Practical Number: 1A

Aim: Design a simple machine learning model to train the training instances and test the same using Python**.**

Code:

```
# python library to generate random numbers
from random import randint

# the limit within which random numbers are generated
TRAIN_SET_LIMIT = 1000

# to create exactly 100 data items
TRAIN_SET_COUNT = 100

# list that contains input and corresponding output
TRAIN_INPUT = list()
TRAIN_OUTPUT = list()

# loop to create 100 data  items with three columns each
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)

# creating the output for each data item
   op = a + (2 * b) + (3 * c)
   TRAIN_INPUT.append([a, b, c])

# adding each output to output list
   TRAIN_OUTPUT.append(op)

# Sk-Learn contains the linear regression model
# To install sklearn package in python type in command prompt: pip3 install sklearn
from sklearn.linear_model import LinearRegression

# Initialize the linear regression model
# n_jobs int, default=None : The number of jobs to use for the computation.
# This will only provide speedup in case of sufficiently large problems.
# -1 means using all processors
predictor = LinearRegression(n_jobs =-1)

# Fill the Model with the Data
predictor.fit(X = TRAIN_INPUT, y = TRAIN_OUTPUT)

# Random Test data
X_TEST = [[ 10, 20, 30 ]]
```

Practical Number: 1B

Aim: 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
with open('tennis.csv', 'r') as f:
  reader = csv.reader(f)
  your_list = list(reader)
h = [['0', '0', '0', '0', '0', '0']]
for i in your_list:
  print(i)
  if i[-1]=="Yes":
    j=0
    for x in i:
      if x!="Yes":
        if x!=h[0][j] and h[0][j]=='0':
          h[0][j]=x
        elif x!=h[0][j] and h[0][j]!='0':
          h[0][j]='?'
        else:
          pass
      j=j + 1
print("Most specific hypothesis is")
print(h)
```

tennis.csv file

PRACTICAL NO: 2A

Aim: Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.

Code:

```
from pandas import read_csv

from sklearn.decomposition import PCA

from sklearn.ensemble import ExtraTreesClassifier


#We will use PCA to select best 3 Principal components from Pima Indians Diabetes dataset.

path = 'pima-indians-diabetes.csv'

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe = read_csv(path, names=names)

array = dataframe.values


#Next, we will separate array into input and output components −

X = array[:,0:8]

Y = array[:,8]


#The following lines of code will extract features from dataset −

pca = PCA(n_components=3)

fit = pca.fit(X)

print("Explained Variance: %s",fit.explained_variance_ratio_)

print(fit.components_)


#From the output, we can observe that there are scores for each attribute.

#The higher the score, higher is the importance of that attribute.

model = ExtraTreesClassifier()

model.fit(X, Y)

print("Scores for each attribute")

print(model.feature_importances_)
```

PRACTICAL NO: 2B

Aim: 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("tennis.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:
```

Practical Number: 3A

Aim: 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.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt


df = pd.read_csv('Naive-Bayes-Classification-Data.csv')


#Data pre-processing step

#Here, we’ll create the x and y variables by taking them from the dataset

#and using the train_test_split function of scikit-learn to split the data into training
and test sets.

#Note that the test size of 0.25 indicates we’ve used 25% of the data for testing.

#random_state ensures reproducibility. For the output of train_test_split, we get
x_train, x_test, y_train, and y_test values.


x=df.drop('diabetes',axis=1)

y=df['diabetes']

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=42)


#Train the model

#We’re going to use x_train and y_train, obtained above, to train our naive Bayes
classifier model.

#We’re using the fit method and passing the parameters.
```

```
model=GaussianNB()
model.fit(x_train,y_train)

#Prediction
#Once the model is trained, it’s ready to make predictions.
#We can use the predict method on the model and pass x_test as a parameter to get the output as y_pred.
#Notice that the prediction output is an array of real numbers corresponding to the input array.

y_pred=model.predict(x_test)
print(y_pred)

#Model Evaluation
#Finally, we need to check to see how well our model is performing on the test data.
#We evaluate our model by finding the accuracy score produced by the model.

accuracy=accuracy_score(y_test,y_pred)*100
print(accuracy)
```

PRACTICAL NUMBER: 3B

Aim: Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix**.**

Code:

```
#Numerical computing libraries
import pandas as pd
import numpy as np
#Visalization libraries
import matplotlib.pyplot as plt
import seaborn as sns

raw_data = pd.read_csv('kyphosis-data.csv')
raw_data.columns

#Exploratory data analysis
raw_data.info()
sns.pairplot(raw_data, hue = 'Kyphosis')
plt.show()
#Split the data set into training data and test data
from sklearn.model_selection import train_test_split
x = raw_data.drop('Kyphosis', axis = 1)
y = raw_data['Kyphosis']
x_training_data, x_test_data, y_training_data, y_test_data = train_test_split(x, y, test_size =
0.3)
#Train the decision tree model
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(x_training_data, y_training_data)
predictions = model.predict(x_test_data)
#Measure the performance of the decision tree model
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

print('Performance of decision tree model:')
print(classification_report(y_test_data, predictions))
print('Confusion matrix of decision tree model:')
print(confusion_matrix(y_test_data, predictions))

#Train the random forests model
from sklearn.ensemble import RandomForestClassifier
random_forest_model = RandomForestClassifier()
random_forest_model.fit(x_training_data, y_training_data)
random_forest_predictions = random_forest_model.predict(x_test_data)

#Measure the performance of the random forest model
print('Performance of random forest model:')
print(classification_report(y_test_data, random_forest_predictions))
print('Confusion matrix of random forest model:')
print(confusion_matrix(y_test_data, random_forest_predictions))
```

PRACTICAL NUMBER:4A

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

Code:

```
#Import the required libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#Import the data set
#Reading Data
data = pd.read_csv('headbrain.csv')
#With the shape method, comes the flexibility to obtain the dimensions of any Python
object.
print("Dimensions:",data.shape)
#The head function in Python displays the first five rows of the dataframe by default.
print("First 5 rows:",data.head())

#Assigning ‘X’ as independent variable and ‘Y’ as dependent variable
# Coomputing X and Y
X = data['Head Size(cm^3)'].values
Y = data['Brain Weight(grams)'].values

#Next, in order to calculate the slope and y-intercept we first need to compute the means of
‘x’ and ‘y’.
#Mean X and Y
mean_x = np.mean(X)
mean_y = np.mean(Y)
# Total number of values
n = len(X)
#Calculate the values of the slope and y-intercept
#Using the formula to calculate 'm' and 'c'
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)

#Printing coefficients
print("Coefficients")
print(m, c)
```

PRACTICAL NUMBER:4B

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

Code:

```
#Importing libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix

from sklearn.metrics import accuracy_score

from matplotlib.colors import ListedColormap


#Reading the data

dataset = pd.read_csv('User_Data.csv')

#Now, to predict whether a user will purchase the product or not, one needs to find out the relationship between Age and Estimated Salary

# input

x = dataset.iloc[:, [2, 3]].values

# output

y = dataset.iloc[:, 4].values


#Splitting the dataset to train and test. 75% of data is used for training the model and 25% of it is used to test the performance of our model.

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.25, random_state = 0)


#Performing feature scaling

sc_x = StandardScaler()
```

```
xtrain = sc_x.fit_transform(xtrain)

xtest = sc_x.transform(xtest)

print (xtrain[0:10, :])


#Training The Model

classifier = LogisticRegression(random_state = 0)

classifier.fit(xtrain, ytrain)


#Performing predictions on test data

Y_prediction = classifier.predict(xtest)


#Testing the performance of model using confusion matrix

co_mat = confusion_matrix(ytest, Y_prediction)

print ("Confusion Matrix : \n", co_mat)


#Accuracy of model

print ("Accuracy : ", accuracy_score(ytest, Y_prediction))


#Visualizing the performance of our model.

X_set, y_set = xtest, ytest

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,

stop = X_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X_set[:, 1].min() - 1,

stop = X_set[:, 1].max() + 1, step = 0.01))


plt.contourf(X1, X2, classifier.predict(

np.array([X1.ravel(), X2.ravel()]).T).reshape(

X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))
```

PRACTICAL NUMBER: 5

Aim: Write a program to implement k-Nearest Neighbour algorithm to classify the iris dataset.

Code:

```
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report

from sklearn.metrics import confusion_matrix

import pandas as pd

import numpy as numpy

from sklearn import datasets


#Loading the iris dataset

iris = datasets.load_iris()

iris_data = iris.data

iris_labels = iris.target

#The train_test_split function is for splitting a single dataset for two different
#purposes: training and testing. The testing subset is for building your model. The
#testing subset is for using the model on unknown data to evaluate the performance
#of the model. The first parameter is the dataset you're selecting to use.
#test_size: This parameter specifies the size of the testing dataset.

X_train,X_test,y_train,y_test = (train_test_split(iris_data,iris_labels,test_size=0.3))

# Classifier implementing the k-nearest neighbors.

classifier = KNeighborsClassifier(n_neighbors=13)

classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

print("Accuracy is: ")

print(classification_report(y_test,y_pred))

print("Confusion matrix is:")

print(confusion_matrix(y_test,y_pred))
```

PRACTICAL NUMBER: 6

Aim: Implement the classification model using clustering for following techniques with Kmeans clustering with Prediction, Test Score and Confusion Matrix.

Code:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import sklearn.metrics as sm
import pandas as pd
import numpy as np

#Loading the iris dataset
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
X_train,X_test,y_train,y_test = (train_test_split(X,y,test_size=0.3))
#n_clusters: The number of clusters to form as well as the number of centroids to
generate.
model = KMeans(n_clusters=3)
model.fit(X)
y_pred = model.predict(X_test)

print("Model labels:")
print(model.labels_)
print("Accuracy is: ")
print(classification_report(y_test,y_pred))
print("Confusion matrix is:")
print(confusion_matrix(y_test,y_pred))

plt.figure(figsize=(14,7))
colormap = np.array(['red','cyan','black'])
plt.subplot(1,2,1)
plt.scatter(X.Petal_Length,X.Petal_Width, c=colormap[y.Targets], s = 40)
plt.title("Real clusters")
plt.subplot(1,2,2)
plt.scatter(X.Petal_Length,X.Petal_Width, c=colormap[model.labels_], s= 40)
plt.title("K Means Cluster")
plt.show()
```

PRACTICAL NUMBER: 7

Aim: Implement hierachical clustering using Dendogram

Code:

```
import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import scipy.cluster.hierarchy as shc

from sklearn.cluster import AgglomerativeClustering


customer_data = pd.read_csv('shopping_data.csv')

#shape: Returns tuple of shape (Rows, columns) of dataframe/series

print("Rows,colums in data:",customer_data.shape)

#head: Displays the first five rows of the dataframe by default.

print("First five rows:",customer_data.head())

#iloc: helps us to select a specific row or column from the data set.

data = customer_data.iloc[:,3:5].values

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

plt.title("Customer Dendograms")

# create a dendrogram variable linkage is actually the algorithm itself of hierarchical
clustering and then

# in linkage we have to specify on which data we apply and engage.

# Ward method is actually a method that tries to minimize the variance within each cluster.

# We choose Euclidean distance and ward method for our algorithm class.

dend = shc.dendrogram(shc.linkage(data,method = 'ward'))

cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')

cluster.fit_predict(data)

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

plt.scatter(data[:,0], data[:,1], c=cluster.labels_,cmap='rainbow')

plt.show()
```

PRACTICAL NUMBER: 8

Aim: 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 numpy as np

import csv

import pandas as pd

from pgmpy.models import BayesianModel

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.inference import VariableElimination

import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)

#read Cleveland Heart Disease data

heartDisease = pd.read_csv('heart.csv')

heartDisease = heartDisease.replace('?',np.nan)

#display the data

print('Few examples from the dataset are given below')

heartDisease.head()

#Model Bayesian Network

model =
BayesianModel([('age','trestbps'),('age','fbs'),('sex','trestbps'),('exang','trestbps'),('trestbps','heartdisease'),

('fbs','heartdisease'),('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')])

#Learning CPDs using Maximum Likelihood Estimators

print("\nLearning CPDs using Maximum Likelihood Estimators")

model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

#Inferencing with Bayesian Network

print("Inferencing with Bayesian Network")

HeartDisease_infer = VariableElimination(model)

print("1.Probability of HeartDisease given evidence = restecg")

q1 = HeartDisease_infer.query(variables=['heartdisease'],evidence={'restecg':1})

print(q1)
```

practical No 9:

```
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 derivatives_sigmoid(x):
    return x*(1-x)
epoch=5
lr=0.1
inputlayer_neurons=2
hiddenlayer_neurons=1
output_neurons=1
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,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=derivatives_sigmoid(output)
    d_output=EO*outgrad
    EH=d_output.dot(wout.T)
    hiddengrad=derivatives_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,"start-------")
    print("input:\n"+str(y))
    print("actual output:\n",output)
    print("pradicted output:\n",output)
    print("-------Epoch",i+1,"end-------\n")
    print("input:\n"+str(X))
    print("actual output:\n",str(y))
```

practical No 10

```
import pandas as pd
import math
import numpy as np
data=pd.read_csv("PlayTennis.csv")
features=[feat for feat in data]
features.remove("PlayTennies")
class Node:
    def__init__(self):
        self.children=[]
        self.value=""
        self.isLeaf=False
        self.pred=""
        def entropy(examples):
            pos=0.0
            neg=0.0
            for_ row in examples.iterrows():
                if row["PlayTennis"]=="Yes":
                    pos+=1
                    else:
                        neg+=1
                        if pos==0.0 or neg==0.0:
                            return 0.0
                        else:
                            p=pos/(pos+neg)
                            n=neg/(pos+neg)
                            return-(p*math.log(p,2)+n*math.log(n,2))
                        def ID3(examples,attrs):
                            root=Node()
                            max_gain=0
                            max_feat=""
                            for feature in attrs:
                                gain=info_gain(examples,feature)
                                if gain>max_gain:
                                    max_gain=gain
                                    max_feat=feature
                                    root.value=max_feat
                                    uniq=np.unique(examples[max_feat]==u)
                                    if entropy(subdatd)0.0
                                    newNode=Node()
                                    newNode.isleaf=True
                                    newNode.value=u
                                    newNode.pred=np.unique(subdata["PlayTennis"])
                                    root.children.append(newNode)
                                    else:
                                        dummyNode=Node()
                                        dummyNode.value=u
                                        new_attrs.remove(max_feat)
                                        child=ID3(subdata,new_attrs)
                                        dummyNode.children.append(child)
                                        root.children.append(dummyNode)
                                        return root
                                    def printTree(root:Node,depth=0):
                                        for i in range(depth):
                                            print("\t", end="")
                                            print(root.value,end="")
                                            if root.isleaf:
                                                print("->",root.pred)
                                                print()
                                                for child in root.children:
                                                    printTree(child,depth+1)
                                                    def classify(root:Node,new):
                                                        for child in root.children:
                                                            if child.value==new[root.value]:
                                                                if child.isleaf:
                                                                    print("Predict label for new example",new,"is:",child.pred)
                                                                    exit
                                                                    else:
                                                                        classify(child.children[0],new)
                                                                        root=ID3(data,features)
                                                                        print("Decision Tree is:")
                                                                        printTree(root)
                                                                        print("-----------")
                                                                        new={"Outlook":"Sunny","Temperature":"Hot","Humidity":"Normal","wind":"Strong"}
                                                                        classify(root,new)
```