log(count, 2)

return sums

def gain\_ratio(data, col):

items, dict = subtables(data, col, delete=False)

total\_size = data.shape[0]

entropies = np.zeros((items.shape[0], 1))

intrinsic = np.zeros((items.shape[0], 1))

for x in range(items.shape[0]):

ratio = dict[items[x]].shape[0]/(total\_size \* 1.0)

entropies[x] = ratio \* entropy(dict[items[x]][:, -1])

intrinsic[x] = ratio \* math.log(ratio, 2)

total\_entropy = entropy(data[:, -1])

iv = -1 \* sum(intrinsic)

for x in range(entropies.shape[0]):

total\_entropy -= entropies[x]

return total\_entropy / iv

def create\_node(data, metadata):

if (np.unique(data[:, -1])).shape[0] == 1:

node = Node("")

node.answer = np.unique(data[:, -1])[0]

return node

gains = np.zeros((data.shape[1] - 1, 1))

for col in range(data.shape[1] - 1):

gains[col] = gain\_ratio(data, col)

split = np.argmax(gains)

node = Node(metadata[split])

metadata = np.delete(metadata, split, 0)

items, dict = subtables(data, split, delete=True)

for x in range(items.shape[0]):

child = create\_node(dict[items[x]], metadata)

node.children.append((items[x], child))

return node

def empty(size):

s = ""

for x in range(size):

s += " "

return s

def print\_tree(node, level):

if node.answer != "":

print(empty(level), node.answer)

return

print(empty(level), node.attribute)

for value, n in node.children:

print(empty(level + 1), value)

print\_tree(n, level + 2)

metadata, traindata = read\_data("C:/Users/dsouz/OneDrive/Desktop/ML prac/tennisdata.csv")

data = np.array(traindata)

node = create\_node(data, metadata)

print\_tree(node, 0)

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

Code :

from sklearn.datasets import load\_iris

from sklearn.neighbors import KNeighborsClassifier

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

import numpy as np

dataset=load\_iris()

#print(dataset)

X\_train,X\_test,y\_train,y\_test=train\_test\_split(dataset["data"],dataset["target"],random\_state=0)

kn=KNeighborsClassifier(n\_neighbors=1)

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

for i in range(len(X\_test)):

x=X\_test[i]

x\_new=np.array([x])

prediction=kn.predict(x\_new)

print("TARGET=",y\_test[i],dataset["target\_names"][y\_test[i]],"PREDICTED=",prediction,dataset["target\_names"][prediction])

print(kn.score(X\_test,y\_test))

Practical No 6

1. Implement the different Distance methods (Euclidean) with Prediction, Test Score

and Confusion Matrix  
  
Code:  
  
from math import sqrt

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

# calculate euclidean distance

def euclidean\_distance(a, b):

return sqrt(sum((e1-e2) \*\* 2 for e1, e2 in zip(a, b)))

# calculate manhattan distance

def manhattan\_distance(a, b):

return sum(abs(e1-e2) for e1, e2 in zip(a, b))

# calculate minkowski distance

def minkowski\_distance(a, b, p):

return sum(abs(e1-e2)\*\*p for e1, e2 in zip(a, b)) \*\* (1/p)

# define data

# actual values

actual = [1, 0, 0, 1, 0, 0, 1, 0, 0, 1]

# predicted values

predicted = [1, 0, 0, 1, 0, 0, 0, 1, 0, 0]

# calculate distance

dist1 = euclidean\_distance(actual, predicted)

dist2 = manhattan\_distance(actual, predicted)

# calculate distance (p-1)

dist3 = minkowski\_distance(actual, predicted,1)

# calculate distance (p=1)

dist3 = minkowski\_distance(actual, predicted,1)

print(dist3)

# calculate distance (p=2)

dist3 = minkowski\_distance(actual, predicted, 2)

print(dist3)

# confusion matrix

matrix = confusion\_matrix(actual, predicted, labels=[1, 0])

print('Confusion matrix : \n', matrix)

# outcome values order in sklearn

tp, fn, fp, tn = confusion\_matrix(actual, predicted, labels=[1, 0]).reshape(-1)

print('Outcome values : \n', tp, fn, fp, tn)

# classification report for precision, recall f1-score and accuracy

matrix = classification\_report(actual, predicted, labels=[1, 0])

print('Classification report : \n', matrix)

Practical No 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 nm

import matplotlib.pyplot as mtp

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('C:/Users/dsouz/OneDrive/Desktop/ML prac/Mall\_Customers.csv')

x = dataset.iloc[:, [3, 4]].values

#Finding the optimal number of clusters using the dendrogram

import scipy.cluster.hierarchy as shc

dendro = shc.dendrogram(shc.linkage(x, method="ward"))

mtp.title("Dendrogrma Plot")

mtp.ylabel("Euclidean Distances")

mtp.xlabel("Customers")

mtp.show()

#training the hierarchical model on dataset

from sklearn.cluster import AgglomerativeClustering

hc= AgglomerativeClustering(n\_clusters=5, affinity='euclidean', linkage='ward')

y\_pred= hc.fit\_predict(x) #visulaizing the clusters

mtp.scatter(x[y\_pred == 0, 0], x[y\_pred == 0, 1], s = 100, c = 'purple', 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 = 'yellow', 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()

Practical No 8

1. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set

Code :

import pandas as pd

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

data = pd.read\_csv("C:/Users/dsouz/OneDrive/Desktop/ML prac/ds4.csv")

heart\_disease = pd.DataFrame(data)

print(heart\_disease)

model = BayesianModel([

('age', 'Lifestyle'),

('Gender', 'Lifestyle'),

('Family', 'heartdisease'),

('diet', 'cholestrol'),

('Lifestyle', 'diet'),

('cholestrol', 'heartdisease'),

('diet', 'cholestrol')

])

model.fit(heart\_disease, estimator=MaximumLikelihoodEstimator)

HeartDisease\_infer = VariableElimination(model)

print('For Age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4')

print('For Gender enter Male:0, Female:1')

print('For Family History enter Yes:1, No:0')

print('For Diet enter High:0, Medium:1')

print('for LifeStyle enter Athlete:0, Active:1, Moderate:2, Sedentary:3')

print('for Cholesterol enter High:0, BorderLine:1, Normal:2')

q = HeartDisease\_infer.query(variables=['heartdisease'], evidence={

'age': int(input('Enter Age: ')),

'Gender': int(input('Enter Gender: ')),

'Family': int(input('Enter Family History: ')),

'diet': int(input('Enter Diet: ')),

'Lifestyle': int(input('Enter Lifestyle: ')),

'cholestrol': int(input('Enter Cholestrol: '))

})

print(q)

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

Code :

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

# load data points

data = pd.read\_csv('C:/Users/dsouz/OneDrive/Desktop/ML prac/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))

#set k here

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()

Practical No 10

1. Bayesian documents

Code

import pandas as pd

msg = pd.read\_csv('C:/Users/dsouz/OneDrive/Desktop/ML prac/document.csv', names=['message', 'label'])

print("Total Instances of Dataset: ", msg.shape[0])

msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})

X = msg.message

y = msg.labelnum

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

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y)

from sklearn.feature\_extraction.text import CountVectorizer

count\_v = CountVectorizer()

Xtrain\_dm = count\_v.fit\_transform(Xtrain)

Xtest\_dm = count\_v.transform(Xtest)

#df = pd.DataFrame(Xtrain\_dm.toarray(),columns=count\_v.get\_feature\_names())

#print(df[0:5])

from sklearn.naive\_bayes import MultinomialNB

clf = MultinomialNB()

clf.fit(Xtrain\_dm, ytrain)

pred = clf.predict(Xtest\_dm)

for doc, p in zip(Xtrain, pred):

p = 'pos' if p == 1 else 'neg'

print("%s -> %s" % (doc, p))

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

print('Accuracy Metrics: \n')

print('Accuracy: ', accuracy\_score(ytest, pred))

print('Recall: ', recall\_score(ytest, pred))

print('Precision: ', precision\_score(ytest, pred))

print('Confusion Matrix: \n', confusion\_matrix(ytest, pred))