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,!.