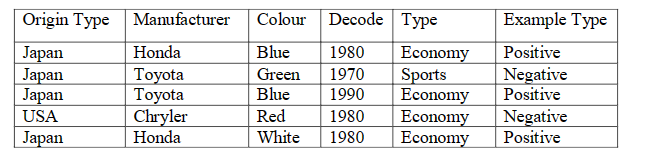
1. **FINDS-ALGORITHM**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Example | Sky | Temp | Humidity | Wind | Water | Forecast | EnjoySport |
| 1 | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| 2 | Sunny | Warm | High | Strong | Warm | Same | Yes |
| 3 | Rainy | Cold | High | Strong | Warm | Change | No |
| 4 | Sunny | Warm | High | Strong | Cool | Change | Yes |

**Dataset:1)** Enjoying sports

**Dataset: 2)** Manifacturers



**Dataset: 3) Shapes (**size,color,shape,class Lable)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Example** | **Size** | **Color** | **Shape** | **Class Label** |
| 1 | Big | Red | Circle | No |
| 2 | Small | Red | Triangle | No |
| 3 | Small | Red | Circle | Yes |
| 4 | Big | Blue | Circle | No |
| 5 | Small | Blue | Circle | Yes |

**PROGRAM:**

import pandas as pd

**# Read the dataset from CSV file**

data = pd.read\_csv(r'C:/Users/Documents/data.csv')

**# Extract the features and target**

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

y = data.iloc[:, -1].values

**# Initialize the hypothesis with the first positive example**

hypothesis = None

for i in range(len(X)):

if y[i] == 'Yes':

hypothesis = list(X[i])

break

**# Refine the hypothesis by checking all positive examples**

for i in range(len(X)):

if y[i] == 'Yes':

for j in range(len(X[i])):

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

hypothesis[j] = '?'

**# Print the final hypothesis**

print('The final hypothesis is:', hypothesis)

**OUTPUT:**

The final Hypothesis: ['sunny', 'warm', '?', 'strong', '?', '?']

**\*NOTE: Here I have used the word Yes for Enjoying sports (\*sunny ,warm, normal, strong, warm, same,yes) and shapes (\*Big,Red,Circle,No) dataset.For manufacturer (\*Japan, Honda, Blue, 1980, Economy , Positive ) dataset use the word ‘Positive’ instead of ‘Yes’.**

**2)CANDITATE-ELIMINATION ALGORITHM**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1. Example | Sky | Temp | Humidity | Wind | Water | Forecast | EnjoySport |
| 1 | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| 2 | Sunny | Warm | High | Strong | Warm | Same | Yes |
| 3 | Rainy | Cold | High | Strong | Warm | Change | No |
| 4 | Sunny | Warm | High | Strong | Cool | Change | Yes |

**Dataset:1)** Enjoying sports

**Dataset: 2) Shapes (**size,color,shape,class Lable)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Example** | **Size** | **Color** | **Shape** | **Class Label** |
| 1 | Big | Red | Circle | No |
| 2 | Small | Red | Triangle | No |
| 3 | Small | Red | Circle | Yes |
| 4 | Big | Blue | Circle | No |
| 5 | Small | Blue | Circle | Yes |

**PROGRAM:**

import numpy as np

**# Define the training examples**

training\_examples = [

(['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same'], 'Yes'),

(['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same'], 'Yes'),

(['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change'], 'No'),

(['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change'], 'Yes'),

]

def candidate\_elimination(examples):

specific\_hypothesis = ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

general\_hypothesis = ['?','?','?','?','?','?']

for x, y in examples:

if y == 'Yes':

for i in range(len(x)):

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

specific\_hypothesis[i] = '?'

for i in range(len(x)):

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

general\_hypothesis[i] = x[i]

else:

for i in range(len(x)):

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

general\_hypothesis[i] = '?'

else :

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

print(f'Specific hypothesis: {specific\_hypothesis}')

print(f'General hypothesis: {general\_hypothesis}\n')

candidate\_elimination(training\_examples)

**OUTPUT:**

Specific hypothesis: ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

General hypothesis: ['?', '?', '?', '?', '?', '?']

Specific hypothesis: ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

General hypothesis: ['?', '?', '?', '?', '?', '?']

Specific hypothesis: ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

General hypothesis: ['?', '?', '?', 'Strong', 'Warm', '?']

Specific hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']

General hypothesis: ['?', '?', '?', 'Strong', 'Warm', '?']

**\*NOTE: Here I enter the datas of enjoying sports dataset directly to the program.for shapes re-enter the iths respective datas and proceed the program.**

**3) ID3/DECISION TREE CLASSIFIER (with iris dataset)**

**PROGRAM:**

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))

**OUTPUT:**

Accuracy: 0.9333333333333333

**ID3/DECISION TREE CLASSIFIER:(When dataset is given in the question paper)**

**Datasets: 1)** Covid infection problem

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Fever | Cough | Breathing  Issues | Infected |
| 1 | NO | NO | NO | NO |
| 2 | YES | YES | YES | YES |
| 3 | YES | YES | NO | NO |
| 4 | YES | NO | YES | YES |
| 5 | YES | YES | YES | YES |
| 6 | NO | YES | NO | NO |
| 7 | YES | NO | YES | YES |
| 8 | YES | NO | YES | YES |
| 9 | NO | YES | YES | YES |
| 10 | YES | YES | NO | YES |
| 11 | NO | YES | NO | NO |
| 12 | NO | YES | YES | YES |
| 13 | NO | YES | YES | NO |
| 14 | YES | YES | NO | NO |

**Datasets: 2)** tennis in the given condition

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Outlook | Temperature | Humidity | Windy | PlayTennis |
| Sunny | Hot | High | FALSE | No |
| Sunny | Hot | High | TRUE | No |
| Overcast | Hot | High | FALSE | Yes |
| Rainy | Mild | High | FALSE | Yes |
| Rainy | Cool | Normal | FALSE | Yes |
| Rainy | Cool | Normal | TRUE | No |
| Overcast | Cool | Normal | TRUE | Yes |
| Sunny | Mild | High | FALSE | No |
| Sunny | Cool | Normal | FALSE | Yes |
| Rainy | Mild | Normal | FALSE | Yes |
| Sunny | Mild | Normal | TRUE | Yes |
| Overcast | Mild | High | TRUE | Yes |
| Overcast | Hot | Normal | FALSE | Yes |
| Rainy | Mild | High | TRUE | No |

**PROGRAM:**

import pandas as pd

import numpy as np

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

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

x = dataset.iloc[:, [1,2,3]].values

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

ct= ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0,1,2])], remainder='passthrough')

x = np.array(ct.fit\_transform(x))

#print(x)

#print(y)

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

clf = DecisionTreeClassifier(criterion='entropy')

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

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

print(y\_pred)

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

**OUTPUT:**

['YES' 'YES' 'NO']

Accuracy: 0.3333333333333333

**\*NOTE: The above program is for the covid infection dataset,for playing tennis dataset the below line must be changer(line for reading the value of x from the dataset).The change is nothing but the index of first column ‘0’ is added with it.The modified line for playing tennis dataset is:**

**x = dataset.iloc[:, [0,1,2,3]].values (the same program can run after changing this line)**

**4)** **Artificial Neural Network by implementing the Back propagation algorithm.**

**PROGRAM:**

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}")

**OUTPUT:**

Accuracy: 0.865

**5)** **naïve Bayesian classifier (with iris dataset)**

**PROGRAM:**

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))

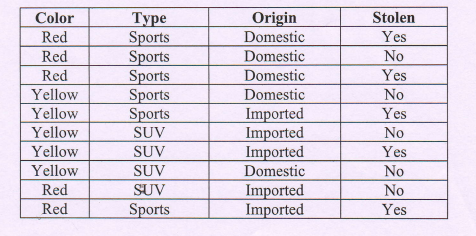
**OUTPUT:**

[2 2 1 0 1 1 0 0 1 2 0 1 2 1 2 1 2 0 0 1 0 1 0 0 1 2 0 1 2 0]

ACCURACY 1.0

**Naïve Bayesian Classifier: (When dataset is given in the question paper)**

**Datasets: 1)** Stolen Vehicles(cars



**Datasets: 2)**Playing tennis on the given condition

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Outlook | Temperature | Humidity | Windy | PlayTennis |
| Sunny | Hot | High | FALSE | No |
| Sunny | Hot | High | TRUE | No |
| Overcast | Hot | High | FALSE | Yes |
| Rainy | Mild | High | FALSE | Yes |
| Rainy | Cool | Normal | FALSE | Yes |
| Rainy | Cool | Normal | TRUE | No |
| Overcast | Cool | Normal | TRUE | Yes |
| Sunny | Mild | High | FALSE | No |
| Sunny | Cool | Normal | FALSE | Yes |
| Rainy | Mild | Normal | FALSE | Yes |
| Sunny | Mild | Normal | TRUE | Yes |
| Overcast | Mild | High | TRUE | Yes |
| Overcast | Hot | Normal | FALSE | Yes |
| Rainy | Mild | High | TRUE | No |

**PROGRAM:**

import pandas as pd

import numpy as np

from sklearn.naive\_bayes import GaussianNB

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

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

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

x = dataset.iloc[:, [0,1,2,3]].values

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

ct= ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0,1,2])], remainder='passthrough')

x = np.array(ct.fit\_transform(x))

#print(x)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size =0.3, random\_state =0)

gnb= GaussianNB()

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

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

print(y\_pred)

print('Accuracy=',gnb.score(x\_test,y\_test))

**OUTPUT:**

['No' 'Yes' 'No' 'Yes' 'Yes']

Accuracy= 0.6

**\*NOTE: The above program is for the playing tennis dataset,for playing Stolen vehicle dataset the below line must be changer(line for reading the value of x and y from the dataset).The change is nothing but the index of fifth column ‘4’ was removed from y and index of forth column ‘3’ was removed from x and added with y.The modified line for playing tennis dataset is:**

**x = dataset.iloc[:, [0,1,2]].values**

**y = dataset.iloc[:, 3].values**

**(the same program can run after changing this lines)**

**6)-naïve Bayesian classifier-II**

**DATASET:**

|  |  |
| --- | --- |
| message | labelnum |
| I love this sandwich | pos |
| This is an amazing place | pos |
| I feel very good about these beers | pos |
| This is my best work | pos |
| What an awesome view | pos |
| I do not like this restaurant | neg |
| I am tried of this stuff | neg |
| I can't deal with this | neg |
| He is my sworn enemy | neg |
| My boss is horrible | neg |
| This is an awesome place | pos |
| I do not like the taste of this juice | neg |
| I love to dance | pos |
| I am sick and tried of this place | neg |
| What a great holiday | pos |
| That is a bad locality to stay | neg |
| We will have good fun tomorrow | pos |
| I went to my enemy's house today | neg |

**PROGRAM:**

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

**7) 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)

**8) EM algorithm**

**PROGRAM:**

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\_)

**OUTPUT:**

Means: [[-2.51336974 9.03492867]

[-6.83120002 -6.75657544]

[ 4.61416263 1.93184055]]

Covariances: [[[ 0.90129853 -0.01320113]

[-0.01320113 0.95416819]]

[[ 0.77515889 -0.09007485]

[-0.09007485 1.03680033]]

[[ 1.12983272 0.0239471 ]

[ 0.0239471 0.93604854]]]

Weights: [0.334 0.332 0.334]

**9)-KNN(with iris dataset)**

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)

**OUTPUT:**

Accuracy: 1.0

**KNN** **(When dataset is given in the question paper)**

**Datasets: 1)** Angelina sports dataset

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Age | Gender | Sport |
| Ajay | 32 | M | Football |
| Mark | 40 | M | Neither |
| Sara | 16 | F | Cricket |
| Zaira | 34 | F | Cricket |
| Sachin | 55 | M | Neither |
| Rahul | 40 | M | Cricket |
| Pooja | 20 | F | Neither |
| Smith | 15 | M | Cricket |
| Lakshmi | 55 | F | Football |
| Michael | 15 | M | Football |
| Angelina | 5 | F | ? |

**NOTE:\*\***While entering the value of gender in csv file assign **M=1** and **F=0**

**PROGRAM:**

import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

**# Define the dataset**

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

X = dataset.iloc[:, [1,2]].values

y = dataset.iloc[:, 3].values

#print(X)

#print(y)

**# Reshape the input data to have 2 dimensions**

X = X.reshape(-1, 2)

**# Create the KNN model**

model = KNeighborsClassifier(n\_neighbors=3)

model.fit(X, y)

**# Use the model to predict the sport for a new data point**

new\_data = np.array([[5,0]])

sport = model.predict(new\_data)

print("Angelina used to play:",sport)

**# Predict the classes of the test data**

y\_pred = model.predict(X)

**# Calculate the accuracy of the classifier**

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

print('Accuracy:', accuracy)

**OUTPUT:**

Angelina used to play: ['Cricket']

Accuracy: 0.5

**\*NOTE: Here I have used M=1 and F=0 on the dataset to perform the operation as we doing in theory exams.(changes need to be done in the datasets itself)**

**10) non-parametric Locally Weighted Regression algorithm**

**PROGRAM:**

from sklearn.datasets import load\_diabetes

from sklearn.neighbors import KNeighborsRegressor

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

from sklearn.metrics import mean\_squared\_error

**# Load the diabetes dataset**

diabetes = load\_diabetes()

**# Split the data into training and testing sets**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(diabetes.data, diabetes.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)

**OUTPUT:**

Mean Squared Error: 3190.614716201732