Expt. No.: 2 Page. No.: 3

Expt. Name: Candidate - Eliminate algorithm Date:

Aim:- For a given set of training data example stored in a csv file implement and demonstrate the description - Elimination algorithm to output a with the training examples.

Algorithm:-

St-1:- Initialize 'specific - h' to the first positive instance and 'generally - h' to a '?'

St-2:- For each positive instance generals 'spech-h' by setting different attribute to '?'

Step-3:- For each negative instance special 'general-h' by setting mathing attribute.

Step-4:- Remove redudant negative instance special 'general-h'.

Step-5:- Return the final 'special-h' and general-h'

Code:-

```
import numpy as np
import pandas as pd
data = pd.read_csv("E:\Goms_Academics\AI & ml_
LAB\sport new.csv")
concepts = np.array(data.iloc[:,0:-1]
print("\n Instances are:\n", concepts)
target = np.array(data.iloc[:,-1])
print("\n Target Values are:", target)
def learn(concepts, target)
Specific_h = concept[0].copy()
print("\n Specific Boundary:", Specific_h)
```

Expt. No.: 3

Expt. Name: Working of decision tree based ID3 Date:

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:-

St-1:- Compute the entropy of the data set to measure the impurity (or) disorder.

St-2:- For each attribute, compute the information gain, which is the reduction entropy.

St-3:- Choose the attribute with highest information gain of decision node.

St-4:- Divide the dataset into subset based on chosen attribute branch for each unique value of the attribute.

St-5:- Apply same processing to subset.

Code:-

```
import pandas as p
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 = []
```

Expt. No.: 4 Page. No.: 8

Expt. Name: Back Propogation algorithm Date:

Aim:

Build an ANN by implementing the Back propogatio -n algorithm and test the same using appropriate data sets.

Algorithm:

→ Randomly set initial weight & biases
→ Feed your input data into the network.
→ Calculate how main the network output differ from the target values.
→ Adjust weight and biases in each layer to reduce the error.
→ Repeat steps 3 to 4 multiple times.

Code:

```
import numpy as np
X = np.array([[2,9],[1,5],[3,6]], dtype = float)
Y = np.array([[92],[86],[89]), dtype = float,
longitudinally
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
    for i in range(epoch)
hinp1 = np.dot(X, wh)
hinp 1 = hinp1 + bh
```

Expt. No. 5 Date :

Expt. Name Naive Bayesian classifier

Aim:-
Write a program to implement the Naive Baye -sian Classifier for a sample training data set stored csv file. Compute the accuracy of the classifier, considering few test data sets.

Algorithm:-
→ import necessary libraries

→ Load the data set and split data into inputs and outputs.

→ Scale the input data training and testing data

→ Import and initialize classifiers and train the modules.

→ Test the model & evaluate the model

Code:-

```
#importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd.
import seaborn as sns.
dataset = pd.read_csv("D://Naive Bayes.csv")
X = dataset.iloc[:, [0,1]].values.
Y = dataset.iloc[:,2].values.
from sklearn.model_selection import train_test_
                                    split
```

Expt. No.: 6 Page. No.: 12

Expt. Name: Classification of documents using naive Bayesian classifier Date:

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 can be used write the program. Calculate the accuracy, percision and recall for your dataset.

Algorithm:

→ Import necessary libraries, 'numpy', 'pandas', 'csv' and modulus from 'pgmpy'

→ Load the dataset from csv. file named heartdi -sease.

→ Print a sample of the dataset to understand its structure.

→ Create a Bayesian network model with specific relationship between variable.

→ Initialize the influence object and calculate probabilities with evidence.

Code:

```
import numpy as np
import matplotlib.pylot as plt
import pandas as pd
import seaborn as sns
data set = pd.read_csv("Naive Bayes.csv")
```

Expt. No.: 7 Page. No.: 14

Expt. Name: Bayesian Network Date:

Aim:-

Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patient. Using Standard Heart Disease Data Set. You can use Java/python ML Library classes/API.

Algorithm:-

⇒ Initialize libraries and dataset & split it into i/p & o/p

⇒ Scale the data and fit & transform the training data.

⇒ Import and initialize Guassian Naive Bayes classifier. fit the classifier uses the training data and predict the labels for the testing data.

⇒ Calculate & print the accuracy, recall precision and confusion matrix.

Code:-

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import maximumLikelihood
                              Estimator.
from pgmpy.models import BayesianModel.

heart Disease = pd.read_csv('Exp-7.csv')
heart Disease = heart Disease.replace('?', np.nan)
print('sample instances from dataset are given below')
print(heart Disease.head())
print('In Attributes and datatypes')
print(heart Disease.head())
model = BayesianModel([['age', 'heart disease'],
```

Expt. No.: 8 Page. No.: 16

Expt. Name: EM Algorithm and kMean Algorithm Date:

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:

→ Initialize libraries and load dataset assign column names.

→ Visualize the data with plot a scatter plot of the real data

→ Initialize the k-means model with data and plot the clustering result, print the accuracy score and confusion matrix.

→ Guassian matrix model (Gmm) clustering with fit the model the data and predict the label and plot the clustering result.

→ Calculate print the accuracy score and confusion matrix.

Code:

```
from sklearn.cluster import kmeans
from sklearn.mixture import Guassian mixture
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']
X=dataset.iloc[:,:-1]
Y = [label[c] for c in dataset.iloc[:,-1]]
```

Expt. No.: 9 Page. No.: 18

Expt. Name: k-Nearest Neighbour Date:

Aim:-

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. print both correct and wrong predictions. Javalpython ml library classes can be used for their problem.

Algorithm:-

→ import the necessary libraries and dataset.
→ Split the dataset into features and target.
→ Printing the first few rows of features Dataframes
→ Split the 'x' & 'y' data set into training and testing sets.
→ Create an instance and train the KNN classifier
→ Compare & print performance metrics and print a final seperator line

Code:-

```
import numpy as np
import pandas as pd
from sklearn.neighbour import kNeighbourclassifier
from sklearn.model_selection import train_test_split
    from sklearn import metrics
        iris=load_iris()
ames = ['Sepal']-length', 'Sepal-width', 'Petal_length',
'Petal-width', 'class']
df = pd.DataFrame(iris.data, columms= iris.
                                feature_name)

x = df.iloc[:,:-1]
y = df.iloc[:,-1]
print(x.head())
```

Expt. No.: 11 Page. No.: 23

Expt. Name: 8 queens problem Date:

Aim:-

Write a program to solve 8 queens problem.

Algorithm:-

→ Define the Queen problem

→ checks Safety for all queens

→ Check Safety for a Single Queen

→ Construct to ensure Queen do not attact each other

→ If necesary check each Queen Specification again.

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([Qs]).

safe_queens([], _, _).
    Qo #\= Q
```

Expt. No.: 12 Page No.: 25

Expt. Name: Depth First Search Date:

Aim:-
Write a program to solve any problem using

Algorithm:-
→ Solve the path from a node to a goal.
→ Depth-First Search base case - goal reached
→ DFS recursive case extend path.
→ This path will be loop until the goal reached.
→ Define goal states & edges.

Code:-
```
% solve(Node, Solution):
% Solution is an acyclic path (in reverse orde)
between Node and goal.

solve(Node, Solution):-
  depth first([], Node, Solution).

% depth first(Path, Node, Solution):
% extending the path [Node|Path] to a goal gives
                                    Solution.
depth first(Path, Node[Node|path]):-
  goal(Node).

depthfirst(Path, Node, Sol):-
  s(Node, Node1),
  1+member(Node1, path),
```

Expt. No.: 13 Page. No.: 27

Expt. Name: Solve any problem using Best for Search Date:

Aim:-

Write the program to implementation of Best First Search Algorithm

Algorithm:-

→ Initialize the open list (priority queue) with the start node.

→ Initialize the closed list (visited nodes) as empty.

→ While the open list is not empty:

a) Remove the node with the lowest heuristic value from the open list.

b) If the node is the goal, return the path.

c) Otherwise, generate all successor of the current node, D

→ If the successor is not in the open list (or) closed list, add it to the open list and record it's parent.

→ If the successor is in the open list with a higher cost, update it's cost and parent.

→ If the open list is empty and no goal is found, return failure.

Source Code:-

```
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):
```

Expt. No.: 14 Page. No.: 30

Expt. Name: 8 Puzzle Date:

Aim:-

Write a problem using 8 puzzle.

Algorithm:-

→ The start/1 predicate defines the initial state of the puzzle.

→ The goal/1 predicate defines the goal state of the puzzle.

→ The move/3 predicates define the possible moves in the puzzle

→ The dfs/3 predicates performs the depth-first search

→ The show/2 predicate prints the puzzle state & the moves.

Code:-

```
ids:-
 start(State),
 length[moves,N),
 dfs([[state],Moves Path),!,
  show([[start|Moves] path),
 format('~n moves = ~w~n', [N]).
dfs([stat-e[states],[],path]:-
   goal(state),!,
   reverse([state|states],Moves,Path).

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

Expt. No.: 15 Page. No.: 32

Expt. Name: Travelling Salesman Date:

Aim:-

Write a program to solve any problem using traveling salesman.

Algorithm:-

→ The route/3 predicted defines the routing rules. It takes three arguments

→ The road/3 predicate defines the road network the first clause of route/3 says that if there is a direct road between two towns.

→ The second clause of route/3 says that if there is a road from start to intermediate.

→ The domins and predicates declaration specify the data types.

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)
```