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.

Code:

```
import pandas as pd

import numpy as np


#to read the data in the csv file

data = pd.read_csv("C://users/siva/sport.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))
```

Result:

```
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:

Code:

```
import numpy as np
import pandas as pd
```

```
data = pd.read_csv("E:\Goms_Academic\AI & ML LAB\sport new.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")
```

Result:

```
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' 'wa
rm' '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' 'wa
rm' '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 appro priate data set for building the decision tree and apply this knowledge to classify a new sample.

Code:

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv("3-dataset.csv")
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)
```

Result:

```
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 th e same using appropriate data sets

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 e
rror
    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)
```

Result:

```
-----------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.

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("D://NaiveBayes.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

Result:

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.

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("NaiveBayes.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]]
```

Result:

Ex. 7 Bayesian network

Aim:

Write a program to construct a Bayesian network considering medical data. Use this model to demo nstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Pyth on ML library classes/API.

Code:

```
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('Exp 7.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  oldpe
ak  \
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

Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+

 2. Probability of HeartDisease given evidence= cp
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

Result:

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.

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("8-dataset.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]]
```

Result:

Ex. 9 **k-Nearest Neighbour**

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

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
from sklearn.datasets import load_iris
iris = load_iris()

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

# Read dataset to pandas dataframe
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df['target'] = iris.target
X = df.iloc[:, :-1]
y = df.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 classifer 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
-----------------------------------------------------------------------
--
```

Result:

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.

Code:

```
:- use_module(library(clpfd)).


n_queens(N, Qs) :-
        length(Qs, N),
        Qs ins 1..N,
        safe_queens(Qs).


safe_queens([]).
safe_queens([Q|Qs]) :-
        safe_queens(Qs, Q, 1),
        safe_queens(Qs).


safe_queens([], _, _).
safe_queens([Q|Qs], Q0, D0) :-
        Q0 #\= Q,
        abs(Q0 - Q) #\= D0,
        D1 #= D0 + 1,
        safe_queens(Qs, Q0, D1).
```

Query:

```
queens(8, Qs), labeling([ff], Qs).
```

Ex. 12 **Depth First Search**

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

Code:

```
% solve( Node, Solution):
%    Solution is an acyclic path (in reverse order) between Node and a goal

solve( Node, Solution)  :-
  depthfirst( [], Node, Solution).

% depthfirst( Path, Node, Solution):
%   extending the path [Node | Path] to a goal gives Solution

depthfirst( Path, Node, [Node | Path] )  :-
   goal( Node).

depthfirst( Path, Node, Sol)  :-
  s( Node, Node1),
  \+ member( Node1, Path),                % Prevent a cycle
  depthfirst( [Node | Path], Node1, Sol).

depthfirst2( Node, [Node], _)  :-
   goal( Node).

depthfirst2( Node, [Node | Sol], Maxdepth)  :-
   Maxdepth > 0,
   s( Node, Node1),
   Max1 is Maxdepth - 1,
   depthfirst2( Node1, Sol, Max1).


goal(f).
goal(j).
s(a,b).
s(a,c).
s(b,d).
s(b,e).
s(c,f).
s(c,g).
s(d,h).
s(e,i).
s(e,j).
```

Result:

Ex. 13

8 Puzzle

Aim:

Write a program to solve any problem using 8 puzzle.

Code:

```
ids :-
  start(State),
  length(Moves, N),
  dfs([State], Moves, Path), !,
  show([start|Moves], Path),
  format('~nmoves = ~w~n', [N]).

dfs([State|States], [], Path) :-
  goal(State), !,
  reverse([State|States], Path).

dfs([State|States], [Move|Moves], Path) :-
  move(State, Next, Move),
  not(memberchk(Next, [State|States])),
  dfs([Next,State|States], Moves, Path).

show([], _).
show([Move|Moves], [State|States]) :-
  State = state(A,B,C,D,E,F,G,H,I),
  format('~n~w~n~n', [Move]),
  format('~w ~w ~w~n',[A,B,C]),
  format('~w ~w ~w~n',[D,E,F]),
  format('~w ~w ~w~n',[G,H,I]),
  show(Moves, States).

% Empty position is marked with '*'

start( state(6,1,3,4,*,5,7,2,0) ).

goal( state(*,0,1,2,3,4,5,6,7) ).

move( state(*,B,C,D,E,F,G,H,J), state(B,*,C,D,E,F,G,H,J), right).
move( state(*,B,C,D,E,F,G,H,J), state(D,B,C,*,E,F,G,H,J), down ).
move( state(A,*,C,D,E,F,G,H,J), state(*,A,C,D,E,F,G,H,J), left ).
move( state(A,*,C,D,E,F,G,H,J), state(A,C,*,D,E,F,G,H,J), right).
move( state(A,*,C,D,E,F,G,H,J), state(A,E,C,D,*,F,G,H,J), down ).
move( state(A,B,*,D,E,F,G,H,J), state(A,*,B,D,E,F,G,H,J), left ).
move( state(A,B,*,D,E,F,G,H,J), state(A,B,F,D,E,*,G,H,J), down ).
move( state(A,B,C,*,E,F,G,H,J), state(*,B,C,A,E,F,G,H,J), up   ).
move( state(A,B,C,*,E,F,G,H,J), state(A,B,C,E,*,F,G,H,J), right).
move( state(A,B,C,*,E,F,G,H,J), state(A,B,C,G,E,F,*,H,J), down ).
move( state(A,B,C,D,*,F,G,H,J), state(A,*,C,D,B,F,G,H,J), up   ).
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,D,F,*,G,H,J), right).
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,D,H,F,G,*,J), down ).
```

```
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,*,D,F,G,H,J), left ).
move( state(A,B,C,D,E,*,G,H,J), state(A,B,*,D,E,C,G,H,J), up   ).
move( state(A,B,C,D,E,*,G,H,J), state(A,B,C,D,*,E,G,H,J), left ).
move( state(A,B,C,D,E,*,G,H,J), state(A,B,C,D,E,J,G,H,*), down ).
move( state(A,B,C,D,E,F,*,H,J), state(A,B,C,D,E,F,H,*,J), left ).
move( state(A,B,C,D,E,F,*,H,J), state(A,B,C,*,E,F,D,H,J), up   ).
move( state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,E,F,*,G,J), left ).
move( state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,*,F,G,E,J), up   ).
move( state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,E,F,G,J,*), right).
move( state(A,B,C,D,E,F,G,H,*), state(A,B,C,D,E,*,G,H,F), up   ).
move( state(A,B,C,D,E,F,G,H,*), state(A,B,C,D,E,F,G,*,H), left )
```

Result:

Ex. 15 **traveling salesman**

Aim:

Write a program to solve any problem using traveling salesman .

Code:

Production Rules:-

route(Town1,Town2,Distance) road(Town1,Town2,Distance).

route(Town1,Town2,Distance)

road(Town1,X,Dist1),route(X,Town2,Dist2),Distance=Dist1+Dist2,

```
domains
town = symbol
distance = integer
predicates
nondeterm road(town,town,distance)
nondeterm route(town,town,distance)
clauses
road("tampa","houston",200).
road("gordon","tampa",300).
road("houston","gordon",100).
road("houston","kansas_city",120).
road("gordon","kansas_city",130).
route(Town1,Town2,Distance):-
road(Town1,Town2,Distance).
route(Town1,Town2,Distance):-
road(Town1,X,Dist1),
route(X,Town2,Dist2),
Distance=Dist1+Dist2,!.
```