Machine Learning Lab 9

Simran Kaur

311443

Exercise 1: Implement Decision Tree

In [96]:
```
import pandas as pd
import math
```

Iris Dataset

In [110…
```
data2 = pd.read_csv("iris.data", sep = ",", header = None)
data2.columns = ["sepal length", "sepal width", "petal length", "petal width", "clas
data2
```

Out[110…

	sepal length	sepal width	petal length	petal width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

For each Feature, splits function takes all the unique values that feature has and sort it in ascending order and then takes the average of each of the two consecutive points. This function returns a dictionary with each feature as a key of the dictionary with list of averages as the value corressponding to that key.

In [111…
```
def splits(df):              # input is dataframe
    c1 = {}                          # dictionary for storing features and list of p
    for col in df.columns:
        values = []
        if df[col].dtypes == 'float64':
            l1 = list(set(df[col].unique()))
```

```
            l1.sort()
            for i in range(len(l1) - 1):
                values.append((l1[i] + l1[i+1])/2)
            c1[col] = values
    return c1
```

This function takes a particular feature and the value at which to split rows of the dataframe as left and right branch for further using quality criterion on these branches to know whether they form best splits or not.

In [5]:
```
def branches(df, col, val):
    left = df.loc[df[col] < val]
    right = df.loc[df[col] >= val]
    return left, right
```

Possible classes in our dataframe.

In [112…
```
n_unique_classes = data2["class"].unique()
n_unique_classes
```

Out[112…
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

For each of the left and right branch we will get from branches function we will calculate misclassification rate for each of them to take the split in the end with minimum misclassification rate.

In [114…
```
def misclassificationRate(df1, df2):
    rate_dict_left = {}
    rate_dict_right = {}
    for c in n_unique_classes:          # for each class count number of times it oc
        left_count = 0
        right_count = 0
        for lrow in df1.iterrows():
            if lrow[1]['class'] == c:
                left_count += 1
        rate_dict_left[c] = left_count
        for rrow in df2.iterrows():
            if rrow[1]['class'] == c:
                right_count += 1
        rate_dict_right[c] = right_count
    misclass = ((df1.shape[0] - max(rate_dict_left.values()) + (df2.shape[0] -
                                                                max(rate_dict_right.

    return misclass
```

For example lets check on feature sepal length at value 4.35

In [115…
```
lt, rt = branches(data2, 'sepal length', 4.35)
misclassificationRate(lt, rt)
```

Out[115…
```
0.66
```

Node function sums up whatever has been done so far. For each of the feature and at each of the value that feature will be splitted it calculates misclassification rate and then stores the best

split for that particular feature and then this process is repeated on all of the features and finally we get the best split.

In [116…

```
def Node(df):
    column_values = splits(df)
    min_misclassRate = {}
    for key in column_values:
        ll = []
        for val in column_values[key]:
            left, right = branches(df, key, val)
            ll.append([misclassificationRate(left, right), left, right, val])
        min_misclassRate[key] = min(ll, key = lambda x: x[0])
    val = (min(min_misclassRate.items(), key=lambda x: x[0])[1])[-1]
    name = (min(min_misclassRate.items(), key = lambda x: x[0])[0])
    left_n = (min(min_misclassRate.items(), key=lambda x: x[0])[1])[1]
    right_n = (min(min_misclassRate.items(), key=lambda x: x[0])[1])[2]
    return name, val, left_n, right_n
```

In [117…

```
class CreatingNode:

    def __init__(self, name, value, left, right):
        self.name = name
        self.value = value
        self.left = left
        self.right = right

    def leftnode(self):
        if len(self.left["class"].unique()) == 1:
            self.leftn = self.left["class"][0]
            print(f'{self.name} < {self.value}')
            return self.leftn
        else:
            return Node(self.left)

    def rightnode(self):
        if len(self.right["class"].unique()) == 1:
            self.rightn = self.right["class"][0]
            return self.rightn
        else:
            return Node(self.right)

    def displayTree(self, space):
        print(f'{self.name} < {self.value}'.center(space))
```

In [109…

```
name_r, val_r, left_r, right_r = Node(data2)
rootnode = CreatingNode(name_r, val_r, left_r, right_r)
```

In [79]:

```
try:
    name2, val2, left2, right2 = rootnode.leftnode()
    newnode_1 = CreatingNode(name2, val2, left2, right2)
    newnode_1.displayTree()
except:
    label1 = rootnode.leftnode()
    print(label1)
```

```
Iris-setosa
```

In [75]:
```
try:
    name3, val3, left3, right3 = rootnode.rightnode()
    newnode_r = CreatingNode(name3, val3, left3, right3)
    newnode_r.displayTree()
except:
    print(f"{rootnode.rightnode()}")
```

```
petal length < 4.75
```

I tried implementing tree but it didn't work and I don't where it's wrong.

In [106…
```
def tree(df):
    max_depth = 3
    for i in range(max_depth):


        name_r, val_r, left_r, right_r = Node(df)
        rootnode = CreatingNode(name_r, val_r, left_r, right_r)
        rootnode.displayTree(100 - i*10)

        if not isinstance(left_r, str):

            try:
                name2, val2, left2, right2 = rootnode.leftnode()
                newnode_l = CreatingNode(name2, val2, left2, right2)
                newnode_1.displayTree(100 - (i+1)*20)
            except:
                label1 = rootnode.leftnode()
                print(label1)

        if not isinstance(right_r, str):

            try:
                name3, val3, left3, right3 = rootnode.rightnode()
                newnode_r = CreatingNode(name3, val3, left3, right3)
                newnode_r.displayTree(100 + (i+1)*20)
            except:
                label2 = rootnode.rightnode()
                print(label2)
```

In [107…
```
tree(data2)
```

```
                                    petal length < 2.45
Iris-setosa
                                                  petal length < 4.75
                                petal length < 2.45
Iris-setosa
                                                                petal length < 4.75
                            petal length < 2.45
Iris-setosa
                                                                                petal length <
4.75
```

This function calculates the entropy for a given split by calculating the frequencies of different classes.

In [102…
```
def entropy(df):
    label_count = dict(df["class"].value_counts())
    sum = 0
    for key in label_count:
```

```
        sum += (label_count[key]/df.shape[0])*math.log(label_count[key]/df.shape[0],

    return sum
```

This function calculates the Information gain by using entropy function defined above.

In [104…

```
def informationGain(df, df1, df2):
    return entropy(df) - (df1.shape[0]/df.shape[0])*entropy(df1) - (df2.shape[0]/df.
```

This function splits our dataset into training, validation and test set.

In [120…

```
def trainValTest(df):
    r = int(.7*df.shape[0])
    Train = df.iloc[:r, :]
    Val = df.iloc[r: int(.85*df.shape[0]), :]
    Test = df.iloc[int(.85*df.shape[0]):, :]
    return Train, Val, Test
```

In [121…

```
train, val, test = trainValTest(data2)
train
```

Out[121…

	sepal length	sepal width	petal length	petal width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
…	…	…	…	…	…
100	6.3	3.3	6.0	2.5	Iris-virginica
101	5.8	2.7	5.1	1.9	Iris-virginica
102	7.1	3.0	5.9	2.1	Iris-virginica
103	6.3	2.9	5.6	1.8	Iris-virginica
104	6.5	3.0	5.8	2.2	Iris-virginica

105 rows × 5 columns

In [122…

```
tree(train)
```

```
                                        petal length < 2.45
petal length < 2.45
petal length < 2.45
Iris-setosa
                                                    petal length < 5.05
                                    petal length < 2.45
petal length < 2.45
petal length < 2.45
Iris-setosa
                                                                petal length < 5.05
                                petal length < 2.45
petal length < 2.45
```

```
petal length < 2.45
Iris-setosa
                                                    petal length <
5.05
```

In [ ]: