머신러닝 이론과 실전 HW 6

2021311175 이재현

Question 1

> Write your own code for 1-level decision tree. '1-level decision tree' means a tree that splits only the root node.
>
> a. Use a file named "pid.dat" for the training and 'pidtest.dat' as the test data.
>
> b. Make your program to implement 1-level decision tree only for two classes.
>
> c. Find CART splitting rule assuming all variables are continuous, then split the current node into two subnodes.
>
> d. Print out the 1-level tree information and number of observations from each class.

```
import pandas as pd
import numpy as np
import sys
import statistics as st
```

```
tr_data_name = input("Enter the name of training data file [(ex) pid.dat] : ")

tst_data_name = input("Enter the name of test data file [(ex) pidtest.dat] : ")

coding_fm = int(input("Select the data coding format(1 = 'a b c' or 2 =
'a,b,c'):"))

if(coding_fm == 1) : separator_fm = " "
else : separator_fm = ","

res_pos = int(input("Enter the column position of the response variable : [from
1 to p] : "))

header= input("Does the data have column header? (True/False) : ")

if(header == 'True'):
    trdata = pd.read_csv(tr_data_name, sep=separator_fm)
    tstdata = pd.read_csv(tst_data_name, sep=separator_fm)
else:
    trdata = pd.read_csv(tr_data_name, sep=separator_fm, header=None)
    tstdata = pd.read_csv(tst_data_name, sep=separator_fm, header=None)

out_name=input("Enter the output file name to export [(ex) result.txt] : ")
```

```
Enter the name of training data file [(ex) pid.dat] : pid.dat
Enter the name of test data file [(ex) pidtest.dat] : pidtest.dat
Select the data coding format(1 = 'a b c' or 2 = 'a,b,c'):2
Enter the column position of the response variable : [from 1 to p] : 8
Does the data have column header? (True/False) : False
Enter the output file name to export [(ex) result.txt] : result_6.txt
```

```
def My_CART():
    X_tr = trdata.drop(res_pos-1, axis=1)
    Y_tr = trdata[res_pos-1]
    X_tst = tstdata.drop(res_pos-1, axis=1)
    Y_tst = tstdata[res_pos-1]
    df = pd.DataFrame(columns = ['i','j','i_t1'])

    for i in range(X_tr.shape[1]):
        target_X = X_tr.iloc[:,i]
        unisort_dummy = np.sort(target_X.unique())
        for j in range(1, len(unisort_dummy)):
            vector_L = Y_tr[target_X < unisort_dummy[j]]
            vector_R = Y_tr[target_X >= unisort_dummy[j]]
            p_L1 = sum(vector_L == Y_tr.unique()[0]) / len(vector_L)
            p_L2 = sum(vector_L == Y_tr.unique()[1]) / len(vector_L)
            p_R1 = sum(vector_R == Y_tr.unique()[0]) / len(vector_R)
            p_R2 = sum(vector_R == Y_tr.unique()[1]) / len(vector_R)
            i_L = 1 - p_L1**2 - p_L2**2
            i_R = 1 - p_R1**2 - p_R2**2
            w_L = len(vector_L) / len(Y_tr)
            w_R = len(vector_R) / len(Y_tr)
            i_t1 = i_L * w_L + i_R * w_R
            df = df.append({'i': i, 'j': j, 'i_t1': i_t1}, ignore_index = True)

    answer_key = df.loc[df.iloc[:,2] == min(df.iloc[:,2]),:]
    answer_key = np.array(answer_key)[0]
    target_X = X_tr.iloc[:,int(answer_key[0])]
    unisort_dummy = np.sort(target_X.unique())
    vector_L = Y_tr[target_X < unisort_dummy[int(answer_key[1])]]
    vector_R = Y_tr[target_X >= unisort_dummy[int(answer_key[1])]]
    max(sum(vector_L == Y_tr.unique()[0]), sum(vector_L == Y_tr.unique()[0]))

    Prediction = []
    for i in range(len(Y_tst)):
        if X_tst.iloc[i, int(answer_key[0])] <
np.sort(X_tr.iloc[:,int(answer_key[0])].unique())[int(answer_key[1])]:
            Y_pred = st.mode(vector_L)
        else:
            Y_pred = st.mode(vector_R)
        Prediction.append(Y_pred)
    accuracy_tst = sum(Prediction == Y_tst) / len(Y_tst)

    df = pd.DataFrame()
    df['Actual'] = Y_tst
    df['Prediction'] = Prediction
    df['count'] = np.ones(len(Y_tst))
    confusion_matrix = pd.pivot_table(df, values='count', index=['Actual'],
                                columns=['Prediction'], aggfunc=np.sum,
fill_value=0)

    sys.stdout = open(out_name,'w')
    print("Tree Structure (#Class{0}, #Class{1})".format(
        Y_tr.unique()[0], Y_tr.unique()[1]))
    print("Node 1: X{0} < {1} {2}".format(
        int(answer_key[0]) + 1,
np.sort(X_tr.iloc[:,int(answer_key[0])].unique())[int(answer_key[1])],
        (sum(Y_tr == Y_tr.unique()[0]),sum(Y_tr == Y_tr.unique()[1]))))
```

```
    print("Node 2: {0} {1}".format(st.mode(vector_L),(sum(vector_L ==
Y_tr.unique()[0]), sum(vector_L == Y_tr.unique()[1]))))
    print("Node 3: {0} {1}".format(st.mode(vector_R),(sum(vector_R ==
Y_tr.unique()[0]), sum(vector_R == Y_tr.unique()[1]))))
    print("\nConfusion Matrix (Test)\n--------------------------")
    print(confusion_matrix)
    print('\nModel Summary (Test)\n--------------------------')
    print("Overall Accuracy = {}".format(np.round(accuracy_tst, 3)))
```

$$\Delta i(s,1) = i(1) - w_L \cdot i(1_L) - w_R \cdot i(1_R)$$
$$\Delta i(s^*,1) = \max \Delta_{s \in S}\, i(s,1)$$

- The idea of finding **minimum impurity** using greedy search is applied to `My_CART()`.
- By obtaining the `answer_key`, the partition that splits the training data most efficiently is found.
- To obtain the **accuracy**, the sum of the diagonal elements of the confusion matrix is divided by the size of the test data.

```
My_CART()
```

The results are saved as `result_6.txt`.

```
Tree Structure (#Class2, #Class1)
Node 1: X2 < 147 (180, 72)
Node 2: 2 (164, 34)
Node 3: 1 (16, 38)

Confusion Matrix (Test)
--------------------------
Prediction   1    2
Actual
1           37   50
2           11  128

Model Summary (Test)
--------------------------
Overall Accuracy = 0.73
```

- Since `Y_tr.unique()` is [2, 1], **the first class is 2 and the second class is 1**. It may be counterintuitive in this case.
- Node 2 corresponds to the data with X2 < 147
- Node 3 corresponds to the data with X2 ≥ 147