1. FIND S

```
import pandas as pd
import numpy as np
data = pd.read_csv("2_1.csv")
print("Data:\n", data, "\n")
data_array = np.array(data)
d = data_array[:, :-1]
print("The attributes are:\n", d, "\n")
target = data_array[:, -1]
print("The target is:\n", target, "\n")
def train(c, t):
    specific_hypothesis = None
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break
    if specific_hypothesis is None:
        return "No positive examples found."
    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] = '?'
    return specific_hypothesis
final_hypothesis = train(d, target)
print("The final hypothesis is:\n", final_hypothesis)
```

2. CANDIDATE ELIMINATION

```
import numpy as np
import pandas as pd
data = pd.read_csv("2_1.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")
```

3. ID3 DECISION TREE

```
data = pd.read_csv("ws1.csv")
print("Columns in the dataset:", data.columns)
if 'answer' not in data.columns:
    raise KeyError("The 'answer' column is not found in the dataset. Please check the column names.")
features = [feat for feat in data.columns if feat != "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])
    gain = entropy(examples)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
    return gain
def ID3(examples, attrs):
    root = Node()
    max_gain = -1
    max_feat = None
    for feature in attrs:
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    if max_feat is None:
        return None
    root.value = max_feat
    uniq = np.unique(examples[max_feat])
    for u in uniq:
        subdata = examples[examples[max_feat] == u]
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
```

```
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])[0]
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = [attr for attr in attrs if attr != max_feat]
            child = ID3(subdata, new_attrs)
            if child:  # Only add the child if it's not None
                dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root
def printTree(root: Node, depth=0):
    if root is None:
        return
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    else:
        print()
    for child in root.children:
        printTree(child, depth + 1)
root = ID3(data, features)
printTree(root)
```

4. BACK PROPOGATION

```
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)
y = y/100
def sigmoid (x):
    return 1/(1 + np.exp(-x))
def derivatives_sigmoid(x):
    return x * (1 - x)
epoch=5
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1
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))
for i in range(epoch):
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+bout
    output = sigmoid(outinp)
```

```
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr
    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)
```

5. NAIVE BAYESIAN

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
dataset = pd.read_csv("naive.csv")
X = dataset.iloc[:, [0,1]].values
y = dataset.iloc[:, 2].values
from sklearn.model_selection import train_test_split
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
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import GaussianNB
classifer1 = GaussianNB()
classifer1.fit(X_train, y_train)
y_pred1 = classifer1.predict(X_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,y_pred1))
```

6. NAIVE BAYESAIN - CLASSIFICATION

```
import matplotlib.pyplot as plt

import seaborn as sns

dataset = pd.read_csv("naive.csv")

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

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

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =train_test_split(X,y,test_size= 0.25,
random_state=0)

from sklearn.preprocessing import StandardScaler

sc_X = StandardScaler()

X_train = sc_X.fit_transform(X_train)

X_test = sc_X.fit_transform(X_test)

from sklearn.naive_bayes import BernoulliNB

from sklearn.naive_bayes import GaussianNB

classifer1 = GaussianNB(

classifer1.fit(X_train, y_train)

y_pred1 = classifer1.predict(X_test)

from sklearn.metrics import accuracy_score

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))
```

7. BAYESIAN NETWORK

```
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('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)
```

8. K MEANS AND EM ALGORITHM

```
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.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']
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])
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=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))
```

9. KNN

```
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']
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 ("-------------------------------------------------------------------------")
```

11. 8 QUEENS

```
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).
```

12.DFS

```
% 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).
```

13. BFS

```
import heapq
class Node:
def __init__(self, state, parent, cost, heuristic):
        self.state = state
        self.parent = parent
        self.cost = cost
        self.heuristic = heuristic
def __lt__(self, other):
        return self.heuristic < other.heuristic
def best_first_search(start, goal, heuristic_fn, get_neighbors_fn):
        open_list = []
        closed_list = set()
        start_node = Node(start, None, 0, heuristic_fn(start, goal))
        heapq.heappush(open_list, start_node)
        while open_list:
                current_node = heapq.heappop(open_list)
                if current_node.state == goal:
                        return reconstruct_path(current_node)
                closed_list.add(current_node.state)
        for neighbor, cost in get_neighbors_fn(current_node.state):
                if neighbor in closed_list:
                        continue
                neighbor_node = Node(neighbor, current_node,
                current_node.cost + cost, heuristic_fn(neighbor, goal))
                for open_node in open_list:
                        if open_node.state == neighbor and open_node.cost <=
neighbor_node.cost:
```

```
                    break
                else:
                    heapq.heappush(open_list, neighbor_node)
        return None
def reconstruct_path(node):
        path = []
        while node:
            path.append(node.state)
            node = node.parent
        return path[::-1]
def manhattan_distance(state, goal):
        return abs(state[0] - goal[0]) + abs(state[1] - goal[1])
def get_neighbors(state):
        neighbors = []
        x, y = state
        moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]
        for move in moves:
            neighbor = (x + move[0], y + move[1])
            if 0 <= neighbor[0] < 5 and 0 <= neighbor[1] < 5:
                neighbors.append((neighbor, 1))
            return neighbors
start = (0, 0)
goal = (4, 4)
path = best_first_search(start, goal, manhattan_distance, get_neighbors)
print("Path found:", path)
```

14. 8 PUZZLE

```
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 ).
```

15. TRAVELLING SALESMAN

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,!.