

1. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data sample. Read the training data from a file.

Import random.

import csv.

```
attributes = [['sunny', 'rainy'], ['warm', 'cold'], ['windy'],
    ['light'], ['strong', 'weak'], ['warm', 'cool'], ['same',
    'change']]
```

print(attributes)

sun\_attributes = len(attributes)

print(sun\_attributes)

print("b the most general hypothesis: [ '?', '?', '?',
 '?', '?', '?']")

print("b the most specific hypothesis: [ '0', '0', '0',
 '0', '0', '0']")

q = []

print("b the gives training dataset")

with open('c:\Users\Adm\desktop\Aids\data.csv',
 'r') as csvfile:

reader = csv.reader(csvfile)

for row in reader:

q.append(row)

print(row)

print("the initial value of hypothesis: ")

h = ['0'] \* sun\_attributes

print(h)

for j in range(0, num\_attributes):

Date.....

Expt. No. ....

Page No. ....

$$\neg c_{ij} = \neg c_j c_{ij}$$

for  $i$  is range(1, len(a));

if ( $a[i]$ [sum-attributes] == 'yes');

for  $j$  is range(sum-attributes);

if ( $c_{ijj} == '0'$  or  $\neg c_{ij} == a[i][j]$ ):

$$\neg c_{ijj} = \neg c_{ij} c_{ij}$$

else:

$$\neg c_{ij} = '?'$$

Prior ("1s for training examples : 20% the hypothesis") formed ( $i+1$ , b)

Teacher's Signature : .....

Date.....

Expt. No.....

Page No.....

Dataset:

Sl. No	Sky	Airtemp	Humidity	Wind	water	Forecast	Enjoy Sport
1	Sunny	cooms	normal	strong	warm	same	Yes
2	Sunny	cooms	high	strong	warm	same	Yes
3	Rainy	cold	high	strong	warm	change	No
4.	Sunny	cooms	high	strong	cool	change	Yes

output:-

[['Sunny', 'Rainy'], ['warm', 'cold'], ['normal', 'high'], ['strong', 'weak'], ['cooms', 'cool'], ['same', 'change']]

b.

the most general hypothesis. [?, ?, ?, ?, ?, ?, ?]

the most specific hypothesis: [0, 0, 0, 0, 0, 0, 0]

The given training dataset

['Sky', 'Airtemp', 'Humidity', 'Wind', 'water', 'Forecast', 'Enjoy Sport']

['Sunny', 'cooms', 'normal', 'strong', 'warm', 'same', 'Yes']

['Sunny', 'cooms', 'high', 'strong', 'warm', 'same', 'Yes']

['Rainy', 'cold', 'high', 'strong', 'warm', 'change', 'Yes']

The initial hypothesis is :

[0, 0, 0, 0, 0, 0]

Date.....

Expt. No.....

Page No.....

for training example : {0} the hypothesis is  
['sunni', 'wants', '?', 'strange', '?', '?']

2. for a given set of set of training data example stored in a .csv file, implement and demonstrate the candidate-elimination algorithm to output a description of the set of all hypothesis consistent with the training example.

```
import csv
```

```
with open('c://users//Admisi//Desktop//41667//adult.csv')  
as f:
```

```
csv_file = csv.reader(f)
```

```
data = list(csv_file)
```

```
print(data)
```

```
s = data[1][:-1]
```

```
g = [[?] for i in range(len(s))] for j in range(  
len(s))]
```

```
for i in data:
```

```
if [-1] == "yes":
```

```
    for j in range(len(s)):
```

```
        if i[j] == s[j]:
```

```
            g[i][j] = '?'
```

```
        g[i][j] = '?'
```

```
elif [-1] == "No":
```

```
    for j in range(len(s)):
```

```
        if i[j] == s[j]:
```

```
            g[i][j] = s[j].
```

```
else:
```

```
    g[i][j] = "?"
```

```
print("10 steps of candidate elimination algorithm")  
data.index(i+1)
```

Date.....

Expt. No.....2.....

Page No.....4.....

printf(s)

printf(g)

gh = {}

for i is g:

for j is i:

if 'j' == '?' :

gb.append(i)

break

printf(") to find specific hypothesis: (", s)

printf(") to find general hypothesis: (", "igh")

Teacher's Signature : .....

Date.....

Expt. No.....

Page No.....

Dataset:

Sunny.	warm.	normal	strong	warm	same	Yes
Sunny	warm.	high.	strong	warm	same	No
Rainy.	cold.	high.	strong	warm	change	No
Sunny	warm	high.	strong	warm.	change	Yes

Output:

steps of candidate elimination algorithm 1

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

steps of candidate elimination algorithm 2.

[['sunny', 'warm', '?', 'strong', '?', '?'],  
[['sunny', '?', '?', '?', '?', '?'], ['?', 'wind', '?', '?', '?', '?'],  
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],  
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

steps of candidate elimination algorithm 3.

[['sunny', 'warm', '?', 'strong', '?', '?'],  
[['sunny', '?', '?', '?', '?', '?'], ['?', 'wind', '?', '?', '?', '?'],  
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Date.....

Expt. No.....

Page No.....

steps of candidate elimination algorithm &

[ 'sunray', 'wants', '?', 'strong', '?', '?' ]

[ 'sunray', '?', '?', '?', '?', '?' ], [ '?', 'wants', '?', '?', '?', '?' ],

[ '?', '?', '?', '?', '?', '?' ], [ '?', '?', '?', '?', '?', '?' ], [ '?', '?', '?', '?', '?', '?' ]

[ '?', '?', '?', '?', '?' ],

find specific by postnysis

[ 'sunray', 'wants', '?', 'strong', '?', '?' ]

find general hypothesis

[ [ 'sunray', '?', '?', '?', '?', '?' ], [ '?', 'wants', '?', '?', '?', '?' ] ]

3. 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.

```
import pandas as pd.
```

```
from pandas import DataFrame.
```

```
df_tennis = pd.read_csv('c://Users//Admin//Desktop//  
Aibin//playtennis.csv')
```

```
attribute_names = list(df_tennis.columns)
```

```
attribute_names.remove('playTennis').
```

```
print(attribute_names).
```

```
def entropy_of_list(lst):
```

```
from collections import Counter.
```

```
count = Counter(x for x in lst)
```

```
num_instances = len(lst)**1.
```

```
probs = [x/num_instances for x in count.values]
```

```
return entropy(probs).
```

```
def entropy(probs):
```

```
import math
```

```
return sum([-prob * math.log(prob) for  
prob in probs]).
```

```
Total_entropy = entropy_of_list(df_tennis['  
playTennis'])
```

```
def information_gains(df, split_attribute_name,  
target_attribute_name, tree_id=0):
```

```
df_split = df.groupby(split_attribute_name)
```

```
nobs = len(df.index)**1.
```

$df\_agg\_ent = df \text{ split-agg} (\{\text{target\_attribute\_name:}$   
 $\text{[entropy\_of\_list, lambda } x: \text{len}(x)/\text{len(df)}\})$   
 $df\_agg\_ent.\text{columns} = [\text{'entropy'}, \text{'prop\_observations'}]$   
 $\text{new\_entropy} = \text{sum}(\text{df\_agg\_ent}[\text{'entropy'}] * \text{df\_}$   
 $\text{agg\_ent}[\text{'prop\_observations'}])$   
 $\text{old\_entropy} = \text{entropy\_of\_list}(\text{df}[\text{target\_attribute\_name}])$   
 $\text{print(split\_attribute\_name, 'Ig:', old\_entropy\_new\_}$   
 $\text{entropy})$   
 returns  $\text{old\_entropy new\_entropy}$

$\text{def id3(df, target\_attribute\_name, attribute\_names, default\_class=None):}$   
 from collections import Counter

$\text{count} = \text{Counter}(\text{x for } x \in df[\text{target\_attribute\_name}])$   
 if len(count) == 1:  
 return next(iter(count))

if df.empty or (not attribute\_names):  
 return default\_class  
 else:

$\text{default\_class} = \max(\text{count}.keys())$

$\text{gains} = [\text{Information\_gain(df, attr, target\_attribute\_name)} \text{ for } \text{attr} \in \text{attribute\_names}]$   
 $\text{print('')}$

$\text{index\_of\_max} = \text{gains.index}(\max(\text{gains}))$

$\text{best\_attr} = \text{attribute\_names}[index\_of\_max]$

$\text{tree} = \{ \text{best\_attr: } \{ \dots \} \}$

$\text{remaining\_attribute\_names} = [i \text{ for } i \in \text{attribute\_names} \text{ if } i \neq \text{best\_attr}]$

Date.....

Expt. No..... 3.....

Page No..... 7.....

name of  $\{ \} = \text{best-attr} \}$

for attr-val, data-subset is df. group by.  
(best attr):

subtree = DB (data-subset, target-attribute-  
name, remaining attribute-names-default-  
class).

tree [best attr] [attr-val] = subtree

returns:

from print to print print.

tree = DB (df, team's, 'playtennis', attribute-  
names).

print("b") The resultant decision tree is:  
b") print(tree)

Teacher's Signature : .....

Expt. No.....

Output

['outlook', 'temperature', 'humidity', 'wind'].

outlook 1G = 0.2467498197744391

temperature 1G = 0.029222565658554649

humidity 1G : 0.15183550136234136

wind 1G : 0.04818703040826927

temperature 1G : 0.01924309402199489

humidity 1G : 0.0199730400197489

wind 1G : 0.9209505944546686

temperature 2G : 0.570909505944546686

humidity 2G : 0.9709505944546686

wind 2G : 0.01997309402197489

The result of decision tree is

{'outlook': {'overcast': 'Yes',

'Rain': {'wind': {'strong': 'No', 'weak': 'Yes'},  
 'sun': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}

4 Build an artificial neural network by implementing the back propagation algorithm & test the same using appropriate data sets.

from import math as

from random import seed,

from random import random.

def initialize\_network (n\_inputs, n\_hidden,  
n\_outputs):

network = list()

hidden\_layer = [{}'weights': [random() for i in  
range (n\_inputs+1)]} for i in range (n\_hidden)]

network.append (hidden\_layer)

output\_layer = [{}'weights': [random() for i in  
range (n\_hidden+1)]} for i in range (n\_outputs)].

network.append (output\_layer)

return network

def activate (weights, inputs):

activation = weights [-1].

for i in range (len(weights)-1):

activation += weights [i] \* inputs [i].

return activation

def transfer (activation)

return log(1.0 + exp(-activation))

Date.....

Expt. No.....4.....

Page No.....9.....

def forward\_propagate (network, row):

    Input = row

    For layer i in network:

        new\_inputs = []

        for neuron in layer:

            activations = activate (neuron['weights'])

            inputs = neuron['outputs'] = transfer  
            (activations)

            new\_inputs.append (neuron['outputs'])

    Inputs = new\_inputs.

    return inputs.

def transfer\_derivative (output):

    return outputs (1.0 - output).

def backward\_propagate\_error (network,  
    expected):

    for i in reversed (range (len (network))):

        layer = network [i].

        errors = [ ]

        if i != len (network) - 1:

            for j in range (len (layer)):

                error = 0.0.

            For neuron i in network [i+1]:

                error += (neuron ['weights'][i])

                neuron ['delta']).

            errors.append (error).

        else:

            for j in range (len (layer)):

                neuron = layer [j]

Teacher's Signature : .....

errors.append(excepted[j].neuron['output'])  
 for j in range(len(layer)):  
 zeros = layer[i].  
 zeros['delta'] = error[j] \* transfer-  
 derivative(zeros['output'])

def update\_weights(network, row, l-rate):

for i in range(len(network)):

inputs = row[i-1]

if i == 0:

inputs = [zeros['output']] for zeros  
 in network[i-1].

for neuron in network[i]:

for j in range(len(inputs)):

zeros['weights'][i][j] += l-rate \* neuron.  
 ['delta'] \* inputs[i].

zeros['weights'][i][j] += l-rate \* neuron['  
 delta'].

def train\_network(network, train, l-rate, n-  
 epoch, n\_outputs):

for epoch in range(n-epoch):

sum\_error = 0.

for row in train:

outputs = forward\_propagate(network,  
 row).

expected = [0 for i in range(n-  
 outputs)]

expected[-1] = 1

Date.....

Expt. No.....

Page No.....

sum\_error += sum([expected[i] - outputs[i]]\*\*2)

for i in range(len(expected))])

broadcast\_propagate\_error(network, expected)

update\_weights(network, vars\_l\_rate)

print('epoch = ', d, 'l\_rate = ', l\_rate, 'error = ', error)

'\nL epoch, l\_rate, sum error)'

seed(d)

dataset = [[2.4810850, 0.550537003, 0],  
[1.465489372, 0.862125076, 0],  
[3.326561688, 4.400293529, 0],  
[1.38807019, 1.8500000317, 0],  
[3.06407232, 3.00530599310],  
[7.627531014, 0.759262235, 1],  
[5.0330441048, 2.088626795, 1],  
[6.922596716, 1.77106367, 1],  
[8.675418631, -0.242068655, 1],  
[7.673756466, 3.505563011, 1]].

→ inputs = len(dataset) - 1

→ outputs = len(dataset) - 1 for var in dataset]) .

network = initialize\_network(inputs, 2, outputs)

train\_network(network, dataset, 0.8, 20, outputs)

For layer is network:

print(layer)

Teacher's Signature : .....

Expt. No.....

epoch = 0	lrade = 0.500	error = 6.350
epoch = 1	lrade = 0.500	error = 5.531
epoch = 2	lrade = 0.500	error = 5.001
epoch = 3	lrade = 0.500	error = 4.998
epoch = 4	lrade = 0.500	error = 4.899
epoch = 5	lrade = 0.500	error = 4.173
epoch = 6	lrade = 0.500	error = 3.835
epoch = 7	lrade = 0.500	error = 3.806
epoch = 8	lrade = 0.500	error = 3.198
epoch = 9	lrade = 0.500	error = 0.898
epoch = 10	lrade = 0.500	error = 0.626
epoch = 11	lrade = 0.500	error = 0.347
epoch = 12	lrade = 0.500	error = 0.183
epoch = 13	lrade = 0.500	error = 1.953
epoch = 14	lrade = 0.500	error = 1.776
epoch = 15	lrade = 0.500	error = 1.614
epoch = 16	lrade = 0.500	error = 1.478
epoch = 17	lrade = 0.500	error = 1.366
epoch = 18	lrade = 0.500	error = 1.283
epoch = 19	lrade = 0.500	error = 1.152

{'weights': [-1.468875095432307, 1.950887325439314, 1.0858178629550299], 'output': 0.009980305604426185, 'delta': -0.059566604162323605},

{'weights': [0.37711098140462157, -0.0625909814552989, 0.027651237026427167], 'output': 0.94568890021123, 'delta': 0.00262796528508638379},

Date.....

Page No.....

Expt. No.....

[ $\{w\}$  weights]:  $[2.518394649397849, -0.339182750264,$   
 $5.988, -0.9671565426390275],$  'output': -  
 $6.2564879202357587,$  'delta': -  
 $-0.092700592783645873,$   
 $1.00364221062$

[ $\{w\}$  weights]:  $[-2.65891498484263, 0.423830864675827157]$   
 $0.9202, 0.423830864675827157],$  'output':  $0.7790535202438367,$   
'delta':  $0.83803132564373543].$

Copy & type your handwritten notes  
in the following boxes