**Ex. 1 FIND-S algorithm**

**Aim:** 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.

**Algorithm:**

1.Initialize h to the most specific hypothesis in.

2.For each positive training instance x for each attribute constraint a, in h.

3.If the constrain a, is satisfied by x then do nothing.

4.Else,replace a, in h by the next more general constrain that is satisfied by x.

5.Oytput hypothesis h.

**Code:**

import pandas as pd

import numpy as np

#to read the data in the csv file

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

print(data,"n")

#making an array of all the attributes

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

print("n The attributes are: ",d)

#segragating the target that has positive and negative examples

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

print("n The target is: ",target)

#training function to implement find-s algorithm

def train(c,t):

    for i, val in enumerate(t):

        if val == "Yes":

            specific\_hypothesis = c[i].copy()

            break

    for i, val in enumerate(c):

        if t[i] == "Yes":

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

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

                    specific\_hypothesis[x] = '?'

                else:

                    pass

    return specific\_hypothesis

#obtaining the final hypothesis

print("n The final hypothesis is:",train(d,target))

**output**

Sky Temp Humidity Wind Water Forecast EnjoySport

0 1 Sunny Warm Normal Strong Warm Same Yes

1 2 Sunny Warm High Strong Warm Same Yes

2 3 Rainy Cold High Strong Warm Change No

3 4 Sunny Warm High Strong Cool Change Yes n

n The attributes are: [[1 'Sunny ' 'Warm ' 'Normal ' 'Strong ' 'Warm ' 'Same ']

[2 'Sunny ' 'Warm ' 'High ' 'Strong ' 'Warm ' 'Same ']

[3 'Rainy ' 'Cold ' 'High ' 'Strong ' 'Warm ' 'Change ']

[4 'Sunny ' 'Warm ' 'High ' 'Strong ' 'Cool ' 'Change ']]

n The target is: ['Yes' 'Yes' 'No' 'Yes']

n The final hypothesis is: ['?' 'Sunny ' 'Warm ' '?' 'Strong ' '?' '?']

**Ex.2 Candidate-Elimination algorithm**

**Aim:**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.

**Algorithm:**

1.Load data set.

2.Intialiaze general hypothesis and specific hypothesis

3. For each training example

4.If example is positive example

If attribute\_value == hypothesis value:

20 nothing

Else:

Replace otherwise value with’?’

5.If example is negative example model generalize hypothesis more specific.

**Code:**

import numpy as np

import pandas as pd

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

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

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

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

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] == "yes":

            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] == "no":

            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")

    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")

**output:**

Instances are:

[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

['sunny' 'warm' 'high' 'strong' 'warm' 'same']

['rainy' 'cold' 'high' 'strong' 'warm' 'change']

['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Target Values are: ['yes' 'yes' 'no' 'yes']

Initialization of specific\_h and genearal\_h

Specific Boundary: ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Boundary: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Instance is Positive

Specific Bundary after 1 Instance is ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Boundary after 1 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 2 is ['sunny' 'warm' 'high' 'strong' 'warm' 'same']

Instance is Positive

Specific Bundary after 2 Instance is ['sunny' 'warm' '?' 'strong' 'warm' 'same']

Generic Boundary after 2 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 3 is ['rainy' 'cold' 'high' 'strong' 'warm' 'change']

Instance is Negative

Specific Bundary after 3 Instance is ['sunny' 'warm' '?' 'strong' 'warm' 'same']

Generic Boundary after 3 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

Instance 4 is ['sunny' 'warm' 'high' 'strong' 'cool' 'change']

Instance is Positive

Specific Bundary after 4 Instance is ['sunny' 'warm' '?' 'strong' '?' '?']

Generic Boundary after 4 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific\_h:

['sunny' 'warm' '?' 'strong' '?' '?']

Final General\_h:

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

**Ex.3 Working of decision tree based ID3 algorithm**

**Aim:**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.

**Algorithm:**

1.Calculate entropy for the algorithm

2.For each node (i)calculate entropy for all its categorial values.

(ii)calculate information gain for the node.

3.Find the node with the highest information gain dataparticular level.

4.Repeat step from 1 to 3 till.

5.We reach the leaf node and have created our decision tree.

**Code**

import pandas as pd

import math

import numpy as np

data = pd.read\_excel("kk3.xlsx")

features = [feat for feat in data]

features.remove("answer")

class Node:

    def \_\_init\_\_(self):

        self.children = []

        self.value = ""

        self.isLeaf = False

        self.pred = ""

def entropy(examples):

    pos = 0.0

    neg = 0.0

    for \_, row in examples.iterrows():

        if row["answer"] == "yes":

            pos += 1

        else:

            neg += 1

    if pos == 0.0 or neg == 0.0:

        return 0.0

    else:

        p = pos / (pos + neg)

        n = neg / (pos + neg)

        return -(p \* math.log(p, 2) + n \* math.log(n, 2))

def info\_gain(examples, attr):

    uniq = np.unique(examples[attr])

    #print ("\n",uniq)

    gain = entropy(examples)

    #print ("\n",gain)

    for u in uniq:

        subdata = examples[examples[attr] == u]

        #print ("\n",subdata)

        sub\_e = entropy(subdata)

        gain -= (float(len(subdata)) / float(len(examples))) \* sub\_e

        #print ("\n",gain)

    return gain

def ID3(examples, attrs):

    root = Node()

    max\_gain = 0

    max\_feat = ""

    for feature in attrs:

        #print ("\n",examples)

        gain = info\_gain(examples, feature)

        if gain > max\_gain:

            max\_gain = gain

            max\_feat = feature

    root.value = max\_feat

    #print ("\nMax feature attr",max\_feat)

    uniq = np.unique(examples[max\_feat])

    #print ("\n",uniq)

    for u in uniq:

        #print ("\n",u)

        subdata = examples[examples[max\_feat] == u]

        #print ("\n",subdata)

        if entropy(subdata) == 0.0:

            newNode = Node()

            newNode.isLeaf = True

            newNode.value = u

            newNode.pred = np.unique(subdata["answer"])

            root.children.append(newNode)

        else:

            dummyNode = Node()

            dummyNode.value = u

            new\_attrs = attrs.copy()

            new\_attrs.remove(max\_feat)

            child = ID3(subdata, new\_attrs)

            dummyNode.children.append(child)

            root.children.append(dummyNode)

    return root

def printTree(root: Node, depth=0):

    for i in range(depth):

        print("\t", end="")

    print(root.value, end="")

    if root.isLeaf:

        print(" -> ", root.pred)

    print()

    for child in root.children:

        printTree(child, depth + 1)

root = ID3(data, features)

printTree(root)

**output:**

outlook

overcast -> ['yes']

rain

wind

strong -> ['no']

weak -> ['yes']

sunny

humidity

high -> ['no']

normal -> ['yes']

**Ex. 4 Back propagation algorithm**

**Aim:**Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets

**Algorithm:**

1.Inputs x, arrive through the preconnected path.

2.The input is modelled using tree weights W.Weights are usually chosen randomly.

3.Calculate the output of each neuron from the input layer to the hidden layer to the output layer.

4.Calculate the errors in the outputs.

BackPropagation Error = Actual Output – Desired Output

5.From the output layer,go back to the hidden layer to adjust the weights to reduce the error.

6.Repeat the process until the desired output is achieved.

**Code:**

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) #maximum of X array longitudinally

y = y/100

#Sigmoid Function

def sigmoid (x):

    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function

def derivatives\_sigmoid(x):

    return x \* (1 - x)

#Variable initialization

epoch=5 #Setting training iterations

lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set

hiddenlayer\_neurons = 3 #number of hidden layers neurons

output\_neurons = 1 #number of neurons at output layer

#weight and bias initialization

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

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

#draws a random range of numbers uniformly of dim x\*y

for i in range(epoch):

    #Forward Propogation

    hinp1=np.dot(X,wh)

    hinp=hinp1 + bh

    hlayer\_act = sigmoid(hinp)

    outinp1=np.dot(hlayer\_act,wout)

    outinp= outinp1+bout

    output = sigmoid(outinp)

    #Backpropagation

    EO = y-output

    outgrad = derivatives\_sigmoid(output)

    d\_output = EO \* outgrad

    EH = d\_output.dot(wout.T)

    hiddengrad = derivatives\_sigmoid(hlayer\_act)#how much hidden layer wts contributed to error

    d\_hiddenlayer = EH \* hiddengrad

    wout += hlayer\_act.T.dot(d\_output) \*lr   # dotproduct of nextlayererror and currentlayerop

    wh += X.T.dot(d\_hiddenlayer) \*lr

    print ("-----------Epoch-", i+1, "Starts----------")

    print("Input: \n" + str(X))

    print("Actual Output: \n" + str(y))

    print("Predicted Output: \n" ,output)

    print ("-----------Epoch-", i+1, "Ends----------\n")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

**output:**

-----------Epoch- 1 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.81361748]

[0.80545255]

[0.80887549]]

-----------Epoch- 1 Ends----------

-----------Epoch- 2 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.81464174]

[0.80640982]

[0.80987396]]

-----------Epoch- 2 Ends----------

-----------Epoch- 3 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.81564531]

[0.8073482 ]

[0.81085253]]

-----------Epoch- 3 Ends----------

-----------Epoch- 4 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.81662881]

[0.80826822]

[0.81181177]]

-----------Epoch- 4 Ends----------

-----------Epoch- 5 Starts----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.81759282]

[0.80917043]

[0.81275225]]

-----------Epoch- 5 Ends----------

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.81759282]

[0.80917043]

[0.81275225]]

**Ex.5 Naive Bayesian Classifier**

**Aim:** 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.

**Algorithm:**

1.Read the training dataset.

2.Calculate the mean and standard derivation of the predictor variables in each class.

3.Repeat until the probability of all prediction variables

4.Calculate the likelihood for each class.

5.Get the greatest likelihood.

**Code:**

# importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

# importing the dataset

dataset = pd.read\_csv("Naive-Bayes-Classification-Data.csv")

# split the data into inputs and outputs

X = dataset.iloc[:, [0,1]].values

y = dataset.iloc[:, 2].values

# training and testing data

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

# assign test data size 25%

X\_train, X\_test, y\_train, y\_test =train\_test\_split(X,y,test\_size= 0.25, random\_state=0)

# importing standard scaler

from sklearn.preprocessing import StandardScaler

# scalling the input data

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.fit\_transform(X\_test)

# importing classifier

from sklearn.naive\_bayes import BernoulliNB

# import Gaussian Naive Bayes classifier

from sklearn.naive\_bayes import GaussianNB

# create a Gaussian Classifier

classifer1 = GaussianNB()

# training the model

classifer1.fit(X\_train, y\_train)

# testing the model

y\_pred1 = classifer1.predict(X\_test)

# importing accuracy score

from sklearn.metrics import accuracy\_score

# printing the accuracy of the model

print(accuracy\_score(y\_test,y\_pred1))

**Output:**

0.91

**Ex. 6 NaiveBayesian Classifier**

**Aim:**By assuming a set of documents that need to be classified, use the naive Bayesian classifier model to perform this task. Built in java classes / API can be used to write the program. Calculate the accuracy, precision and recall for your data set.

**Algorthim:**

1.Import basic libraries.

2.Importing dataset.

3.Data preprocessing.

4.Training the models

5.Testing and evaluation of the model

6.Visualizing the model

**Code:**

# importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

# importing the dataset

dataset = pd.read\_csv("Naive-Bayes-Classification-Data.csv")

# split the data into inputs and outputs

X = dataset.iloc[:, [0,1]].values

y = dataset.iloc[:, 2].values

# training and testing data

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

# assign test data size 25%

X\_train, X\_test, y\_train, y\_test =train\_test\_split(X,y,test\_size= 0.25, random\_state=0)

# importing standard scaler

from sklearn.preprocessing import StandardScaler

# scalling the input data

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.fit\_transform(X\_test)

# importing classifier

from sklearn.naive\_bayes import BernoulliNB

# import Gaussian Naive Bayes classifier

from sklearn.naive\_bayes import GaussianNB

# create a Gaussian Classifier

classifer1 = GaussianNB()

# training the model

classifer1.fit(X\_train, y\_train)

# testing the model

y\_pred1 = classifer1.predict(X\_test)

# importing accuracy score

from sklearn.metrics import accuracy\_score

# printing the accuracy of the model

print(accuracy\_score(y\_test,y\_pred1))

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

print('Accuracy Metrics: \n')

print('Accuracy: ', accuracy\_score(y\_test, y\_pred1))

print('Recall: ', recall\_score(y\_test, y\_pred1))

print('Precision: ', precision\_score(y\_test, y\_pred1))

print('Confusion Matrix: \n', confusion\_matrix(y\_test, y\_pred1))

**Output:**

0.91

Accuracy Metrics:

Accuracy: 0.91

Recall: 0.84375

Precision: 0.8709677419354839

Confusion Matrix:

[[64 4]

[ 5 27]]

**Ex. 7 Bayesian network**

**Aim:**Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

**Algorithm:**

1.First,identify which are the main variable in the problem to solve.

2.Second fine structure of the network that is the causal relationships between all the variable(nodes).

3.Third, define the probabilityrules governing the relationships between the variables.

**Code:** !pip install pgmpy

import numpy as np

import pandas as pd

import csv

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

heartDisease = pd.read\_csv("kk7 (1).csv")

heartDisease = heartDisease.replace('?', np.nan)

print('Sample instances from the dataset are given below')

print(heartDisease.head())

print('\n Attributes and datatypes')

print(heartDisease.dtypes)

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

print('\nLearning CPD using Maximum likelihood estimators')

model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')

HeartDiseasetest\_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')

q1 = HeartDiseasetest\_infer.query(variables=['heartdisease'], evidence={'restecg': 1})

print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')

q2 = HeartDiseasetest\_infer.query(variables=['heartdisease'], evidence={'cp': 2})

print(q2)

**Output:**

Sample instances from the dataset are given below

age gender cp trestbps chol fbs restecg thalach exang oldpeak \

0 63 1 1 145 233 1 2 150 0 2.3

1 67 1 4 160 286 0 2 108 1 1.5

2 67 1 4 120 229 0 2 129 1 2.6

3 37 1 3 130 250 0 0 187 0 3.5

4 41 0 2 130 204 0 2 172 0 1.4

slope ca thal heartdisease

0 3 0 6 0

1 2 3 3 2

2 2 2 7 1

3 3 0 3 0

4 1 0 3 0

Attributes and datatypes

age int64

gender int64

cp int64

trestbps int64

chol int64

fbs int64

restecg int64

thalach int64

exang int64

oldpeak float64

slope int64

ca object

thal object

heartdisease int64

dtype: object

**Ex. 8 EM Algorithm and K-Means Algorithm**

**Aim:** To apply EM algorithm to cluster a set of data stored in a .csv file. Use the same dataset for

clustering using k-means algorithm.

**Algorithm:**

1.Initializing the parameter

2.Calculate the expected value of the variable.

3.Update the parameter by maximizing the likelihood.

4.Repeat step 2 & step 3

5.Output the final parameter

**Code:**

from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

import sklearn.metrics as metrics

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

names = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width', 'Class']

dataset = pd.read\_csv("/content/kk8.csv", names=names)

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

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))

colormap=np.array(['red','lime','black'])

# REAL PLOT

plt.subplot(1,3,1)

plt.title('Real')

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

# K-PLOT

model=KMeans(n\_clusters=3, random\_state=0).fit(X)

plt.subplot(1,3,2)

plt.title('KMeans')

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[model.labels\_])

print('The accuracy score of K-Mean: ',metrics.accuracy\_score(y, model.labels\_))

print('The Confusion matrixof K-Mean:\n',metrics.confusion\_matrix(y, model.labels\_))

# GMM PLOT

gmm=GaussianMixture(n\_components=3, random\_state=0).fit(X)

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

plt.subplot(1,3,3)

plt.title('GMM Classification')

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

print('The accuracy score of EM: ',metrics.accuracy\_score(y, y\_cluster\_gmm))

print('The Confusion matrix of EM:\n ',metrics.confusion\_matrix(y, y\_cluster\_gmm))

**Output:**

The accuracy score of K-Mean: 0.09333333333333334

The Confusion matrixof K-Mean:

[[ 0 50 0]

[ 2 0 48]

[36 0 14]]

The accuracy score of EM: 0.9666666666666667

The Confusion matrix of EM:

[[50 0 0]

[ 0 45 5]

[ 0 0 50]]

**Ex. 9 k-Nearest Neighbour**

**Aim:** 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.

**Algorithm:**

1.Lead the datasets.

2.Normalizing the features.

3.Specify the value of K.

4.Calculate the distance

5.Find k-nearest neighbour

6.Make predictions

7.Output the predictions

8.Repeat 4-7 steps for each query point in the test set if applicable.

**Code:**

import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn import metrics

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe

dataset = pd.read\_csv("/content/kk9.csv", names=names)

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

y = dataset.iloc[:, -1]

print(X.head())

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y, test\_size=0.10)

classifier = KNeighborsClassifier(n\_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0

print ("\n-------------------------------------------------------------------------")

print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))

print ("-------------------------------------------------------------------------")

for label in ytest:

    print ('%-25s %-25s' % (label, ypred[i]), end="")

    if (label == ypred[i]):

        print (' %-25s' % ('Correct'))

    else:

        print (' %-25s' % ('Wrong'))

    i = i + 1

print ("-------------------------------------------------------------------------")

print("\nConfusion Matrix:\n", metrics.confusion\_matrix(ytest, ypred))

print ("-------------------------------------------------------------------------")

print("\nClassification Report:\n", metrics.classification\_report(ytest, ypred))

print ("-------------------------------------------------------------------------")

print('Accuracy of the classifier is %0.2f' % metrics.accuracy\_score(ytest, ypred))

print ("-------------------------------------------------------------------------")

**Output:**

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)

0 5.1 3.5 1.4 0.2

1 4.9 3.0 1.4 0.2

2 4.7 3.2 1.3 0.2

3 4.6 3.1 1.5 0.2

4 5.0 3.6 1.4 0.2

-------------------------------------------------------------------------

Original Label Predicted Label Correct/Wrong

-------------------------------------------------------------------------

0 0 Correct

0 0 Correct

0 0 Correct

1 1 Correct

2 2 Correct

1 1 Correct

0 0 Correct

2 2 Correct

0 0 Correct

2 2 Correct

1 1 Correct

2 2 Correct

0 0 Correct

0 0 Correct

2 2 Correct

-------------------------------------------------------------------------

Confusion Matrix:

[[7 0 0]

[0 3 0]

[0 0 5]]

-------------------------------------------------------------------------

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 7

1 1.00 1.00 1.00 3

2 1.00 1.00 1.00 5

accuracy 1.00 15

macro avg 1.00 1.00 1.00 15

weighted avg 1.00 1.00 1.00 15

-------------------------------------------------------------------------

Accuracy of the classifer is 1.00

\_\_

**Ex. 10 Study of PROLOG**

**Prolog Study**

* Prolog stands for programming in logic. In the logic programming paradigm, prolog language is most widely available. Prolog is a declarative language, which means that a program consists of data based on the facts and rules (Logical relationship) rather than computing how to find a solution. A logical relationship describes the relationships which hold for the given application.
* To obtain the solution, the user asks a question rather than running a program. When a user asks a question, then to determine the answer, the run time system searches through the database of facts and rules.
* Starting Prolog
* Prolog system is straightforward. From one person to other person, the precise details of Prolog will vary. Prolog will produce a number of lines of headings in the starting, which is followed by a line. It contains just
* **?-**
* The above symbol shows the system prompt. The prompt is used to show that the Prolog system is ready to specify one or more goals of sequence to the user. Using a full stop, we can terminate the sequence of goals.
* **?- write('Welcome to Javatpoint'),nl,write('Example of Prolog'),nl.**
* **nl** indicates 'start a new line'. When we press 'return' key, the above line will show the effect like this:
* **Welcome to Javatpoint**
* **Example of Prolog**
* **yes**
* **?- prompt** shows the sequence of goal which is entered by the user. The user will not type the prompt. Prolog system will automatically generate this prompt. It means that it is ready to receive a sequence of goals.
* The above example shows a sequence of goals entered by the user like this:
* **write('Welcome to Javatpoint'), write('Example of Prolog'), nl(twice).**

Consider the following sequence of goals:

**write('Welcome to AI'),nl,write('Example of Prolog'),nl.**

The above sequence of goals has to succeed in order to be succeeded.

* **write('Welcome to AI')**On the screen of the user, Welcome to AI has to be displayed
* **nl**  
  On the screen of the user, a new line has to be output
* **write('Example of Prolog')**
* On the screen of the user, Example of Prolog has to be displayed
* **nl**  
  On the screen of the user, a new line has to be output

All these goals will simply achieve by the Prolog system by outputting the line of text to the screen of the user. To show that the goals have succeeded, we will output **yes**.

The Prolog system predefined the meanings of **nl** and **write**. Write and nl are called as built-in predicates.

**Halt** and **statistics** are the two other built-in predicates. In almost all Prolog versions, these predicates are provided as standard.

* **?-halt.**  
  The above command is used to terminate the Prolog system.
* **?-statistics.**  
  This command will cause the Prolog system statistics. This statistics feature is mainly used to experienced user. In statistics, the following things will generate:

**Ex. 11 8 queens problem**

**Aim:**Write a program to solve 8 queens problem.

**Algorithm:**

**Code:**

% render solutions nicely.

:- use\_rendering(chess).

%% queens(+N, -Queens) is nondet.

%

% @param Queens is a list of column numbers for placing the queens.

% @author Richard A. O'Keefe (The Craft of Prolog)

queens(N, Queens) :-

length(Queens, N),

board(Queens, Board, 0, N, \_, \_),

queens(Board, 0, Queens).

board([], [], N, N, \_, \_).

board([\_|Queens], [Col-Vars|Board], Col0, N, [\_|VR], VC) :-

Col is Col0+1,

functor(Vars, f, N),

constraints(N, Vars, VR, VC),

board(Queens, Board, Col, N, VR, [\_|VC]).

constraints(0, \_, \_, \_) :- !.

constraints(N, Row, [R|Rs], [C|Cs]) :-

arg(N, Row, R-C),

M is N-1,

constraints(M, Row, Rs, Cs).

queens([], \_, []).

queens([C|Cs], Row0, [Col|Solution]) :-

Row is Row0+1,

select(Col-Vars, [C|Cs], Board),

arg(Row, Vars, Row-Row),

queens(Board, Row, Solution).

**Query-**

?- queens(8, Queens).

**Ex. 12 Depth First Search**

**Aim:**Write a program to solve any problem using depth first search.

**Code:**

dynamic true/1, does/2.

role(farmer).

init(left(cabbage)).

init(left(goat)).

init(left(wolf)).

init(left(farmer)).

init(step(1)).

legal(farmer, boat(X)) :-

true(left(farmer)),

true(left(X)),

X \== farmer.

legal(farmer, boat(X)) :-

true(right(farmer)),

true(right(X)),

X \== farmer.

legal(farmer, boat(empty)).

next(left(farmer)) :- true(right(farmer)).

next(right(farmer)) :- true(left(farmer)).

next(step(N)) :- true(step(M)), N is M + 1.

next(left(X)) :-

true(right(X)),

does(farmer, boat(X)).

next(right(X)) :-

true(left(X)),

does(farmer, boat(X)).

next(left(X)) :-

true(left(X)),

does(farmer, boat(Y)),

X \== Y,

X \== farmer.

next(right(X)) :-

true(right(X)),

does(farmer, boat(Y)),

X \== Y,

X \== farmer.

goal(farmer, 100) :-

true(right(cabbage)),

true(right(goat)),

true(right(wolf)),

true(right(farmer)), !.

goal(farmer, 0) :-

true(left(cabbage)),

true(left(goat)),

true(right(farmer)), !.

goal(farmer, 0) :-

true(left(wolf)),

true(left(goat)),

true(right(farmer)), !.

goal(farmer, 0) :-

true(right(cabbage)),

true(right(goat)),

true(left(farmer)), !.

goal(farmer, 0) :-

true(right(wolf)),

true(right(goat)),

true(left(farmer)), !.

goal(farmer, 0) :-

true(step(8)), !.

goal(farmer, 50).

terminal :- goal(farmer, 100).

terminal :- goal(farmer, 0).

%% Heuristic predicate with auxilaries which are puzzle specific

% Must always be higher than zero and monotonic (ie never decreasing)

heuristic(State, [goal(farmer, Value)]) :-

member(step(Step), State),

countrights(State, 0, Rights),

Value is Step + Rights.

countrights([], Value, Value).

countrights([right(Item)|State], Count, Value) :-

Item \== farmer, !,

CountInc is Count + 1,

countrights(State, CountInc, Value).

countrights([\_|State], Count, Value) :-

countrights(State, Count, Value).

**Query -** time(solve\_dfs(Path))

**o/p:**

Path = [

does(farmer,boat(goat)),

does(farmer,boat(empty)),

does(farmer,boat(cabbage)),

does(farmer,boat(goat)),

does(farmer,boat(wolf)),

does(farmer,boat(empty)),

does(farmer,boat(goat))

]

**Ex. 13 8 Puzzle**

**Aim:** Write a program to solve any problem using 8 puzzle.

**Code:**

test(Plan):-

write('Initial state:'),nl,

Init= [at(tile4,1), at(tile3,2), at(tile8,3), at(empty,4), at(tile2,5), at(tile6,6), at(tile5,7), at(tile1,8), at(tile7,9)],

write\_sol(Init),

Goal= [at(tile1,1), at(tile2,2), at(tile3,3), at(tile4,4), at(empty,5), at(tile5,6), at(tile6,7), at(tile7,8), at(tile8,9)],

nl,write('Goal state:'),nl,

write(Goal),nl,nl,

solve(Init,Goal,Plan).

solve(State, Goal, Plan):-

solve(State, Goal, [], Plan).

%Determines whether Current and Destination tiles are a valid move.

is\_movable(X1,Y1) :- (1 is X1 - Y1) ; (-1 is X1 - Y1) ; (3 is X1 - Y1) ; (-3 is X1 - Y1).

solve(State, Goal, Plan, Plan):-

is\_subset(Goal, State), nl,

write\_sol(Plan).

solve(State, Goal, Sofar, Plan):-

act(Action, Preconditions, Delete, Add),

is\_subset(Preconditions, State),

\+ member(Action, Sofar),

delete\_list(Delete, State, Remainder),

append(Add, Remainder, NewState),

solve(NewState, Goal, [Action|Sofar], Plan).

act(move(X,Y,Z),

[at(X,Y), at(empty,Z), is\_movable(Y,Z)],

[at(X,Y), at(empty,Z)],

[at(X,Z), at(empty,Y)]).

is\_subset([H|T], Set):-

member(H, Set),

is\_subset(T, Set).

is\_subset([], \_).

% Remove all elements of 1st list from second to create third.

delete\_list([H|T], Curstate, Newstate):-

remove(H, Curstate, Remainder),

delete\_list(T, Remainder, Newstate).

delete\_list([], Curstate, Curstate).

remove(X, [X|T], T).

remove(X, [H|T], [H|R]):-

remove(X, T, R).

write\_sol([]).

write\_sol([H|T]):-

write\_sol(T),

write(H), nl.

append([H|T], L1, [H|L2]):-

append(T, L1, L2).

append([], L, L).

member(X, [X|\_]).

member(X, [\_|T]):-

member(X, T).

%-----------------------**Output Queries**---------------------------------->

?- test(Plan).

Initial state:

at(tile7,9)

at(tile1,8)

at(tile5,7)

at(tile6,6)

at(tile2,5)

at(empty,4)

at(tile8,3)

at(tile3,2)

at(tile4,1)

Goal state:

[at(tile1,1),at(tile2,2),at(tile3,3),at(tile4,4),at(empty,5),at(tile5,6),at(tile6,7),at(tile7,8),at(tile8,9)]

False**.**

**Ex. 14 traveling salesman**

**Aim:** Write a program to solve any problem using traveling salesman .

**Code:**

edge(a, b, 3).

edge(a, c, 4).

edge(a, d, 2).

edge(a, e, 7).

edge(b, c, 4).

edge(b, d, 6).

edge(b, e, 3).

edge(c, d, 5).

edge(c, e, 8).

edge(d, e, 6).

edge(b, a, 3).

edge(c, a, 4).

edge(d, a, 2).

edge(e, a, 7).

edge(c, b, 4).

edge(d, b, 6).

edge(e, b, 3).

edge(d, c, 5).

edge(e, c, 8).

edge(e, d, 6).

edge(a, h, 2).

edge(h, d, 1).

len([], 0).

len([H|T], N):- len(T, X), N is X+1 .

best\_path(Visited, Total):- path(a, a, Visited, Total).

path(Start, Fin, Visited, Total) :- path(Start, Fin, [Start], Visited, 0, Total).

path(Start, Fin, CurrentLoc, Visited, Costn, Total) :-

edge(Start, StopLoc, Distance), NewCostn is Costn + Distance, \+ member(StopLoc, CurrentLoc),

path(StopLoc, Fin, [StopLoc|CurrentLoc], Visited, NewCostn, Total).

path(Start, Fin, CurrentLoc, Visited, Costn, Total) :-

edge(Start, Fin, Distance), reverse([Fin|CurrentLoc], Visited), len(Visited, Q),

(Q\=7 -> Total is 100000; Total is Costn + Distance).

shortest\_path(Path):-setof(Cost-Path, best\_path(Path,Cost), Holder),pick(Holder,Path).

best(Cost-Holder,Bcost-\_,Cost-Holder):- Cost<Bcost,!.

best(\_,X,X).

pick([Cost-Holder|R],X):- pick(R,Bcost-Bholder),best(Cost-Holder,Bcost-Bholder,X),!.

pick([X],X).

**Query** ?- shortest\_path(Path).

**OUTPUT** Path = 20-[a,h,d,e,b,c,a]

yes