13/2/23

EXPERIMENT - 3A

AIM

To implement the Candidate Elimination algorithm in Python

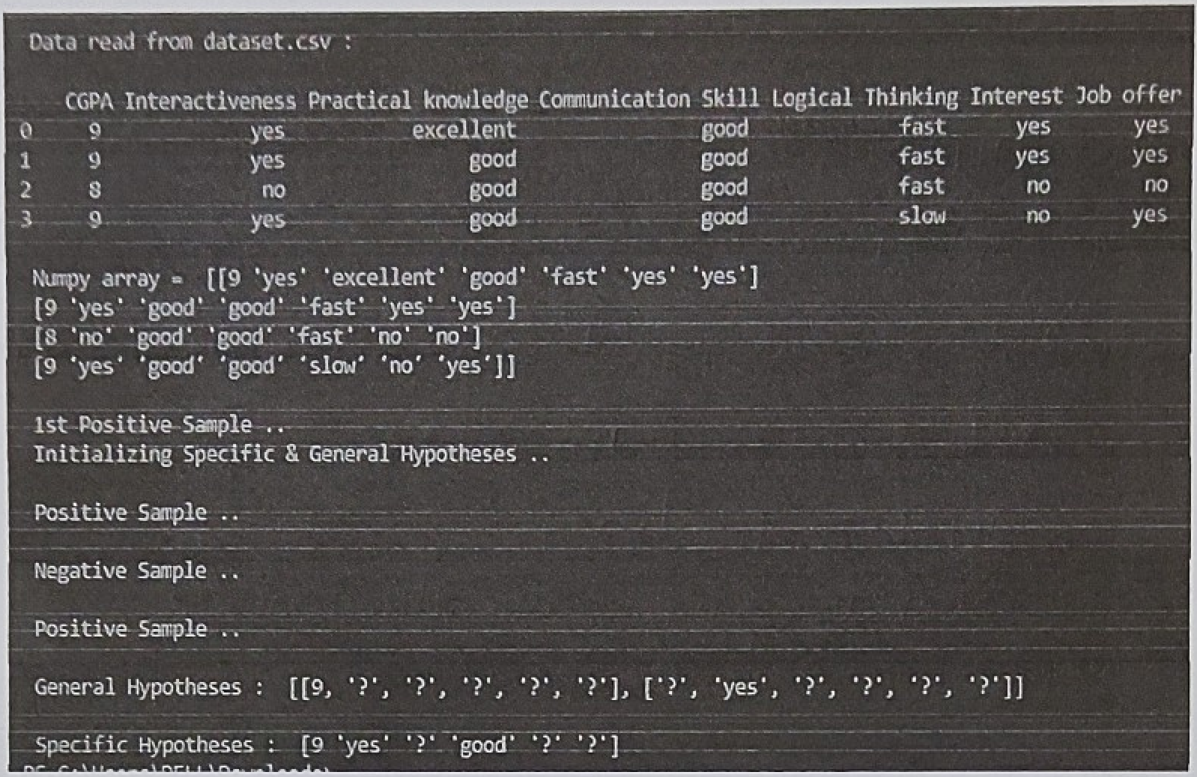
ALGORITHM

```
1  Load dataset
2  Initialize general hypothesis and specific hypothesis
3  For each training escample
       If escample is positive
           If attribute_value == hypothesis_value
               Do nothing
           Else
               Replace attribute value with '?' (basically
               generalizing it)
       Else if escample is negative
           Make generalize hypothesis more specific
```

THEORY

The candidate elimination algorithm incrementally builds the version space by adding the escamples one by one and removing inconsistent hypotheses. This is done by updating the general and specific boundary for each new escample.

OUTPUT



```
Data read from dataset.csv :

   CGPA Interactiveness Practical knowledge Communication Skill Logical Thinking Interest Job offer
0     9             yes           excellent                good             fast      yes       yes
1     9             yes                good                good             fast      yes       yes
2     8              no                good                good             fast       no        no
3     9             yes                good                good             slow       no       yes

Numpy array =  [[9 'yes' 'excellent' 'good' 'fast' 'yes' 'yes']
[9 'yes' 'good' 'good' 'fast' 'yes' 'yes']
[8 'no' 'good' 'good' 'fast' 'no' 'no']
[9 'yes' 'good' 'good' 'slow' 'no' 'yes']]

1st Positive Sample ..
Initializing Specific & General Hypotheses ..

Positive Sample ..

Negative Sample ..

Positive Sample ..

General Hypotheses :  [[9, '?', '?', '?', '?', '?'], ['?', 'yes', '?', '?', '?', '?']]

Specific Hypotheses :  [9 'yes' '?' 'good' '?' '?']
PS C:\Users\DELL\Downloads>
```

CODE

```
import pandas as pd
import numpy as np

data = pd.read_csv ( "ML Lab Files/ce_dataset.csv" )
print ( "\n Data read from dataset.csv :\n\n" , data )

arr = np.array ( data )
print ( "\n Numpy array = " , arr )
X = arr [ : , : -1 ]
y = arr [ : , -1 ]

el = 0
while el < len ( y ) :
    if y [ el ] == "yes" :
        print ( "\n 1st Positive Sample ..\n Initializing Specific & General Hypotheses .." )
        break
    el += 1

specific = X [ el ]
general = [ [ '?' for x in range ( len ( X [ 0 ] ) ) ] for y in range ( len ( X [ 0 ] ) ) ]

def getHypotheses ( specific , general , ind ) :
    while ind < len ( y ) :
        if y [ ind ] == "yes" :
            print ( "\n Positive Sample .." )
            for hyp in range ( len ( specific ) ) :
                if specific [ hyp ] != X [ ind ] [ hyp ] :
                    specific [ hyp ] = '?'
                    general [ hyp ] [ hyp ] = '?'
        else :
            print ( "\n Negative Sample .." )
            for hyp in range ( len ( specific ) ) :
                general [ hyp ] [ hyp ] = specific [ hyp ] if specific [ hyp ] != X [ ind ] [ hyp ] else '?'
        ind += 1
    if len ( general ) > 1 :
        lst = [ '?' for x in range ( len ( specific ) ) ]
        while lst in general :
            general.remove ( lst )
    return specific , general

specific_hypothesis , general_hypothesis = getHypotheses ( specific , general , el + 1 )
print ( "\n General Hypotheses : " , general_hypothesis )
print ( "\n Specific Hypotheses : " , specific_hypothesis )
```

RESULT

Hence, Candidate Elimination algorithm has been implemented successfully

13/2/23

EXPERIMENT - 3B

AIM

To implement the list-then-eliminate algorithm in Python

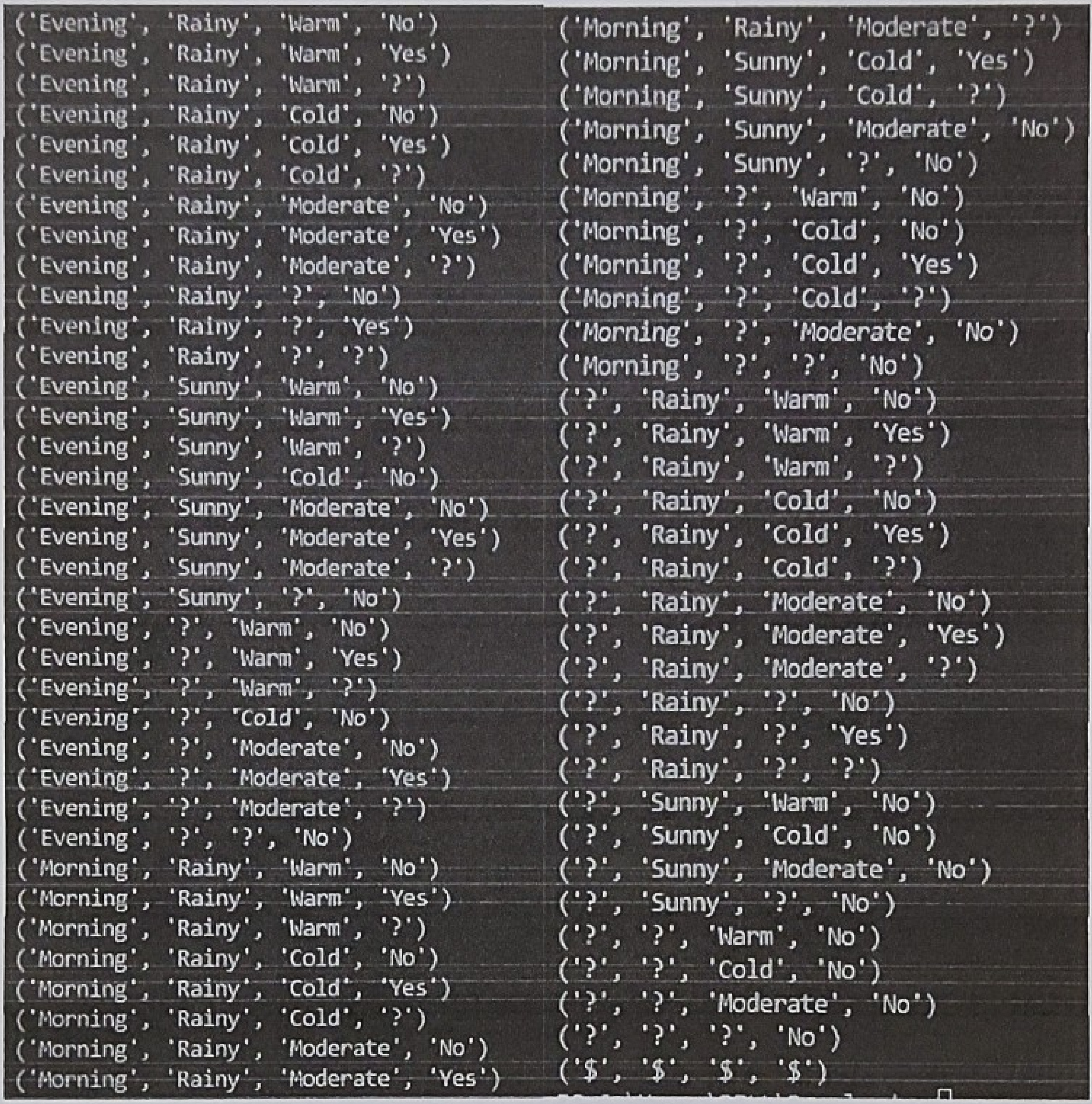
ALGORITHM

```
* Version Space ← list containing every hypothesis in H
* For each training example $(x, c(x))$
    Remove from version space h, for which
    $h(x) \neq c(x)$
* Output list of hypothesis in version space
```

THEORY

The List-Then-Eliminate algorithm is a machine learning algorithm that begins with a full version space (containing every hypothesis in H). Then, for every training example, we remove every hypothesis from the version space that is inconsistent i.e., that does not agree with the training examples.

OUTPUT



```
('Evening', 'Rainy', 'Warm', 'No')
('Evening', 'Rainy', 'Warm', 'Yes')
('Evening', 'Rainy', 'Warm', '?')
('Evening', 'Rainy', 'Cold', 'No')
('Evening', 'Rainy', 'Cold', 'Yes')
('Evening', 'Rainy', 'Cold', '?')
('Evening', 'Rainy', 'Moderate', 'No')
('Evening', 'Rainy', 'Moderate', 'Yes')
('Evening', 'Rainy', 'Moderate', '?')
('Evening', 'Rainy', '?', 'No')
('Evening', 'Rainy', '?', 'Yes')
('Evening', 'Rainy', '?', '?')
('Evening', 'Sunny', 'Warm', 'No')
('Evening', 'Sunny', 'Warm', 'Yes')
('Evening', 'Sunny', 'Warm', '?')
('Evening', 'Sunny', 'Cold', 'No')
('Evening', 'Sunny', 'Moderate', 'No')
('Evening', 'Sunny', 'Moderate', 'Yes')
('Evening', 'Sunny', 'Moderate', '?')
('Evening', 'Sunny', '?', 'No')
('Evening', '?', 'Warm', 'No')
('Evening', '?', 'Warm', 'Yes')
('Evening', '?', 'Warm', '?')
('Evening', '?', 'Cold', 'No')
('Evening', '?', 'Moderate', 'No')
('Evening', '?', 'Moderate', 'Yes')
('Evening', '?', 'Moderate', '?')
('Evening', '?', '?', 'No')
('Morning', 'Rainy', 'Warm', 'No')
('Morning', 'Rainy', 'Warm', 'Yes')
('Morning', 'Rainy', 'Warm', '?')
('Morning', 'Rainy', 'Cold', 'No')
('Morning', 'Rainy', 'Cold', 'Yes')
('Morning', 'Rainy', 'Cold', '?')
('Morning', 'Rainy', 'Moderate', 'No')
('Morning', 'Rainy', 'Moderate', 'Yes')
```

```
('Morning', 'Rainy', 'Moderate', '?')
('Morning', 'Sunny', 'Cold', 'Yes')
('Morning', 'Sunny', 'Cold', '?')
('Morning', 'Sunny', 'Moderate', 'No')
('Morning', 'Sunny', '?', 'No')
('Morning', '?', 'Warm', 'No')
('Morning', '?', 'Cold', 'No')
('Morning', '?', 'Cold', 'Yes')
('Morning', '?', 'Cold', '?')
('Morning', '?', 'Moderate', 'No')
('Morning', '?', '?', 'No')
('?', 'Rainy', 'Warm', 'No')
('?', 'Rainy', 'Warm', 'Yes')
('?', 'Rainy', 'Warm', '?')
('?', 'Rainy', 'Cold', 'No')
('?', 'Rainy', 'Cold', 'Yes')
('?', 'Rainy', 'Cold', '?')
('?', 'Rainy', 'Moderate', 'No')
('?', 'Rainy', 'Moderate', 'Yes')
('?', 'Rainy', 'Moderate', '?')
('?', 'Rainy', '?', 'No')
('?', 'Rainy', '?', 'Yes')
('?', 'Rainy', '?', '?')
('?', 'Sunny', 'Warm', 'No')
('?', 'Sunny', 'Cold', 'No')
('?', 'Sunny', 'Moderate', 'No')
('?', 'Sunny', '?', 'No')
('?', '?', 'Warm', 'No')
('?', '?', 'Cold', 'No')
('?', '?', 'Moderate', 'No')
('?', '?', '?', 'No')
('$', '$', '$', '$')
```

CODE

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

data = pd.read_csv ( "ML Lab Files/dataset.csv" )
d = np.array ( data.iloc [ : , : -1 ] )
t= np.array ( data.iloc [ : , -1 ] )
l , c = [] , []
for i in range ( 0 , 4 ) :
    l.append ( list ( set ( data.iloc [ : , i ] ) ) )
for i in l :
    i.append ( '?' )
total_hypo = list ( itertools.product ( *l ) )
total_hypo.append ( tuple ( [ '$' for i in range ( 4 ) ] ) )
yes = list ( t ).count ( "yes" )
for tup in total_hypo :
    of = 0
    for j , x in enumerate ( d ) :
        flag = 0
        for i in range ( 4 ) :
            if ( tup [ i ] == x [ i ] ) or tup [ i ] == '?' :
                flag += 1
        if flag == 4 :
            if t [ j ] == "Yes" :
                of += 1
    if of == yes :
        c.append ( tup )
for j in c :
    print ( j )
```

RESULT

Hence, list-then-eliminate algorithm has been implemented successfully

13/2/23

EXPERIMENT -3C

AIM

To implement the ID3 decision tree algorithm in Python

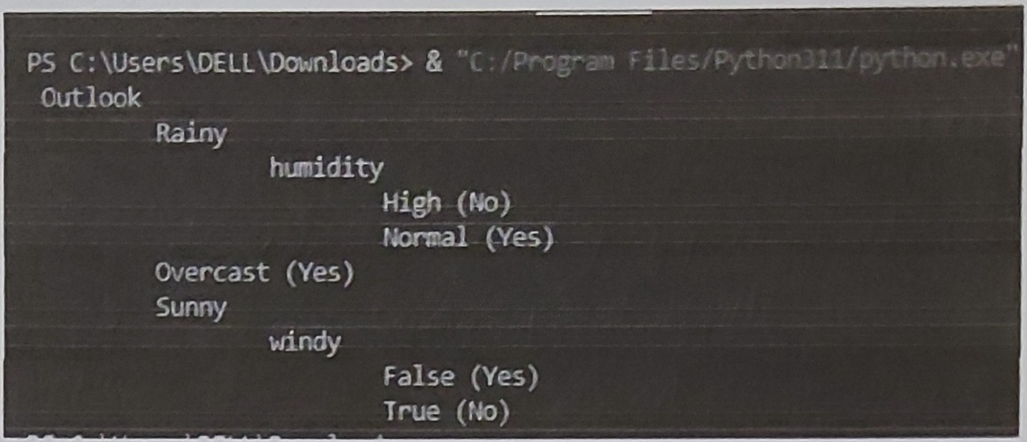
ALGORITHM

- Calculate entropy for the dataset
- For each attribute/feature
  - i) Calculate entropy for all its categorical values
  - ii) Calculate information gain for the feature
- Find the feature with maximum information gain
- Repeat it until we get the desired tree

THEORY

ID3 stands for Iterative Dichotomiser 3 and is named such because it iteratively divides features into 2 or more groups at each step. ID3 uses a top-down greedy approach to build a decision tree. It is mostly used for classification problems with nominal features only.

OUTPUT



```
PS C:\Users\DELL\Downloads> & "C:/Program Files/Python311/python.exe"
 Outlook
        Rainy
                humidity
                        High (No)
                        Normal (Yes)
        Overcast (Yes)
        Sunny
                windy
                        False (Yes)
                        True (No)
```

CODE

```
import pandas as pd
import numpy as np

def calc_total_entropy ( data , target_name , target_val ) :
    size = data.shape [ 0 ]
    total_entropy = 0
    for i in target_val :
        count = data [ data [ target_name ] == i ].shape [ 0 ]
        entropy = - ( count / size ) * np.log2 ( count / size )
        total_entropy += entropy
    return total_entropy

def calc_entropy ( data , target_name , target_val ) :
    size = data.shape [ 0 ]
    total_entropy = 0
    for i in target_val :
        count = data [ data [ target_name ] == i ].shape [ 0 ]
        entropy = 0
        if count != 0 :
            entropy = - ( count / size ) * np.log2 ( count / size )
        total_entropy += entropy
    return total_entropy

def calc_gain ( attribute , data , target_name , target_val ) :
    list_ = data [ attribute ].unique ()
    size = data.shape [ 0 ]
    gain = 0.0
    for i in list_ :
        t_data = data [ data [ attribute ] == i ]
        count = data [ data [ attribute ] == i ].shape [ 0 ]
        entropy = calc_entropy ( t_data , target_name , target_val )
        prob = count / size
        gain += prob * entropy
    return calc_total_entropy ( data , target_name , target_val ) - gain

def info_attribute ( data , target_name , target_val ) :
    list_ = data.columns.drop ( target_name )
    max_gain = -1
    info_feat = None
    for i in list_ :
        gain = calc_gain ( i , data , target_name , target_val )
        if max_gain < gain :
            max_gain = gain
            info_feat = i
    return info_feat

def generate_tree ( attribute , data , target_name , target_val ) :
    count_dict = data [ attribute ].value_counts ( sort = False )
    tree = {}
    for value , count in count_dict.items () :
        feat_data = data [ data [ attribute ] == value ]
        pure = False
        for t in target_val :
            class_count = feat_data [ feat_data [ target_name ] == t ].shape [ 0 ]
            if class_count == count :
                tree [ value ] = t
                data = data [ data [ attribute ] != value ]
                pure = True
        if not pure :
            tree [ value ] = '?'
    return tree , data

def make_tree ( root , prev , data , target_name , target_val ) :
    if data.shape [ 0 ] != 0 :
        max_info_attribute = info_attribute ( data , target_name , target_val )
        tree , data = generate_tree ( max_info_attribute , data , target_name , target_val )
        next_root = None
        if prev != None :
            root [ prev ] = dict ()
            root [ prev ] [ max_info_attribute ] = tree
            next_root = root [ prev ] [ max_info_attribute ]
        else :
            root [ max_info_attribute ] = tree
            next_root = root [ max_info_attribute ]
        for node , branch in list ( next_root.items () ) :
            if branch == "?" :
                feat_data = data [ data [ max_info_attribute ] == node ]
                make_tree ( next_root , node , feat_data , target_name , target_val )

def id3 ( data , target_name ) :
    tree = {}
    target = data [ target_name ].unique ()
    make_tree ( tree , None , data , target_name , target )
    return tree

def printTree ( tree , d = 0 ) :
    if ( tree == None or len ( tree ) == 0 ) :
        print ( "\t" * d , "-" )
    else :
        for key , val in tree.items () :
            if ( isinstance ( val , dict ) ) :
                print ( "\t" * d , key )
                printTree ( val , d + 1 )
            else :
                print ( "\t" * d , key , str ( '(' ) + val + str ( ')' ) )

data = pd.read_csv ( "ML Lab Files/climate.csv" )
tree = id3 ( data , "play golf" )
printTree ( tree )
```

RESULT

Hence, ID3 decision tree algorithm has been implemented successfully