**Ex.No:1**

Aim: Illustrate and Demonstrate the working model and principle of Find-S algorithm.

Program: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples

***FIND-S Algorithm***

1. Initialize h to the most specific hypothesis in H

2. For each positive training instance x

For each attribute constraint ai in h

If the constraint ai is satisfied by x

Then do nothing

Else replace ai in h by the next more general constraint that is satisfied by x

3. Output hypothesis h

***Training Examples:***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Example** | **Sky** | **AirTemp** | **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 |

***Code:***

import csv

a = []

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

for row in csv.reader(csvfile):

a.append(row)

print(a)

print("\n The total number of training instances are : ",len(a))

num\_attribute = len(a[0])-1

print("\n The initial hypothesis is : ")

hypothesis = ['0']\*num\_attribute

print(hypothesis)

for i in range(0, len(a)):

if a[i][num\_attribute] == 'yes':

for j in range(0, num\_attribute):

if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:

hypothesis[j] = a[i][j]

else:

hypothesis[j] = '?'

print("\n The hypothesis for the training instance {} is : \n" .format(i+1),hypothesis)

print("\n The Maximally specific hypothesis for the training instance is ")

print(hypothesis)

**Ex.No:2**

Aim: Demonstrate the working model and principle of candidate elimination algorithm.

Program: 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.

***Candidate Elimination Algorithm***

Initialize G to the set of maximally general hypotheses in H

Initialize S to the set of maximally specific hypotheses in H

For each training example d, do

• If d is a positive example

• Remove from G any hypothesis inconsistent with d

• For each hypothesis s in S that is not consistent with d

• Remove s from S

• Add to S all minimal generalizations h of s such that

• h is consistent with d, and some member of G is more general than h

• Remove from S any hypothesis that is more general than another hypothesis in S

• If d is a negative example

• Remove from S any hypothesis inconsistent with d

• For each hypothesis g in G that is not consistent with d

• Remove g from G

• Add to G all minimal specializations h of g such that

• h is consistent with d, and some member of S is more specific than h

• Remove from G any hypothesis that is less general than another hypothesis in G

***Training Examples:***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Example** | **Sky** | **AirTemp** | **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 |

**CODE:**

import numpy as np

import pandas as pd

data = pd.read\_csv(r'D:\github\ML\ML-LAB\program-1\ENJOYSPORT.csv')

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

print("\nInstances are:\n",concepts)

target = np.array(data.iloc[:,-1],dtype=np.int64)

print("\nTarget Values are: ",target)

def learn(concepts, target):

specific\_h = concepts[0].copy()

print("\nInitialization of specific\_h and genearal\_h")

print("\nSpecific Boundary: ", specific\_h)

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

print("\nGeneric Boundary: ",general\_h)

for i, h in enumerate(concepts):

print("\nInstance", i+1 , "is ", h)

if target[i] == 1:

print("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] == 0:

print("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("Specific Bundary after ", i+1, "Instance is ", specific\_h)

print("Generic Boundary after ", i+1, "Instance is ", general\_h)

print("\n")

learn(concepts, target)

**Ex.No:3**

Aim: To construct the Decision tree using the training data sets under supervised learning concept.

Program: 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 this knowledge to classify a new sample.

**ID3 Algorithm:**

Calculate entropy for dataset.

For each attribute/feature.

Calculate entropy for all its categorical values.

Calculate information gain for the feature.

Find the feature with maximum information gain.

4. Repeat it until we get the desired tree.

**Code:**

from sklearn.datasets import load\_iris

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

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

import matplotlib.pyplot as plt

data = load\_iris()

X = data.data # Features

y = data.target # Target variable

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

clf = DecisionTreeClassifier()

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

plt.figure(figsize=(12,8))

tree.plot\_tree(clf, feature\_names=data.feature\_names, class\_names=data.target\_names, filled=True)

plt.show()

score = clf.score(X\_test, y\_test)

print(f"Model accuracy: {score:.2f}")

**Ex.No:4**

Aim: To understand the working principle of Artificial Neural network with feed forward and feed backward principle.

Program: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

STEPS:

**1. Feedforward Phase**

In the feedforward phase, the input data is passed through the network layer by layer, from the input layer to the output layer. The main steps are:

**Steps in Feedforward Phase:**

1. **Input Layer:**
   * The input features are fed into the input layer, where each neuron in this layer corresponds to a feature.
2. **Hidden Layers:**
   * The input is then passed to the hidden layers. Each neuron in a hidden layer receives input from all neurons in the previous layer.
   * Each neuron performs a weighted sum of its inputs and then applies an activation function (e.g., ReLU, Sigmoid) to introduce non-linearity.
3. **Output Layer:**
   * Finally, the processed data reaches the output layer, where each neuron represents a possible output. The output is calculated similarly using a weighted sum and an activation function (e.g., Softmax for classification tasks).

**2. Backpropagation Phase**

Backpropagation is the learning phase where the network adjusts its weights and biases to minimize the error between the predicted output and the actual output. This phase uses the gradient descent algorithm to update the weights.

**Steps in Backpropagation Phase:**

1. **Calculate the Error:**
   * Compute the error at the output layer (e.g., using Mean Squared Error for regression or Cross-Entropy Loss for classification).
2. **Compute Gradients:**
   * Calculate the gradient of the error with respect to each weight using the chain rule of calculus. This involves computing partial derivatives of the error with respect to the weights.
3. **Update Weights and Biases:**
   * Adjust the weights and biases in the opposite direction of the gradient to reduce the error. This update is typically controlled by a learning rate, which determines the size of the adjustments.

**CODE:**

import numpy as np

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

return x \* (1 - x)

def feedforward(X, weights1, bias1, weights2, bias2):

hidden\_input = np.dot(X, weights1) + bias1

hidden\_output = sigmoid(hidden\_input)

final\_input = np.dot(hidden\_output, weights2) + bias2

final\_output = sigmoid(final\_input)

return hidden\_output, final\_output

def backpropagation(X, y, weights1, bias1, weights2, bias2, hidden\_output, final\_output, learning\_rate):

# Calculate the error

error = y - final\_output

d\_final\_output = error \* sigmoid\_derivative(final\_output)

# Backpropagate the error

error\_hidden\_layer = d\_final\_output.dot(weights2.T)

d\_hidden\_output = error\_hidden\_layer \* sigmoid\_derivative(hidden\_output)

weights2 += hidden\_output.T.dot(d\_final\_output) \* learning\_rate

bias2 += np.sum(d\_final\_output, axis=0, keepdims=True) \* learning\_rate

weights1 += X.T.dot(d\_hidden\_output) \* learning\_rate

bias1 += np.sum(d\_hidden\_output, axis=0, keepdims=True) \* learning\_rate

return weights1, bias1, weights2, bias2

def train(X, y, weights1, bias1, weights2, bias2, learning\_rate, epochs):

for epoch in range(epochs):

hidden\_output, final\_output = feedforward(X, weights1, bias1, weights2, bias2)

weights1, bias1, weights2, bias2 = backpropagation(X, y, weights1, bias1, weights2, bias2, hidden\_output, final\_output, learning\_rate)

if epoch % 1000 == 0:

loss = np.mean((y - final\_output) \*\* 2)

print(f'Epoch {epoch}, Loss: {loss}')

return weights1, bias1, weights2, bias2

if \_\_name\_\_ == "\_\_main\_\_":

# Input data (XOR problem)

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

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

np.random.seed(42)

input\_layer\_neurons = 2

hidden\_layer\_neurons = 2

output\_layer\_neurons = 1

weights1 = np.random.uniform(size=(input\_layer\_neurons, hidden\_layer\_neurons))

bias1 = np.random.uniform(size=(1, hidden\_layer\_neurons))

weights2 = np.random.uniform(size=(hidden\_layer\_neurons, output\_layer\_neurons))

bias2 = np.random.uniform(size=(1, output\_layer\_neurons))

learning\_rate = 0.1

epochs = 10000

weights1, bias1, weights2, bias2 = train(X, y, weights1, bias1, weights2, bias2, learning\_rate, epochs)

\_, final\_output = feedforward(X, weights1, bias1, weights2, bias2)

print("Predicted Output:")

print(final\_output)

print("Actual Output:")

print(y)

Ex.No:5

Aim: Demonstrate the text classifier using Naïve bayes classifier algorithm.

Program: Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Steps:

Step 1 – Collect raw data

Step 2 – Convert data to a frequency table(s)

Step 3 – Calculate prior probability and evidence

Step 4 – Apply probabilities to Bayes’ Theorem equation

**CODE:**

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

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

print(data.head())

X = data.drop('target', axis=1) # Replace 'target' with the name of your target column

y = data['target'] # Replace 'target' with the name of your target column

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

model = GaussianNB()

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

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

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

print(f'Accuracy: {accuracy \* 100:.2f}%')

Ex.No:6

CODE:

import pandas as pd

from pgmpy.models import BayesianNetwork

from pgmpy.factors.discrete import TabularCPD

from pgmpy.inference import VariableElimination

# Step 1: Define the structure of the network

model = BayesianNetwork([('Rain', 'WetGrass'), ('Sprinkler', 'WetGrass'), ('Rain', 'Sprinkler')])

# Step 2: Define the CPDs

cpd\_rain = TabularCPD(variable='Rain', variable\_card=2, values=[[0.7], [0.3]])

cpd\_sprinkler = TabularCPD(variable='Sprinkler', variable\_card=2,

values=[[0.5, 0.9], [0.5, 0.1]],

evidence=['Rain'], evidence\_card=[2])

cpd\_wet\_grass = TabularCPD(variable='WetGrass', variable\_card=2,

values=[[0.99, 0.9, 0.8, 0.0],

[0.01, 0.1, 0.2, 1.0]],

evidence=['Sprinkler', 'Rain'], evidence\_card=[2, 2])

# Add the CPDs to the model

model.add\_cpds(cpd\_rain, cpd\_sprinkler, cpd\_wet\_grass)

# Step 3: Check if the model is valid

assert model.check\_model()

# Step 4: Perform inference

infer = VariableElimination(model)

# Query the probability of Wet Grass being True given that Rain is True

prob\_wet\_grass\_given\_rain = infer.query(variables=['WetGrass'], evidence={'Rain': 1})

print("P(WetGrass | Rain):")

print(prob\_wet\_grass\_given\_rain)

# Query the probability of Rain being True given that Wet Grass is True

prob\_rain\_given\_wet\_grass = infer.query(variables=['Rain'], evidence={'WetGrass': 1})

print("P(Rain | WetGrass):")

print(prob\_rain\_given\_wet\_grass)

Ex.No7

CODE;

Aim: Implement and demonstrate the working model of K-means clustering algorithm with Expectation Maximization Concept.

Program: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Python ML library classes/API in the program.

CODE:

# K- Means Clustering

from sklearn.cluster import KMeans

from sklearn import preprocessing

from sklearn.mixture import GaussianMixture

from sklearn.datasets import load\_iris

# import sklearn.metrics as sm

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

dataset=load\_iris()

# print(dataset)

X=pd.DataFrame(dataset.data) # Extract data from dataset

X.columns=['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width'] # concepts

y=pd.DataFrame(dataset.target)

y.columns=['Targets'] # target funciton

# print(X)

plt.figure(figsize=(14,7)) # size of the plot

colormap=np.array(['red','lime','black']) # colors to plot

# REAL PLOT

plt.subplot(1,3,1) # index in the row of plot (rows, columns, index)

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y.Targets],s=40) # Scatter plot

plt.title('Real')

# K-PLOT

plt.subplot(1,3,2)

model=KMeans(n\_clusters=3)

model.fit(X)

predY=np.choose(model.labels\_,[0,1,2]).astype(np.int64)

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[predY],s=40)

plt.title('KMeans')

# GMM PLOT

scaler=preprocessing.StandardScaler()

xsa=scaler.fit\_transform(X)

xs=pd.DataFrame(xsa,columns=X.columns)

gmm=GaussianMixture(n\_components=3)

gmm.fit(xs)

y\_cluster\_gmm=gmm.predict(xs)

plt.subplot(1,3,3)

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y\_cluster\_gmm],s=40)

plt.title('GMM Classification')

Ex.No:8

Aim: Demonstrate and analyse the results of classification based on KNN Algorithm.

Program: Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

CODE:

from sklearn.datasets import load\_iris

from sklearn.neighbors import KNeighborsClassifier

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

import numpy as np

dataset=load\_iris()

#print(dataset)

X\_train,X\_test,y\_train,y\_test=train\_test\_split(dataset["data"],dataset["target"],random\_state=0)

kn=KNeighborsClassifier(n\_neighbors=1)

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

for i in range(len(X\_test)):

x=X\_test[i]

x\_new=np.array([x])

prediction=kn.predict(x\_new)

print("TARGET=",y\_test[i],dataset["target\_names"][y\_test[i]],"PREDICTED=",prediction,dataset["target\_names"][prediction])

print(kn.score(X\_test,y\_test))

Ex.No: 9

Aim: Understand and analyse the concept of Regression algorithm techniques.

Program: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

CODE:

from math import ceil

import numpy as np

from scipy import linalg

def lowess(x, y, f, iterations):

n = len(x)

r = int(ceil(f \* n))

h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]

w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)

w = (1 - w \*\* 3) \*\* 3

yest = np.zeros(n)

delta = np.ones(n)

for iteration in range(iterations):

for i in range(n):

weights = delta \* w[:, i]

b = np.array([np.sum(weights \* y), np.sum(weights \* y \* x)])

A = np.array([[np.sum(weights), np.sum(weights \* x)],[np.sum(weights \* x), np.sum(weights \* x \* x)]])

beta = linalg.solve(A, b)

yest[i] = beta[0] + beta[1] \* x[i]

residuals = y - yest

s = np.median(np.abs(residuals))

delta = np.clip(residuals / (6.0 \* s), -1, 1)

delta = (1 - delta \*\* 2) \*\* 2

return yest

import math

n = 100

x = np.linspace(0, 2 \* math.pi, n)

y = np.sin(x) + 0.3 \* np.random.randn(n)

f =0.25

iterations=3

yest = lowess(x, y, f, iterations)

import matplotlib.pyplot as plt

plt.plot(x,y,"r.")

plt.plot(x,yest,"b-")

Ex.No:10

Aim: Implement and demonstrate classification algorithm using Support vector machine Algorithm.

Program: Implement and demonstrate the working of SVM algorithm for classification.

Support Vector Machine Algorithm:

**Step** 1: Load the important libraries. ...

**Step** 2: Import dataset and extract the X variables and Y separately. ...

**Step** 3: Divide the dataset into train and test. ...

**Step** 4: Initializing the **SVM** classifier model. ...

**Step** 5: Fitting the **SVM** classifier model. ...

**Step** 6: Coming up with predictions.

CODE:

from sklearn.datasets import load\_iris

import pandas as pd

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

from sklearn.svm import SVC

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

import numpy as np

import matplotlib.pyplot as plt

# Load the dataset

iris = load\_iris()

data = pd.DataFrame(iris.data, columns=iris.feature\_names)

data['target'] = iris.target

# Split the data

X = data.drop(columns=['target'])

y = data['target']

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

# Train the SVM model

svm\_model = SVC(kernel='linear') # You can also use 'rbf', 'poly', etc.

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

# Make predictions

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

# Print evaluation metrics

print("Confusion Matrix:")

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

print("\nClassification Report:")

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

print("\nAccuracy Score:")

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

# Visualization using only the first two features

X\_vis = X.iloc[:, :2]

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

svm\_model\_vis = SVC(kernel='linear')

svm\_model\_vis.fit(X\_train\_vis, y\_train\_vis)

def plot\_decision\_boundary(X, y, model):

h = .02 # step size in the mesh

x\_min, x\_max = X.iloc[:, 0].min() - 1, X.iloc[:, 0].max() + 1

y\_min, y\_max = X.iloc[:, 1].min() - 1, X.iloc[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.8)

plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y, edgecolor='k', marker='o')

plt.xlabel('Sepal length')

plt.ylabel('Sepal width')

plt.title('SVM Decision Boundary')

plt.show()

plot\_decision\_boundary(X\_vis, y, svm\_model\_vis)