ML LAB REPORT

Anitej Prasad

1BM19CS194

6-D

D-2

LAB1- FIND S ALGORITHM

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

In [1]:

```python
import csv

def findS(dataset, hypothesis):
    for i in range(len(dataset)):
        if dataset[i][-1] == 'yes':
            print('The tuple', i+1, 'is a positive instance.')
            for j in range(len(hypothesis)):
                if hypothesis[j] == '0'  or dataset[i][j] == hypothesis[j]:
                    hypothesis[j] = dataset[i][j]
                else:
                    hypothesis[j] = '?'
            print('The hypothesis for traning tuple',i+1,'and instance',j+1, 'is:', hypothesis)
        elif dataset[i][-1] == 'no':
            print('The tuple', i+1, 'is a negative instance.')
            print('The hypothesis for traning tuple',i+1, 'is:', hypothesis)
    return hypothesis



def main():

    dataset = []
    with open('FindS-CSV.csv', 'r') as csvfile:
        next(csvfile)
        for row in csv.reader(csvfile):
            dataset.append(row)
        print(dataset)
    hypothesis = ['0']*len(dataset[0])
    print('The Initial hypothesis:', hypothesis)
    hypothesis = findS(dataset, hypothesis)
    print('Final Hypothesis: ', hypothesis)

if __name__ == "__main__":
    main()
```

```
[['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'], ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'], ['rainy', 'cold', 'high', 's
trong', 'warm', 'change', 'no'], ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']]
The Initial hypothesis: ['0', '0', '0', '0', '0', '0', '0']
The tuple 1 is a positive instance.
The hypothesis for traning tuple 1 and instance 7 is: ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
The tuple 2 is a positive instance.
The hypothesis for traning tuple 2 and instance 7 is: ['sunny', 'warm', '?', 'strong', 'warm', 'same', 'yes']
The tuple 3 is a negative instance.
The hypothesis for traning tuple 3 is: ['sunny', 'warm', '?', 'strong', 'warm', 'same', 'yes']
The tuple 4 is a positive instance.
The hypothesis for traning tuple 4 and instance 7 is: ['sunny', 'warm', '?', 'strong', '?', '?', 'yes']
Final Hypothesis:  ['sunny', 'warm', '?', 'strong', '?', '?', 'yes']
```

In [ ]:

LAB2- Candidate Elimination

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 pandas as pd
import numpy as np
import csv


data = pd.read_csv('Candidate-Elimination.csv')
d = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",d)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
```

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

Target Values are:  ['yes' 'yes' 'no' 'yes']
```

In [2]:

```
def learn(d, target):
    specific_h = d[0].copy()
    print("\nSpecific Hypothesis: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Hypothesis: ",general_h)

    for i, h in enumerate(d):
        print("\nIteration", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("Instance is Negative ")
            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] = '?'

        print("Specific Hypothesis after ", i+1, "Instance is ", specific_h)
        print("Generic Hypothesis after ", i+1, "Instance is ", general_h)
        print("\n")

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

In [3]:

```
specific, general = learn(d, target)

print("Final Specific Hypothesis: ", '<', ', '.join(specific),'>')
print("Final General Hypothesis: ")
for i in general:
    print('<', ', '.join(i),'>, ')
```

```
Specific Hypothesis:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Iteration 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
Specific Hypothesis after  1 Instance is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Hypothesis after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Iteration 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
Specific Hypothesis after  2 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Hypothesis after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Iteration 3 is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Negative
Specific Hypothesis after  3 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Hypothesis after  3 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]



Iteration 4 is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Instance is Positive
Specific Hypothesis after  4 Instance is  ['sunny' 'warm' '?' 'strong' '?' '?']
Generic Hypothesis after  4 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific Hypothesis:  < sunny, warm, ?, strong, ?, ? >
Final General Hypothesis:
< sunny, ?, ?, ?, ?, ? >,
< ?, warm, ?, ?, ?, ? >,
```

In [ ]:

LAB3- Decision Tree

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 [1]:

```
import math
import csv
def load_csv(filename):
    lines=csv.reader(open(filename,"r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset,headers
```

In [2]:

```
class Node:
    def __init__(self,attribute):
        self.attribute=attribute
        self.children=[]
        self.answer=""
```

In [3]:

```
def subtables(data,col,delete):
    dic={}
    coldata=[row[col] for row in data]
    attr=list(set(coldata))

    counts=[0]*len(attr)
    r=len(data)
    c=len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col]==attr[x]:
                counts[x]+=1

    for x in range(len(attr)):
        dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
        pos=0
        for y in range(r):
            if data[y][col]==attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos]=data[y]
                pos+=1
    return attr,dic

def entropy(S):
    attr=list(set(S))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums

def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy
```

```python
def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]


    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node
```

In [4]:

```python
def print_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)
        return

    print("  "*level,node.attribute)
    for value,n in node.children:
        print("  "*(level+1),value)
        print_tree(n,level+2)


def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)
```

In [5]:

```python
dataset,features=load_csv("id3.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("id3.csv")

for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:",end="   ")
    classify(node1,xtest,features)
```

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   overcast
     yes
   sunny
     Humidity
       high
         no
       normal
         yes
   rain
     Wind
       strong


The test instance: ['sunny', 'hot', 'high', 'weak', 'no']
The label for test instance:   no
The test instance: ['sunny', 'hot', 'high', 'strong', 'no']
The label for test instance:   no
The test instance: ['overcast', 'hot', 'high', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'mild', 'high', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'cool', 'normal', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'cool', 'normal', 'strong', 'no']
The label for test instance:   no
The test instance: ['overcast', 'cool', 'normal', 'strong', 'yes']
The label for test instance:   yes
The test instance: ['sunny', 'mild', 'high', 'weak', 'no']
The label for test instance:   no
The test instance: ['sunny', 'cool', 'normal', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'mild', 'normal', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['sunny', 'mild', 'normal', 'strong', 'yes']
The label for test instance:   yes
The test instance: ['overcast', 'mild', 'high', 'strong', 'yes']
The label for test instance:   yes
The test instance: ['overcast', 'hot', 'normal', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'mild', 'high', 'strong', 'no']
The label for test instance:   no
```

In [ ]:

LAB4- Linear Regression

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

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [3]:
```
dataset = pd.read_csv('salary_dataset.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
```

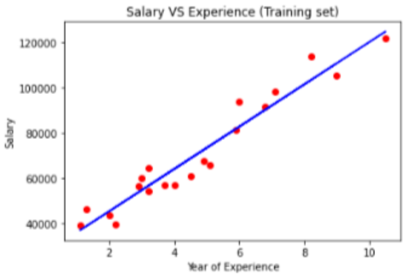
In [4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
```

In [5]:
```
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

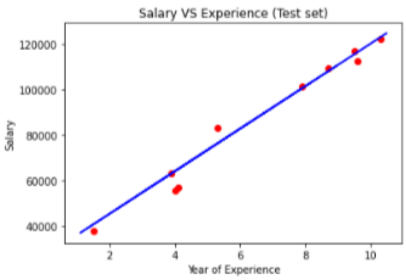
Out[5]:
```
LinearRegression()
```

In [6]:
```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
```

In [7]:
```
# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('Salary VS Experience (Training set)')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()
```



In [8]:
```
# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```



In [ ]:

In [ ]:

LAB5- Naïve Bayes Classifier

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 tests' data set

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

In [2]:
```
data = pd.read_csv('/content/dataset.csv')
data.head()
```

Out[2]:

	PlayTennis	Outlook	Temperature	Humidity	Wind
0	No	Sunny	Hot	High	Weak
1	No	Sunny	Hot	High	Strong
2	Yes	Overcast	Hot	High	Weak
3	Yes	Rain	Mild	High	Weak
4	Yes	Rain	Cool	Normal	Weak

In [3]:
```
y = list(data['PlayTennis'].values)
X = data.iloc[:,1:].values
print(f'Target Values: {y}')
print(f'Features: \n{X}')
```

```
Target Values: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
Features:
[['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']]
```

In [4]:
```
y_train = y[:8]
y_val = y[8:]
X_train = X[:8]
X_val = X[8:]
print(f"Number of instances in training set: {len(X_train)}")
print(f"Number of instances in testing set: {len(X_val)}")
```

```
Number of instances in training set: 8
Number of instances in testing set: 6
```

In [5]:

```
class NaiveBayesClassifier:
    def __init__(self, X, y):
        self.X, self.y = X, y
        self.N = len(self.X)
        self.dim = len(self.X[0])
        self.attrs = [[] for _ in range(self.dim)]
        self.output_dom = {}
        self.data = []
        for i in range(len(self.X)):
            for j in range(self.dim):
                if not self.X[i][j] in self.attrs[j]:
                    self.attrs[j].append(self.X[i][j])
            if not self.y[i] in self.output_dom.keys():
                self.output_dom[self.y[i]] = 1
            else:
                self.output_dom[self.y[i]] += 1
            self.data.append([self.X[i], self.y[i]])
    def classify(self, entry):
        solve = None
        max_arg = -1
        for y in self.output_dom.keys():
            prob = self.output_dom[y]/self.N
            for i in range(self.dim):
                cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y]
                n = len(cases)
                prob *= n/self.N
            if prob > max_arg:
                max_arg = prob
                solve = y
        return solve
```

In [6]:

```
nbc = NaiveBayesClassifier(X_train, y_train)
total_cases = len(y_val)
good = 0
bad = 0
predictions = []
for i in range(total_cases):
    predict = nbc.classify(X_val[i])
    predictions.append(predict)
    if y_val[i] == predict:
        good += 1
    else:
        bad += 1
print('Predicted values:', predictions)
print('Actual values:', y_val)
print()
print('Total number of testing instances in the dataset:', total_cases)
print('Number of correct predictions:', good)
print('Number of wrong predictions:', bad)
print()
print('Accuracy of Bayes Classifier:', good/total_cases)
```

```
Predicted values: ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
Actual values: ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

Total number of testing instances in the dataset: 6
Number of correct predictions: 4
Number of wrong predictions: 2

Accuracy of Bayes Classifier: 0.6666666666666666
```