LAB 1

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

---

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
d=pd.read_csv("data.csv")
print(d)
```

```
     sky air temp humidity    wind water forecast enjoy sport
0  sunny     warm   normal  strong  warm     same         yes
1  sunny     warm     high  strong  warm     same         yes
2  rainy     cold     high  strong  warm   change          no
3  sunny     warm     high  strong  cool   change         yes
```

In [3]:
```
att=np.array(d)[:,:-1]
```

In [4]:
```
print(att)
```

```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
```

In [5]:
```
tar=np.array(d)[:,-1]
```

In [6]:
```
print(tar)
```

```
['yes' 'yes' 'no' 'yes']
```

In [9]:
```
def finds(att, tar):
    for i, val in enumerate(tar):
        if val == "yes":
            res=att[i].copy()
            break
    for i, val in enumerate(att):
        if tar[i] == "yes":
            for x in range (len(res)):
                if val[x] != res[x]:
                    res[x] = "?"
                else:
                    pass
    return res
```

In [10]:
```
print(finds(att,tar))
```

```
['sunny' 'warm' '?' 'strong' '?' '?']
```

In [ ]:

LAB 2

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.

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
data= pd.read_csv("data.csv")
```

In [3]:
```
data
```

Out[3]:

	sky	air temp	humidity	wind	water	forecast	enjoy sport
0	sunny	warm	normal	strong	warm	same	yes
1	sunny	warm	high	strong	warm	same	yes
2	rainy	cold	high	strong	warm	change	no
3	sunny	warm	high	strong	cool	change	yes

In [4]:
```
concepts=np.array(data.iloc[:,0:-1])
```

In [5]:
```
concepts
```

Out[5]:
```
array([['sunny', 'warm', 'normal', 'strong', 'warm', 'same'],
       ['sunny', 'warm', 'high', 'strong', 'warm', 'same'],
       ['rainy', 'cold', 'high', 'strong', 'warm', 'change'],
       ['sunny', 'warm', 'high', 'strong', 'cool', 'change']],
      dtype=object)
```

In [6]:
```
target=np.array(data.iloc[:,-1])
```

In [7]:
```
target
```

Out[7]:
```
array(['yes', 'yes', 'no', 'yes'], dtype=object)
```

In [35]:
```
def learn(concepts, target):
    specific_h=concepts[0].copy()
    general_h=[["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    for i, h in enumerate(concepts):
        if target[i]=="yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x]=specific_h[x]

        if target[i]=="no":
            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]='?'

    indices=[i for i,val in enumerate(general_h) if val==['?','?','?','?','?','?']]
    for i in indices:
        general_h.remove(['?','?','?','?','?','?'])
    return specific_h, general_h
```

In [36]:
```
s_final, f_final = learn(concepts,target)
```

In [37]:
```
s_final
```

Out[37]:
```
array(['sunny', 'warm', '?', 'strong', '?', '?'], dtype=object)
```

In [38]:
```
f_final
```

Out[38]:
```
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

In [ ]:
```

```

LAB 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.

In [17]:
```
import pandas as pd
import math
import numpy as np
```

In [18]:
```
data = pd.read_csv("3-dataset.csv")
```

In [19]:
```
data
```

Out[19]:

	outlook	temperature	humidity	wind	answer
0	sunny	hot	high	weak	no
1	sunny	hot	high	strong	no
2	overcast	hot	high	weak	yes
3	rain	mild	high	weak	yes
4	rain	cool	normal	weak	yes
5	rain	cool	normal	strong	no
6	overcast	cool	normal	strong	yes
7	sunny	mild	high	weak	no
8	sunny	cool	normal	weak	yes
9	rain	mild	normal	weak	yes
10	sunny	mild	normal	strong	yes
11	overcast	mild	high	strong	yes
12	overcast	hot	normal	weak	yes
13	rain	mild	high	strong	no

In [20]:
```
features=[feat for feat in data]
features.remove("answer")
```

In [21]:
```
features
```

Out[21]:
```
['outlook', 'temperature', 'humidity', 'wind']
```

In [22]:
```
class Node:
    def __init__(self):
        self.children=[]
        self.value=""
        self.isLeaf=False
        self.pred=""
```

In [28]:
```
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))
```

In [29]:
```
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
```

In [39]:
```
def ID3(examples, attrs):
    root = Node()
    max_gain = 0
    max_feat = ""
    for feature in attrs:
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    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"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root
```

In [40]:
```
def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end=" ")
    print(root.value, end=" ")
    if root.isLeaf:
        print("->", root.pred)
    print()
    for child in root.children:
        printTree(child, depth+1)
```

In [41]:
```
root=ID3(data, features)
printTree(root)
```

```
outlook
        overcast -> ['yes']

        rain
                wind
                        strong -> ['no']

                        weak -> ['yes']

        sunny
                humidity
                        high -> ['no']

                        normal -> ['yes']
```

LAB 4

Implement the Linear Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
```

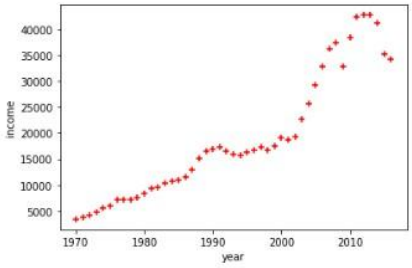
In [ ]:
```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/canada_per_capita_income - canada_per_capita_income.csv')
df
```

Out[ ]:

	year	income
0	1970	3399.299037
1	1971	3768.297935
2	1972	4251.175484
3	1973	4804.463248
4	1974	5576.514583
5	1975	5998.144346
6	1976	7062.131392
7	1977	7100.126170
8	1978	7247.967035
9	1979	7602.912681
10	1980	8355.968120
11	1981	9434.390652
12	1982	9619.438377
13	1983	10416.536590
14	1984	10790.328720
15	1985	11018.955850
16	1986	11482.891530
17	1987	12974.806620
18	1988	15080.283450
19	1989	16426.725480
20	1990	16838.673200
21	1991	17266.097690
22	1992	16412.083090
23	1993	15875.586730
24	1994	15755.820270
25	1995	16369.317250
26	1996	16699.826680
27	1997	17310.757750
28	1998	16622.671870
29	1999	17581.024140

In [ ]:
```
%matplotlib inline
plt.xlabel('year')
plt.ylabel('income')
plt.scatter(df.year,df.income,color='red',marker='+')
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7f1d407f0f10>
```



In [ ]:
```
new_df = df.drop('income',axis='columns')
new_df
```

Out[ ]:

	year
0	1970
1	1971
2	1972
3	1973
4	1974
5	1975
6	1976
7	1977
8	1978
9	1979
10	1980
11	1981
12	1982
13	1983
14	1984
15	1985
16	1986
17	1987
18	1988
19	1989
20	1990

In [ ]:
```
income = df.income
income
```

Out[ ]:
```
0      3399.299037
1      3768.297935
2      4251.175484
3      4804.463248
4      5576.514583
5      5998.144346
6      7062.131392
7      7100.126170
8      7247.967035
9      7602.912681
10     8355.968120
11     9434.390652
12     9619.438377
13    10416.536590
14    10790.328720
15    11018.955850
16    11482.891530
17    12974.806620
18    15080.283450
19    16426.725480
20    16838.673200
21    17266.097690
22    16412.083090
23    15875.586730
24    15755.820270
25    16369.317250
26    16699.826680
27    17310.757750
28    16622.671870
29    17581.024140
30    18987.382410
31    18601.397240
32    19232.175560
33    22739.426280
34    25719.147150
35    29198.055690
36    32738.262900
37    36144.481220
38    37446.486090
39    32755.176820
40    38420.522890
41    42334.711210
42    42665.255970
43    42676.468370
44    41039.893600
45    35175.188980
46    34229.193630
Name: income, dtype: float64
```

In [ ]:
```
reg = linear_model.LinearRegression()
reg.fit(new_df,income)
```

Out[ ]:
```
LinearRegression()
```

In [ ]:
```
reg.predict([[2021]])
reg.coef_
reg.intercept_
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feat
ure names
  "X does not have valid feature names, but"
```

Out[ ]:
```
-1632210.7578554575
```

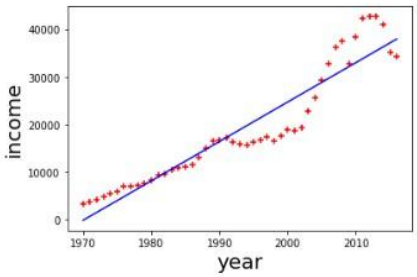
In [ ]:
```
reg.predict([[2020]])
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feat
ure names
  "X does not have valid feature names, but"
```

Out[ ]:
```
array([41288.69409442])
```

In [ ]:
```
plt.xlabel('year',fontsize=20)
plt.ylabel('income',fontsize=20)
plt.scatter(df.year,df.income,color='red',marker='+')
plt.plot(df.year,reg.predict(df[['year']]),color='blue')
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x7f1d402a2590>]
```



LAB 5

Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets

In [9]:
```
from sklearn.datasets import fetch_20newsgroups

data = fetch_20newsgroups()
data.target_names
```

Out[9]:
```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

In [10]:
```
categories = ['talk.religion.misc', 'soc.religion.christian',
              'sci.space', 'comp.graphics']
train = fetch_20newsgroups(subset='train', categories=categories)
test = fetch_20newsgroups(subset='test', categories=categories)
```

In [11]:
```
print(train.data[5])
```

```
From: dmcgee@uluhe.soest.hawaii.edu (Don McGee)
Subject: Federal Hearing
Originator: dmcgee@uluhe
Organization: School of Ocean and Earth Science and Technology
Distribution: usa
Lines: 10


Fact or rumor....?  Madalyn Murray O'Hare an atheist who eliminated the
use of the bible reading and prayer in public schools 15 years ago is now
going to appear before the FCC with a petition to stop the reading of the
Gospel on the airways of America.  And she is also campaigning to remove
Christmas programs, songs, etc from the public schools.  If it is true
then mail to Federal Communications Commission 1919 H Street Washington DC
20054 expressing your opposition to her request.  Reference Petition number

2493.
```

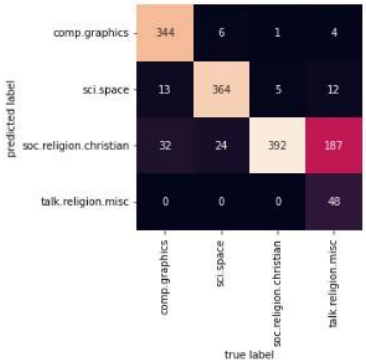
In [12]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline

model = make_pipeline(TfidfVectorizer(), MultinomialNB())
```

In [13]:
```
model.fit(train.data, train.target)
labels = model.predict(test.data)
```

In [17]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
mat = confusion_matrix(test.target, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=train.target_names, yticklabels=train.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



In [18]:

```
def predict_category(s, train=train, model=model):
    pred = model.predict([s])
    return train.target_names[pred[0]]
```

In [22]:

```
predict_category('Rocket launch in 3 months')
```

Out[22]:

```
'sci.space'
```