#Program-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. (enjoySport Dataset)

"""

import pandas as pd

import numpy as np

data = pd.read\_csv(r"C:\Users\HPR\Desktop\ML Syllabus\2.csv")

data

concepts = np.array(data)[:,:-1] #The first colon (:) indicates that all rows of the array are selected.

                                 #The second part (:-1) specifies that all columns except the last one are selected.

Concepts

target = np.array(data)[:,-1] #First colon (:): Selects all rows in the array.

                              #-1: refers to the last column

target

def train(con, tar):

    for i, val in enumerate(tar):

        if val == 'yes':

            specific\_h = con[i].copy()

            break

    for i, val in enumerate(con):

        if tar[i] == 'yes':

            for x in range(len(specific\_h)):

                if val[x] != specific\_h[x]:

                    specific\_h[x] = '?'

                else:

                    pass

    return specific\_h

print(train(concepts, target))

#Program-2

""""

For a given set of training data examples stored in a .CSV file (enjoySport Dataset),

implement and demonstrate the Candidate-Elimination algorithm to output a description of the

set of all hypotheses consistent with the training examples.

"""""

import pandas as pd

import numpy as np

data = pd.read\_csv(r"C:\Users\HPR\Desktop\ML Syllabus\2.csv")

concepts = np.array(data.iloc[:,0:-1])

target = np.array(data.iloc[:,-1])

def learn(concepts, target):

    specific\_h = concepts[0].copy()

    print("initialization of specific\_h \n",specific\_h)

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

    print("initialization of general\_h \n", general\_h)

    for i, h in enumerate(concepts):

        if target[i] == "yes":

            print("If instance is Positive ")

            for x in range(len(specific\_h)):

                if h[x]!= specific\_h[x]:

                    specific\_h[x] ='?'

                    general\_h[x][x] ='?'

        if target[i] == "no":

            print("If instance is Negative ")

            for x in range(len(specific\_h)):

                if h[x]!= specific\_h[x]:

                    general\_h[x][x] = specific\_h[x]

                else:

                    general\_h[x][x] = '?'

        print(" step {}".format(i+1))

        print(specific\_h)

        print(general\_h)

        print("\n")

        print("\n")

    indices = [i for i, val in enumerate(general\_h) if val == ['?', '?', '?', '?', '?', '?']]

    for i in indices:

        general\_h.remove(['?', '?', '?', '?', '?', '?'])

    return specific\_h, general\_h

s\_final, g\_final = learn(concepts, target)

print("Final Specific\_h:", s\_final, sep="\n")

print("Final General\_h:", g\_final, sep="\n")

#Program-3

""""

Write a program to demonstrate the working of the decision tree based ID3 algorithm.

Use an appropriate data set for building the decision tree and apply the knowledge to classify

a new sample.(Play Tennis Dataset)

""""

# Load libraries

import numpy as np

import pandas as pd

from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

df=pd.read\_csv(r"C:\Users\HPR\Desktop\ML Syllabus\Play Tennis.csv")

value=['Outlook','Temprature','Humidity','Wind']

df

len(df)

df.shape  #To see the number of rows and columns in our dataset:

df.head() #prints first five samples

df.tail() #prints last five samples

df.describe()     #To see statistical details of the dataset:

#machine learning algorithms can only learn from numbers (int, float, doubles .. )

#so let us encode it to int

from sklearn import preprocessing

string\_to\_int= preprocessing.LabelEncoder() #encode your data

df=df.apply(string\_to\_int.fit\_transform) #fit and transform it

df

#To divide our data into attribute set and Label:

feature\_cols = ['Outlook','Temprature','Humidity','Wind']

X = df[feature\_cols ]                               #contains the attribute

y = df.Play\_Tennis

#To divide our data into training and test sets:

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30)

# perform training

from sklearn.tree import DecisionTreeClassifier                             # import the classifier

classifier =DecisionTreeClassifier(criterion="entropy", random\_state=100)     # create a classifier object

classifier.fit(X\_train, y\_train)                                              # fit the classifier with X and Y data

#Predict the response for test dataset

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

# Model Accuracy, how often is the classifier correct?

from sklearn.metrics import accuracy\_score

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

data\_p=pd.DataFrame({'Actual':y\_test, 'Predicted':y\_pred})

data\_p

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

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

#Program-4

""""

Write a program to demonstrate the working of the decision tree based CART algorithm.

(Play Tennis Dataset)

""""

# Load libraries

import numpy as np

import pandas as pd

from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

df=pd.read\_csv(r"C:\Users\HPR\Desktop\ML Syllabus\Play Tennis.csv")

value=['Outlook','Temprature','Humidity','Wind']

df

len(df)

df.shape  #To see the number of rows and columns in our dataset:

df.head() #prints first five samples

df.tail() #prints last five samples

df.describe()     #To see statistical details of the dataset:

#machine learning algorithms can only learn from numbers (int, float, doubles .. )

#so let us encode it to int

from sklearn import preprocessing

string\_to\_int= preprocessing.LabelEncoder()                     #encode your data

df=df.apply(string\_to\_int.fit\_transform) #fit and transform it

df

#To divide our data into attribute set and Label:

feature\_cols = ['Outlook','Temprature','Humidity','Wind']

X = df[feature\_cols ]                               #contains the attribute

y = df.Play\_Tennis

#To divide our data into training and test sets:

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30)

# perform training

from sklearn.tree import DecisionTreeClassifier                             # import the classifier

classifier =DecisionTreeClassifier(criterion="gini", random\_state=100)     # create a classifier object

classifier.fit(X\_train, y\_train)                                              # fit the classifier with X and Y data

#Predict the response for test dataset

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

# Model Accuracy, how often is the classifier correct?

from sklearn.metrics import accuracy\_score

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

data\_p=pd.DataFrame({'Actual':y\_test, 'Predicted':y\_pred})

data\_p

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

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

#Program-5

""""

Write a program to demonstrate Decision tree regression for a given dataset.(Play\_Tennis\_reg)

""""

# Load libraries

import numpy as np

import pandas as pd

from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

df=pd.read\_csv(r"C:\Users\HPR\Desktop\ML Syllabus\Play\_Tennis\_reg.csv")

len(df)

df.shape  #To see the number of rows and columns in our dataset:

# Select features and target

X = df.drop("Golf Players", axis=1)

y = df['Golf Players']

X

y

from sklearn.preprocessing import LabelEncoder

from sklearn import preprocessing

string\_to\_int= preprocessing.LabelEncoder()                     #encode your data

X=X.apply(string\_to\_int.fit\_transform) #fit and transform it

X

from sklearn.tree import DecisionTreeRegressor

reg = DecisionTreeRegressor()

reg = reg.fit(X, y)

y\_pred = reg.predict([[2,1,0,1]])

# print the Result

print("Result is: ", y\_pred)

y\_pred = reg.predict([[2,1,0,0]])

# print the Result

print("Result is: ", y\_pred)

y\_pred = reg.predict([[1,2,0,0]])

# print the Result

print("Result is: ", y\_pred)

""""

#Implement a Perceptron Algorithm for AND Logic Gate with 2-bit Binary Input.

#Test for the following Hyper parameters:

    -->w1=1.2, w2=0.6, bias =0, threshold = 1, learning\_rate = 0.5

    -->w1=1.2, w2=0.6, bias =0.5, threshold = 1, learning\_rate = 0.5

    -->w1=1.2, w2=0.6, bias =1.0, threshold = 1, learning\_rate = 0.5

    -->w1=1.2, w2=0.6, bias =-1.0, threshold = 1, learning\_rate = 0.5

"""""

import numpy as np

# Define inputs and expected outputs for an AND gate

inputs = np.array([

    [0, 0],

    [0, 1],

    [1, 0],

    [1, 1]

])

expected\_outputs = np.array([0, 0, 0, 1])

# Initialize weights, bias, threshold, and learning rate

w1, w2 = 1.2, 0.6

bias =-1.0

threshold = 1

learning\_rate = 0.5

# Activation function

def activation\_function(net\_input):

    return 1 if net\_input >= threshold else 0

# Training loop

epochs = 0

while True:

    error\_count = 0  # Track the number of misclassifications

    for i in range(len(inputs)):

        # Calculate weighted sum including the bias

        net\_input = w1 \* inputs[i][0] + w2 \* inputs[i][1] + bias

        # Apply activation function

        output = activation\_function(net\_input)

        # Calculate error

        error = expected\_outputs[i] - output

        # Update weights and bias if there is an error

        if error != 0:

            w1 += learning\_rate \* error \* inputs[i][0]

            w2 += learning\_rate \* error \* inputs[i][1]

            bias += learning\_rate \* error  # Update bias as well

            error\_count += 1

    epochs += 1

    # Break if there are no errors

    if error\_count == 0:

        break

# Display results

print(f"Training completed in {epochs} epochs")

print(f"Final weights: w1 = {w1}, w2 = {w2}, bias = {bias}")

# Test the perceptron on all input cases

print("Testing perceptron for AND gate:")

for i in range(len(inputs)):

    net\_input = w1 \* inputs[i][0] + w2 \* inputs[i][1] + bias

    output = activation\_function(net\_input)

    print(f"Input: {inputs[i]}, Output: {output}, Expected: {expected\_outputs[i]}")

#Program-7

""""

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. (Iris Dataset)

""""

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

# Load the Iris dataset

data = pd.read\_csv("Iris.csv")

# Select features and target

X = data.drop("Species", axis=1)

y = data['Species']

X

Y

# Encoding the Species column to get numerical class

le = LabelEncoder()

y = le.fit\_transform(y)

y

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Gaussian Naive Bayes classifier

gnb = GaussianNB()

# Train the classifier on the training data

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

# Make predictions on the testing data

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

# Calculate the accuracy of the model

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

print(f"The Accuracy of Prediction on Iris Flower is: {accuracy}")

# Create a DataFrame to display actual and predicted values

df = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

# Print the table

print(df)

#Program-8

""""

Implement Naive Bayes Classifier for text classification task.

url: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

""""

import pandas as pd

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

from sklearn.naive\_bayes import MultinomialNB, GaussianNB

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics import accuracy\_score, f1\_score

import matplotlib.pyplot as plt

# Load the SMS Spam Collection Dataset

sms\_data = pd.read\_csv("spam.csv", encoding='latin-1') # url: <https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset>

# Preprocess the data

sms\_data = sms\_data[['v1', 'v2']]

sms\_data = sms\_data.rename(columns={'v1': 'label', 'v2': 'text'})

sms\_data

# Split the data into features and labels

X = sms\_data['text']

y = sms\_data['label']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# EDA 1: Distribution of Classes

class\_distribution = sms\_data['label'].value\_counts()

class\_distribution.plot(kind='pie', autopct='%1.1f%%', colors=['#66b3ff','#99ff99'])

plt.title('Distribution of Spam and Ham Messages')

plt.show()

# Create a CountVectorizer to convert text data into numerical features

vectorizer = CountVectorizer()

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

X\_train\_vec

# Train a Multinomial Naive Bayes classifier

mnb = MultinomialNB(alpha=0.8, fit\_prior=True, force\_alpha=True)

mnb.fit(X\_train\_vec, y\_train)

# Train a Gaussian Naive Bayes classifier

gnb = GaussianNB()

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

# Evaluate the models using accuracy and F1-score

y\_pred\_mnb = mnb.predict(X\_test\_vec)

accuracy\_mnb = accuracy\_score(y\_test, y\_pred\_mnb)

f1\_mnb = f1\_score(y\_test, y\_pred\_mnb, pos\_label='spam')

y\_pred\_gnb = gnb.predict(X\_test\_vec.toarray())

accuracy\_gnb = accuracy\_score(y\_test, y\_pred\_gnb)

f1\_gnb = f1\_score(y\_test, y\_pred\_gnb, pos\_label='spam')

# Print the results

print("Multinomial Naive Bayes - Accuracy:", accuracy\_mnb)

print("Multinomial Naive Bayes - F1-score for 'spam' class:", f1\_mnb)

print("Gaussian Naive Bayes - Accuracy:", accuracy\_gnb)

print("Gaussian Naive Bayes - F1-score for 'spam' class:", f1\_gnb)

#Program-9

""""

Write a program to demonstrate Random Forest for classification task on a given dataset.(Iris Dataset)

""""

# load the iris dataset

from sklearn.datasets import load\_iris

iris = load\_iris()

# store the feature matrix (X) and response vector (y)

X = iris.data

y = iris.target

# Count the number of samples

num\_samples = X.shape[0] # The number of rows represents the number of samples

print(f'Number of samples in the Iris dataset: {num\_samples}')

# splitting X and y into training and testing sets

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

# Count the number of samples in the training and testing sets

train\_samples = X\_train.shape[0] # Number of rows in X\_train

test\_samples = X\_test.shape[0] # Number of rows in X\_test

print(f'Number of samples in the training set: {train\_samples}')

print(f'Number of samples in the testing set: {test\_samples}')

# importing random forest classifier from assemble module

from sklearn.ensemble import RandomForestClassifier

# creating a RF classifier

rf = RandomForestClassifier(n\_estimators = 100)

# Training the model on the training dataset

# fit function is used to train the model using the training sets as parameters

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

# performing predictions on the test dataset

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

# comparing actual response values (y\_test) with predicted response values (y\_pred)

from sklearn import metrics

print("Random Forest model accuracy(in %):", metrics.accuracy\_score(y\_test, y\_pred)\*100)

# Print the actual and predicted values

print("Actual values:", y\_test)

print("Predicted values:", y\_pred)

import pandas as pd

# Create a DataFrame to display actual and predicted values

df = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

# Print the table

print(df)

# Assuming the classes are as follows:

label\_mapping = {0: "iris-setosa", 1: "iris-versicolor", 2: "iris-virginica"}

y\_pred=rf.predict([[3, 3, 2, 2]])

print("Result is:", label\_mapping[y\_pred[0]])

#Program-10

""""

Implement AdaBoost ensemble method on a given dataset.(Iris dataset)

""""

import pandas as pd

import numpy as np

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

from sklearn.ensemble import AdaBoostClassifier

#import warnings warnings.filterwarnings("ignore")

# Reading the dataset from the csv file # separator is a vertical line, as seen in the dataset

data = pd.read\_csv("Iris.csv")

# Printing the shape of the dataset

print(data.shape)

data.head()

data = data.drop('Id',axis=1)

X = data.iloc[:,:-1]

y = data.iloc[:,-1]

print("Shape of X is %s and shape of y is %s"%(X.shape,y.shape))

total\_classes = y.nunique()

print("Number of unique species in dataset are: ",total\_classes)

distribution = y.value\_counts()

print(distribution)

X\_train, X\_val, Y\_train, Y\_val = train\_test\_split( X, y, test\_size=0.25, random\_state=42)

# Creating adaboost classifier model

adb = AdaBoostClassifier()

adb\_model = adb.fit(X\_train,Y\_train)

print("The accuracy of the model on validation set is", adb\_model.score(X\_val,Y\_val))

from sklearn.metrics import accuracy\_score

# Make predictions on the testing data

y\_pred = adb\_model.predict(X\_val)

# Calculate the accuracy of the model

accuracy = accuracy\_score(Y\_val, y\_pred)

print(f"The Accuracy of Prediction on Iris Flower is: {accuracy}")

# Create a DataFrame to display actual and predicted values

df = pd.DataFrame({'Actual': Y\_val, 'Predicted': y\_pred})

# Print the table

print(df)