FOML – Lab Manual

Experiment 1
Design a Simple 3 class dataset with 30 records and the features are represented in 4 columns. Apply rule based classification method to predict the classes in python.

Program

```
import pandas as pd
df = pd.read_csv("Ex1.csv")
#print(df.to_string())
display(df.iloc[1])
for i in range (30):
  if df["Color"].iloc[i] == "White" or "Black" or "BW" :
    if df["Sound"].iloc[i] > 79 and df["Sound"].iloc[i] < 101:
      if df["Tail Length"].iloc[i] > 0.6:
        if df["Height"].iloc[i] > 2.9:
          print("Horse")
  if df["Color"].iloc[i] == "White" or "Black" or "BW" or "Brown" or "BBrW":
    if df["Sound"].iloc[i] < 71:
      if df["Tail Length"].iloc[i] < 0.4:
        if df["Height"].iloc[i] < 1:
          print("Cat")
  if df["Color"].iloc[i] == "White" or "Black" or "BW" or "Brown" or "BBrW":
    if df["Sound"].iloc[i] > 100:
      if df["Tail Length"].iloc[i] > 0.4 and df["Tail Length"].iloc[i] < 0.7:
        if df["Height"].iloc[i] > 1 and df["Height"].iloc[i] < 2:
          print("Dog")
```

OUTPUT

Horse
Horse
Horse
Horse
Horse
Horse
Horse
Horse
Horse
Horse
Cat
Cat
Cat
Cat

Cat
Cat
Cat
Cat
Cat
Cat
Cat
Dog
Dog
Dog
Dog
Dog
Dog

Experiment 2

Extract the input and output data from the CSV file using python and split the training and testing data in the ratio of 70:30

Program

```
import pandas as pd
df = pd.read_csv("Ex1.csv")
from sklearn.model_selection import train_test_split
print(df.head())
Features=["Color","Sound","Tail Length","Height"]
X=df.loc[:, Features]
Y=df.loc[:,['Class']]
X_train, X_test, y_train, y_test = train_test_split(X,Y,
                                    random_state=104,
                                    test_size=0.25,
                                    shuffle=True)

print(X_train.head())
print(X_train.shape)
print(X_test.head())
print(X_test.shape)
print(y_train.head())
print(y_train.shape)
print(y_test.head())
print(y_test.shape)
```

Output

```
Color  Sound  Tail Length  Height  Class
0  Black    100         1.0     3.0  Horse
1  White     80         0.9     3.0  Horse
2     BW     90         0.7     3.0  Horse
```

```
3  Black     95          1.0     3.0  Horse
4  White     91          1.0     3.0  Horse
     Color  Sound  Tail Length  Height
17   Black     55          0.2     0.5
18   White     63          0.3     0.4
9    Black    100          1.0     3.0
19  Brown      73          0.2     0.2
32     BrW    115          0.6     1.5
(25, 4)
     Color  Sound  Tail Length  Height
11   Black     50          0.2     0.6
30   White    134          0.6     1.4
10      BW     45          0.3     0.9
31  Brown     128          0.5     1.3
22      BW    120          0.5     1.2
(9, 4)
    Class
17    Cat
18    Cat
9   Horse
19    Cat
32    Dog
(25, 1)
   Class
11   Cat
30   Dog
10   Cat
31   Dog
22   Dog
(9, 1)
```

Experiment 3

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

Program

```
import pandas as pd
import numpy as np
d = pd.read_csv("Tennis.csv")

a = np.array(d)[:,:-1]
print(" The attributes are: ",a)
t = np.array(d)[:,-1]
print("The target is: ",t)
```

```
print(t)

def train(c,t):
 for i, val in enumerate(t):
  if val == "Yes":
   specific_hypothesis = c[i].copy()
   pass
 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] = '?'
    else:
     pass
   return specific_hypothesis
print(" The final hypothesis is:",train(a,t))
```

Output

```
The attributes are:  [['Sunny' 'Hot' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Strong']
 ['Overcast' 'Hot' 'High' 'Weak']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Cool' 'Normal' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
The target is:  ['No' 'No' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes' 'Yes'
'Yes' 'Yes' 'Yes'
 'No']
['No' 'No' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes' 'Yes' 'Yes' 'Yes' 'Yes'
 'No']
 The final hypothesis is: ['Overcast' 'Hot' '?' 'Weak']
```

Experiment 4

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Program

```
import csv
# open the CSVFile and keep all rows as list of tuples
with open('EnjoySport.csv') as csvFile:
    examples = [tuple(line) for line in csv.reader(csvFile)]
print(examples)
# To obtain the domain of attribute values defined in the instances X
def get_domains(examples):
    # set function returns the unordered collection of items with no
duplicates
    d = [set() for i in examples[0]]
    for x in examples:
        #Enumerate() function adds a counter to an iterable and returns it
in a form of enumerate object i.e(index,value)
        for i, xi in enumerate(x):
            d[i].add(xi)
    return [list(sorted(x)) for x in d]
# Test the get_domains function
get_domains(examples)

# Repeat the '?' and '0' length of domain no of times
def g_0(n):
    return ('?',)*n

def s_0(n):
    return ('0',)*n

# Function to check generality between two hypothesis
def more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        mg = x == '?' or (x != '0' and (x == y or y == '0'))
        more_general_parts.append(mg)
    return all(more_general_parts) # Returns true if all elements of list
or tuple are true

# Function to check whether train examples are consistent with hypothesis
def consistent(hypothesis,example):
    return more_general(hypothesis, example)
```

```
# Function to add min_generalizations
def min_generalizations(h, x):
    h_new = list(h)
    for i in range(len(h)):
        if not consistent(h[i:i+1],x[i:i+1]):
            if h[i] != '0':
                h_new[i] = '?'
            else:
                h_new[i] = x[i]
    return [tuple(h_new)]

# Function to generalize Specific hypto
def generalize_S(x, G, S):
    S_prev = list(S)
    for s in S_prev:
        if s not in S:
            continue
        if not consistent(s,x):
            S.remove(s)
            Splus = min_generalizations(s, x)
            # Keep only generalizations that have a counterpart in G
            S.update([h for h in Splus if any([more_general(g,h)
                                               for g in G])])
            # Remove from S any hypothesis more general than any other
hypothesis in S
            S.difference_update([h for h in S if
                                 any([more_general(h, h1)
                                      for h1 in S if h != h1])])
    return S

# Function to add min_specializations
def min_specializations(h, domains, x):
    results = []
    for i in range(len(h)):
        if h[i] == '?':
            for val in domains[i]:
                if x[i] != val:
                    h_new = h[:i] + (val,) + h[i+1:]
                    results.append(h_new)
        elif h[i] != '0':
            h_new = h[:i] + ('0',) + h[i+1:]
            results.append(h_new)
    return results
```

```
# Function to specialize General hypotheses boundary
def specialize_G(x, domains, G, S):
    G_prev = list(G)
    for g in G_prev:
        if g not in G:
            continue
        if consistent(g,x):
            G.remove(g)
            Gminus = min_specializations(g, domains, x)
            # Keep only specializations that have a counterpart in S
            G.update([h for h in Gminus if any([more_general(h, s)
                                                    for s in S])])
            # Remove hypothesis less general than any other hypothesis in
G
            G.difference_update([h for h in G if
                                 any([more_general(g1, h)
                                      for g1 in G if h != g1])])
    return G

# Function to perform CandidateElimination
def candidate_elimination(examples):
    domains = get_domains(examples)[:-1]

    G = set([g_0(len(domains))])
    S = set([s_0(len(domains))])
    i=0
    print('All the hypotheses in General and Specific boundary are:\n')
    print('\n G[{0}]:'.format(i),G)
    print('\n S[{0}]:'.format(i),S)
    for xcx in examples:
        i=i+1
        x, cx = xcx[:-1], xcx[-1]  # Splitting data into attributes and
decisions
        if cx=='Yes': # x is positive example
            G = {g for g in G if consistent(g,x)}
            S = generalize_S(x, G, S)
        else: # x is negative example
            S = {s for s in S if not consistent(s,x)}
            G = specialize_G(x, domains, G, S)
        print('\n G[{0}]:'.format(i),G)
        print('\n S[{0}]:'.format(i),S)
    return

candidate_elimination(examples)
```

Output

```
[('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'), ('Sunny',
'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'), ('Rainy', 'Cold',
'High', 'Strong', 'Warm', 'Change', 'No'), ('Sunny', 'Warm', 'High',
'Strong', 'Cool', 'Change', 'Yes')]
All the hypotheses in General and Specific boundary are:


 G[0]: {('?', '?', '?', '?', '?', '?')}

 S[0]: {('0', '0', '0', '0', '0', '0')}

 G[1]: {('?', '?', '?', '?', '?', '?')}

 S[1]: {('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same')}

 G[2]: {('?', '?', '?', '?', '?', '?')}

 S[2]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

 G[3]: {('?', '?', '?', '?', '?', 'Same'), ('Sunny', '?', '?', '?', '?',
'?'), ('?', 'Warm', '?', '?', '?', '?')}

 S[3]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

 G[4]: {('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?',
'?')}

 S[4]: {('Sunny', 'Warm', '?', 'Strong', '?', '?')}
```