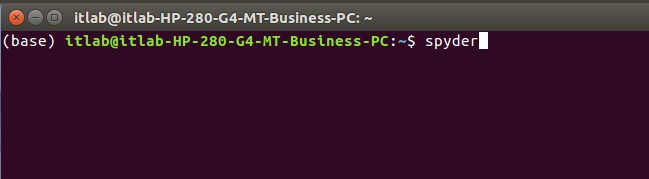
# SPYDER :



1. **FIND S:**
2. **import** pandas as pd
3. **import** numpy as np

4)

1. #to read the data in the csv file
2. data **=** pd.read\_csv("data.csv")
3. print(data,"n")

8)

1. #making an array of all the attributes
2. d **=** np.array(data)[:,:**-**1]
3. print("n The attributes are: ",d)
4. #segragating the target that has positive and negative examples
5. target **=** np.array(data)[:,**-**1]
6. print("n The target is: ",target)
7. #training function to implement find-s algorithm

|  |  |
| --- | --- |
| 18) **def** | train(c,t): |
| 19) | **for** i, val **in** enumerate(t): |
| 20) | **if** val **==** "Yes": |
| 21) | specific\_hypothesis **=** c[i].copy() |
| 22) | **break** |
| 23) |  |
| 24) | **for** i, val **in** enumerate(c): |
| 25) | **if** t[i] **==** "Yes": |
| 26) | **for** x **in** range(len(specific\_hypothesis)): |
| 27) | **if** val[x] !**=** specific\_hypothesis[x]: |
| 28) | specific\_hypothesis[x] **=** '?' |
| 29) | **else**: |
| 30) | **pass** |
| 31) |  |
| 32) | **return** specific\_hypothesis |

33)

1. #obtaining the final hypothesis
2. print("n The final hypothesis is:",train(d,target))

# Candidate-Elimination algorithm

import numpy as np

def candidate\_elimination(examples):

specific\_hypothesis = np.array(['0'] \* len(examples[0][0])) general\_hypothesis = np.array(['?'] \* len(examples[0][0]))

# Iterate over the examples for x, y in examples:

if y == '1':

# If the example is positive, update the specific boundary for i in range(len(x)):

if x[i] != specific\_hypothesis[i]: specific\_hypothesis[i] = '?'

# Update the general boundary for i in range(len(x)):

if specific\_hypothesis[i] == '?': general\_hypothesis[i] = '?'

else:

# If the example is negative, update the general boundary for i in range(len(x)):

if x[i] != specific\_hypothesis[i]:

general\_hypothesis[i] = specific\_hypothesis[i]

return tuple(specific\_hypothesis), tuple(general\_hypothesis)

# ID3

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split from sklearn.datasets import load\_iris

# Load the iris dataset iris = load\_iris()

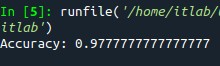
# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.3) # Create an instance of the DecisionTreeClassifier class

clf = DecisionTreeClassifier(criterion='entropy') # Train the model using the training data clf.fit(X\_train, y\_train)

# Predict the classes of the testing data y\_pred = clf.predict(X\_test)

# Print the accuracy score of the model

print("Accuracy:", clf.score(X\_test, y\_test))

# Artificial Neural Network by implementing the Back propagation algorithm

from sklearn.neural\_network import MLPClassifier from sklearn.datasets import make\_classification from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score

# Generate a random dataset

X, y = make\_classification(n\_samples=1000)

# Split the dataset into training and testing sets

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

# Create a multi-layer perceptron classifier with one hidden layer of 5 neurons mlp = MLPClassifier( max\_iter=1000, solver='sgd', random\_state=42)

# Train the model using backpropagation mlp.fit(X\_train, y\_train)

# Calculate the accuracy score of the model accuracy = mlp.score(X\_test, y\_test)

print(f"Accuracy: {accuracy}")

# naïve Bayesian classifier

from sklearn.naive\_bayes import GaussianNB from sklearn.datasets import load\_iris

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

# Load the iris dataset iris = load\_iris()

# Split the data into training and test sets

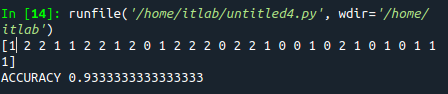
X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.2)

# Train a Gaussian Naive Bayes classifier gnb = GaussianNB()

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

# Test the classifier on the test data y\_pred = gnb.predict(X\_test)

print(y\_pred)

print("ACCURACY",gnb.score(X\_test,y\_test))

# Bayesian network

import pandas as pd

from sklearn.naive\_bayes import GaussianNB from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score

from pgmpy.estimators import MaximumLikelihoodEstimator from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

# Load the heart disease dataset

df = pd.read\_csv('heart\_disease.csv')

# Encode the categorical variables using LabelEncoder le = LabelEncoder()

for column in df.columns:

if df[column].dtype == 'object':

df[column] = le.fit\_transform(df[column])

# Split the data into inputs and targets

inputs = df.iloc[:, :-1] targets = df.iloc[:, -1]

# Split the data into training and testing sets

train\_inputs, test\_inputs, train\_targets, test\_targets = train\_test\_split(inputs, targets, test\_size=0.2, random\_state=42)

# Create a Bayesian network model using the pgmpy library

model = BayesianModel([('age', 'heartdisease'), ('sex', 'heartdisease'), ('exang', 'heartdisease'), ('cp', 'heartdisease'), ('heartdisease', 'restecg'), ('heartdisease', 'chol')])

# Learn the parameters of the model using Maximum Likelihood Estimation model.fit(df, estimator=MaximumLikelihoodEstimator)

# Infer the posterior probabilities of the target variable using Variable Elimination infer = VariableElimination(model)

posterior = infer.query(['heartdisease'], evidence={'age': 28, 'sex': 1, 'exang': 1, 'cp': 2, 'restecg': 1,

'chol': 200})

print('Probability of heart disease:', posterior.values[1])

# Train a Gaussian Naive Bayes classifier using the training data nb = GaussianNB()

nb.fit(train\_inputs, train\_targets)

# Test the classifier using the testing data predictions = nb.predict(test\_inputs)

# Calculate the accuracy of the predictions

accuracy = accuracy\_score(test\_targets, predictions)

print('Accuracy:', accuracy)

# EM algorithm

from sklearn.datasets import make\_blobs from sklearn.mixture import GaussianMixture

# Generate sample data

X, \_ = make\_blobs(n\_samples=500, centers=3, random\_state=42)

# Create a Gaussian Mixture Model with 3 components

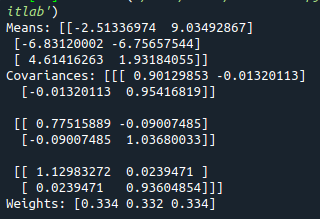
gmm = GaussianMixture(n\_components=3, random\_state=42) # Fit the GMM to the data using the EM algorithm

gmm.fit(X)

# Predict the cluster labels of the data labels = gmm.predict(X)

# Print the parameters learned by the GMM print('Means:', gmm.means\_)

print('Covariances:', gmm.covariances\_) print('Weights:', gmm.weights\_)



# KNN

from sklearn.datasets import load\_iris

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

# Load the iris dataset iris = load\_iris()

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.3, random\_state=42) # Create a k-NN classifier with k=3

knn = KNeighborsClassifier(n\_neighbors=3)

# Fit the classifier to the training data knn.fit(X\_train, y\_train)

# Predict the classes of the test data y\_pred = knn.predict(X\_test)

# Calculate the accuracy of the classifier accuracy = accuracy\_score(y\_test, y\_pred) print('Accuracy:', accuracy)



# non-parametric Locally Weighted Regression algorithm

from sklearn.datasets import load\_boston

from sklearn.neighbors import KNeighborsRegressor from sklearn.model\_selection import train\_test\_split from sklearn.metrics import mean\_squared\_error

# Load the Boston Housing dataset boston = load\_boston()

# Split the data into training and testing sets

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

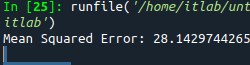
# Create a KNeighborsRegressor with weights='distance' and n\_neighbors=5 lwr = KNeighborsRegressor(weights='distance', n\_neighbors=5)

# Fit the regressor to the training data lwr.fit(X\_train, y\_train)

# Predict the target values of the test data y\_pred = lwr.predict(X\_test)

# Calculate the mean squared error of the regressor mse = mean\_squared\_error(y\_test, y\_pred)

print('Mean Squared Error:', mse)



EX 6

Import pandas as pd

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

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn import metrics

import numpy as np

msg=pd.read\_csv(r'C:\Users\rajd3\Desktop\6-Dataset1.csv',names=['message','label'])

print('The dimensions of the dataset',msg.shape)

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

a=msg['message']

b=msg['labelnum']

X=a[1:-1]

y=b[1:-1]

**#splitting the dataset into train and test data**

xtrain,xtest,ytrain,ytest=train\_test\_split(X,y)

print ('\n the total number of Training Data :',xtrain.shape)

print ('\n the total number of Test Data :',xtest.shape)

**#output the words or Tokens in the text documents**

cv = CountVectorizer()

xtrain\_dtm = cv.fit\_transform(xtrain)

xtest\_dtm=cv.transform(xtest)

print('\n The words or Tokens in the text documents \n')

print(cv.get\_feature\_names\_out())

clf = MultinomialNB()

clf.fit(xtrain\_dtm,ytrain)

predicted = clf.predict(xtest\_dtm)

**#printing accuracy, Confusion matrix, Precision and Recall**

print('\n Accuracy of the classifier is',metrics.accuracy\_score(ytest,predicted))

print('\n Confusion matrix')

print(metrics.confusion\_matrix(ytest,predicted))

print('\n The value of Precision', metrics.precision\_score(ytest,predicted))

print('\n The value of Recall', metrics.recall\_score(ytest,predicted))

**OUTPUT:**

The dimensions of the dataset (19, 2)

the total number of Training Data : (12,)

the total number of Test Data : (5,)

The words or Tokens in the text documents

['am' 'amazing' 'an' 'and' 'awesome' 'bad' 'best' 'boss' 'dance' 'do'

'enemy' 'fun' 'good' 'great' 'have' 'he' 'holiday' 'horrible' 'is' 'like'

'locality' 'love' 'my' 'not' 'of' 'place' 'restaurant' 'sick' 'stay'

'stuff' 'sworn' 'that' 'this' 'to' 'tomorrow' 'tried' 'we' 'what' 'will'

'work']

Accuracy of the classifier is 0.8

Confusion matrix

[[1 1]

[0 3]]

The value of Precision 0.75

The value of Recall 1.0