# prgm-1

import csv

hypo = ['%', '%', '%', '%', '%', '%']

with open('enjoysport.csv', 'r') as csvfile:

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

    data = []

    for line in lines:

        if line[len(line) - 1].upper() == 'YES':

            data.append(line[:-1])

print("Positive eg are: ")

for line in data:

    print(line)

print("Steps of candidate elimination algo are: ")

print(hypo)

hypo = data[0]

for i in range(len(data)):

    for j in range(len(data[0])):

        if hypo[j] != data[i][j]:

            hypo[j] = '?'

    print(hypo)

print("The maximally specific Find-s hypothesis for the given training examples is")

print(hypo)

# prgm-2

import numpy as np

import pandas as pd

data = pd.DataFrame(data=pd.read\_csv(r"candidate.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 and general\_h")

    print(specific\_h)

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

    print(general\_h)

    for i, h in enumerate(concepts):

        if target[i] == "Yes":

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

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

                    specific\_h[x] = '?'

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

        if target[i] == "No":

            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(" steps of Candidate Elimination Algorithm", i + 1)

        print(specific\_h)

        print(general\_h)

    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")

# prgm-3

import pandas as pd

import numpy as np

from sklearn.datasets import load\_iris

from sklearn import tree

iris = load\_iris()

print(iris.feature\_names)

print(iris.target\_names)

removed = [0, 50, 100]

new\_target = np.delete(iris.target, removed)

new\_data = np.delete(iris.data, removed, axis=0)

clf = tree.DecisionTreeClassifier()

clf = clf.fit(new\_data, new\_target)

prediction = clf.predict(iris.data[removed])

print("Original Labels", iris.target[removed])

print("Labels Predicted", prediction)

tree.plot\_tree(clf)

# prgm-4

import numpy as np

class NeuralNetwork:

    def \_\_init\_\_(self, input\_size, hidden\_size, output\_size, learning\_rate):

        self.input\_size = input\_size

        self.hidden\_size = hidden\_size

        self.output\_size = output\_size

        self.learning\_rate = learning\_rate

        self.weights\_input\_hidden = np.random.randn(self.input\_size, self.hidden\_size)

        self.bias\_input\_hidden = np.random.randn(1, self.hidden\_size)

        self.weights\_hidden\_output = np.random.randn(self.hidden\_size, self.output\_size)

        self.bias\_hidden\_output = np.random.randn(1, self.output\_size)

    def sigmoid(self, x):

        return 1 / (1 + np.exp(-x))

    def sigmoid\_derivative(self, x):

        return x \* (1 - x)

    def forward(self, inputs):

        self.hidden\_output = self.sigmoid(np.dot(inputs, self.weights\_input\_hidden) + self.bias\_input\_hidden)

        self.output = self.sigmoid(np.dot(self.hidden\_output, self.weights\_hidden\_output) + self.bias\_hidden\_output)

    def backward(self, inputs, targets):

        output\_error = targets - self.output

        output\_delta = output\_error \* self.sigmoid\_derivative(self.output)

        hidden\_error = output\_delta.dot(self.weights\_hidden\_output.T)

        hidden\_delta = hidden\_error \* self.sigmoid\_derivative(self.hidden\_output)

        self.weights\_hidden\_output += self.hidden\_output.T.dot(output\_delta) \* self.learning\_rate

        self.bias\_hidden\_output += np.sum(output\_delta, axis=0, keepdims=True) \* self.learning\_rate

        self.weights\_input\_hidden += inputs.T.dot(hidden\_delta) \* self.learning\_rate

        self.bias\_input\_hidden += np.sum(hidden\_delta, axis=0, keepdims=True) \* self.learning\_rate

    def train(self, inputs, targets, epochs):

        for epoch in range(epochs):

            self.forward(inputs)

            self.backward(inputs, targets)

    def predict(self, inputs):

        self.forward(inputs)

        return self.output

inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

targets = np.array([[0], [1], [1], [0]])

input\_size = 2

hidden\_size = 3

output\_size = 1

learning\_rate = 0.1

epochs = 10000

nn = NeuralNetwork(input\_size, hidden\_size, output\_size, learning\_rate)

nn.train(inputs, targets, epochs)

predictions = nn.predict(inputs)

print("Predictions:")

print(predictions)

# prgm-5

from sklearn.datasets import fetch\_20newsgroups

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

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

newsgroups\_train = fetch\_20newsgroups(subset='train')

newsgroups\_test = fetch\_20newsgroups(subset='test')

vectorizer = CountVectorizer()

X\_train = vectorizer.fit\_transform(newsgroups\_train.data)

X\_test = vectorizer.transform(newsgroups\_test.data)

y\_train = newsgroups\_train.target

y\_test = newsgroups\_test.target

nb\_classifier = MultinomialNB()

nb\_classifier.fit(X\_train, y\_train)

y\_pred = nb\_classifier.predict(X\_test)

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

precision = precision\_score(y\_test, y\_pred, average='weighted')

recall = recall\_score(y\_test, y\_pred, average='weighted')

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

# prgm-6

import bayespy as bp

import numpy as np

import csv

from colorama import init

from colorama import Fore, Back, Style

init()

ageEnum = {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4}

genderEnum = {'Male': 0, 'Female': 1}

familyHistoryEnum = {'Yes': 0, 'No': 1}

dietEnum = {'High': 0, 'Medium': 1, 'Low': 2}

lifeStyleEnum = {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}

cholesterolEnum = {'High': 0, 'BorderLine': 1, 'Normal': 2}

heartDiseaseEnum = {'Yes': 0, 'No': 1}

with open('heart\_disease\_data.csv') as csvfile:

    lines = csv.reader(csvfile)

    dataset = list(lines)

    data = []

    for x in dataset:

        data.append([ageEnum[x[0]], genderEnum[x[1]], familyHistoryEnum[x[2]], dietEnum[x[3]], lifeStyleEnum[x[4]], cholesterolEnum[x[5]], heartDiseaseEnum[x[6]]])

data = np.array(data)

N = len(data)

p\_age = bp.nodes.Dirichlet(1.0 \* np.ones(5))

age = bp.nodes.Categorical(p\_age, plates=(N,))

age.observe(data[:, 0])

p\_gender = bp.nodes.Dirichlet(1.0 \* np.ones(2))

gender = bp.nodes.Categorical(p\_gender, plates=(N,))

gender.observe(data[:, 1])

p\_familyhistory = bp.nodes.Dirichlet(1.0 \* np.ones(2))

familyhistory = bp.nodes.Categorical(p\_familyhistory, plates=(N,))

familyhistory.observe(data[:, 2])

p\_diet = bp.nodes.Dirichlet(1.0 \* np.ones(3))

diet = bp.nodes.Categorical(p\_diet, plates=(N,))

diet.observe(data[:, 3])

p\_lifestyle = bp.nodes.Dirichlet(1.0 \* np.ones(4))

lifestyle = bp.nodes.Categorical(p\_lifestyle, plates=(N,))

lifestyle.observe(data[:, 4])

p\_cholesterol = bp.nodes.Dirichlet(1.0 \* np.ones(3))

cholesterol = bp.nodes.Categorical(p\_cholesterol, plates=(N,))

cholesterol.observe(data[:, 5])

p\_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))

heartdisease = bp.nodes.MultiMixture([age, gender, familyhistory, diet, lifestyle, cholesterol], bp.nodes.Categorical, p\_heartdisease)

heartdisease.observe(data[:, 6])

p\_heartdisease.update()

m = 0

while m == 0:

    print("\n")

    res = bp.nodes.MultiMixture(

        [int(input('Enter Age: ' + str(ageEnum))), int(input('Enter Gender: ' + str(genderEnum))), int(input('Enter FamilyHistory: ' + str(familyHistoryEnum))), int(input('Enter dietEnum: ' + str(dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter Cholesterol: ' + str(cholesterolEnum)))],

        bp.nodes.Categorical,

        p\_heartdisease

    ).get\_moments()[0][heartDiseaseEnum['Yes']]

    print("Probability(HeartDisease) = " + str(res))

    m = int(input("Enter for Continue: 0, Exit: 1 "))

# prgm-7

import numpy as np

import pandas as pd

np.random.seed(0)

num\_samples = 100

num\_features = 2

data = np.random.rand(num\_samples, num\_features)

df = pd.DataFrame(data, columns=['X', 'Y'])

df.to\_csv('test\_data.csv', index=False)

print("CSV file 'test\_data.csv' has been generated successfully.")

from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

import matplotlib.pyplot as plt

data = pd.read\_csv('test\_data.csv')

X = data.values

k = 3

kmeans = KMeans(n\_clusters=k)

kmeans.fit(X)

kmeans\_labels = kmeans.labels\_

kmeans\_centers = kmeans.cluster\_centers\_

em = GaussianMixture(n\_components=k)

em.fit(X)

em\_labels = em.predict(X)

em\_centers = em.means\_

print("K-means labels:")

print(kmeans\_labels)

print("EM labels:")

print(em\_labels)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.scatter(X[:, 0], X[:, 1], c=kmeans\_labels, cmap='viridis')

plt.scatter(kmeans\_centers[:, 0], kmeans\_centers[:, 1], marker='\*', s=300, c='r')

plt.title('K-means Clustering')

plt.subplot(1, 2, 2)

plt.scatter(X[:, 0], X[:, 1], c=em\_labels, cmap='viridis')

plt.scatter(em\_centers[:, 0], em\_centers[:, 1], marker='\*', s=300, c='r')

plt.title('EM Clustering')

plt.show()

# prgm-8

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

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

from sklearn.neighbors import KNeighborsClassifier

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

iris = load\_iris()

X = iris.data

y = iris.target

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

k = 3

knn = KNeighborsClassifier(n\_neighbors=k)

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

predictions = knn.predict(X\_test)

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

print("Accuracy:", accuracy)

correct\_predictions = []

wrong\_predictions = []

for i in range(len(predictions)):

    if predictions[i] == y\_test[i]:

        correct\_predictions.append((X\_test[i], y\_test[i], predictions[i]))

    else:

        wrong\_predictions.append((X\_test[i], y\_test[i], predictions[i]))

print("\nCorrect Predictions:")

for prediction in correct\_predictions:

    print("Input:", prediction[0], "Actual Class:", iris.target\_names[prediction[1]], "Predicted Class:", iris.target\_names[prediction[2]])

print("\nWrong Predictions:")

for prediction in wrong\_predictions:

    print("Input:", prediction[0], "Actual Class:", iris.target\_names[prediction[1]], "Predicted Class:", iris.target\_names[prediction[2]])

conf\_matrix = confusion\_matrix(y\_test, predictions)

print("\nConfusion Matrix:")

print(conf\_matrix)

# prgm-9

import numpy as np

import matplotlib.pyplot as plt

def locally\_weighted\_regression(x\_query, X\_train, y\_train, tau=0.1):

    m = X\_train.shape[0]

    weights = np.exp(-np.sum((X\_train - x\_query) \*\* 2, axis=1) / (2 \* tau \* tau))

    W = np.diag(weights)

    theta = np.linalg.inv(X\_train.T @ W @ X\_train) @ (X\_train.T @ W @ y\_train)

    prediction = x\_query @ theta

    return prediction

np.random.seed(0)

X\_train = np.linspace(0, 10, 50)

y\_train = np.sin(X\_train) + np.random.normal(0, 0.1, X\_train.shape[0])

X\_query = np.linspace(0, 10, 100)

tau = 0.5

predictions = []

for xq in X\_query:

    x\_query = np.array([1, xq])

    prediction = locally\_weighted\_regression(x\_query, np.c\_[np.ones(X\_train.shape[0]), X\_train], y\_train, tau)

    predictions.append(prediction)

plt.figure(figsize=(10, 6))

plt.scatter(X\_train, y\_train, color='blue', label='Training data')

plt.plot(X\_query, predictions, color='red', label='Locally Weighted Regression')

plt.xlabel('X')

plt.ylabel('Y')

plt.title('Locally Weighted Regression')

plt.legend()

plt.grid(True)

plt.show()

# PRGM-10

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

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

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

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

iris = datasets.load\_iris()

X = iris.data

y = iris.target

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

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

svm\_classifier = SVC(kernel='linear', C=1.0, random\_state=42)

svm\_classifier.fit(X\_train, y\_train)

y\_pred = svm\_classifier.predict(X\_test)

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

print("Accuracy:", accuracy)

print("\nClassification Report:")

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