# Name and ID

Jianan Xu

# HW05 Code

You will complete the following notebook, as described in the PDF for Homework 05 (included in the download with the starter code). You will submit:

1. This notebook file, along with your COLLABORATORS.txt file and the two tree images (PDFs generated using `graphviz` within the code), to the Gradescope link for code.
2. A PDF of this notebook and all of its output, once it is completed, to the Gradescope link for the PDF.

Please report any questions to the class Piazza page.

## Import required libraries.

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

import sklearn.tree
import graphviz
```

# Decision Trees

You should start by computing the two heuristic values for the toy data described in the assignment handout. You should then load the two versions of the abalone data, compute the two heuristic values on features (for the simplified data), and then build decision trees for each set of data.

## 1 Compute both heuristics for toy data.

### (a) Compute the counting-based heuristic, and order the features by it.

In [2]:
```
# For the left part of A, the majority is O, and for the right part of A, the majority is X. After counting, the left side are two
# which is all correct. The right side are two O and 4 X, so there are two mistakes (two O). Overall, the answer for A should be 6/

# Using the same way of computing, For B, the majority of left side is O, and there is one mistake (one X). For the right side, the
# is X, and there is one mistake (one X). So the answer for B is 6/8.
```

### (b) Compute the information-theoretic heuristic, and order the features by it.

In [3]:
```
# For the A:
# the H(A) = -(4/8 * log2(4/8) + 4/8 * log2(4/8)) = 1
# the Remainder(A) = (2/8 * H(A1) + 6/8 * H(A2))
# H(A1) = -(0/2 * log2(0/2) + 2/2 * log2(2/2)) = 0
# H(A2) = -(2/6 * log2(2/6) + 4/6 * log2(4/6)) = 0.918
# Remainder(A) = 0.6885

# Gain(A) = H(A) = Remainder(A) = 1-0.6885 = 0.3115

#For the B:
# Using the same way of computing, the H(A) = H(B) = 1
# the Remainder(B) = (4/8 * H(B1) + 4/8 * H(B2))
# H(B1) = -(3/4 * log2(3/4) + 1/4 * log2(1/4)) = 0.81
# H(B2) = -(3/4 * log2(3/4) + 1/4 * log2(1/4)) = 0.81
# Remainder(B) = 0.81

# Gain(B) = H(B) = Remainder(B) = 1-0.81 = 0.19
```

### (c) Discussion of results.

so if we use the part A's answer to get the results, their accuracy is same, so we cannot distinguish which one is better. if we use the part B's answer, the gain for A is higher, which means that the A is better for this situation.

## 2 Compute both heuristics for simplified abalone data.

### (a) Compute the counting-based heuristic, and order the features by it.

In [196…
```
x_test = np.loadtxt('./data_abalone/small_binary_x_test.csv', delimiter=',', skiprows=1)
x_train = np.loadtxt('./data_abalone/small_binary_x_train.csv', delimiter=',', skiprows=1)
```

```python
y_test = np.loadtxt('./data_abalone/3class_y_test.csv', delimiter=',', skiprows=1)
y_train = np.loadtxt('./data_abalone/3class_y_train.csv', delimiter=',', skiprows=1)


top_row = pd.read_csv('./data_abalone/small_binary_x_train.csv', nrows=0)
full_feature_names = np.array(top_row.columns)
classes = [0, 1, 2]

def get_elements(index):
    m0_0 = 0
    m0_1 = 0
    m0_2 = 0
    m1_0 = 0
    m1_1 = 0
    m1_2 = 0
    l = []

    for i in range(len(x_train)):
        is_male = x_train[i][index]
        label = y_train[i]

        if is_male == 0 and label == 0:
            m0_0 = m0_0 + 1
        if is_male == 0 and label == 1:
            m0_1 = m0_1 + 1
        if is_male == 0 and label == 2:
            m0_2 = m0_2 + 1

        if is_male == 1 and label == 0:
            m1_0 = m1_0 + 1
        if is_male == 1 and label == 1:
            m1_1 = m1_1 + 1
        if is_male == 1 and label == 2:
            m1_2 = m1_2 + 1

    l.append(m0_0)
    l.append(m0_1)
    l.append(m0_2)
    l.append(m1_0)
    l.append(m1_1)
    l.append(m1_2)
    return l
```

```
def get_ans(l):

    big_0 = 0
    length = int(len(l) / 2)
    total_0 = 0
    for i in range(0, length):
        if big_0 < l[i]:
            big_0 = l[i]
        total_0 = total_0 + l[i]

    big_1 = 0
    total_1 = 0
    for i in range(length, len(l)):
        if (big_1 < l[i]):
            big_1 = l[i]
        total_1 = total_1 + l[i]

    ans = []
    ans.append(big_0 + big_1)
    ans.append(total_0 + total_1)
    return ans


lst = []
ans_list1 = []

for i in range(len(x_train[0])):
    l = get_elements(i)
    ans = get_ans(l)
    lst.append(l)
    ans_list1.append(ans)

ans_list1.sort(reverse=True)

print("height_mm: ")
print("diam_mm: ")
print("length_mm:  ")
print("is_male: ")
print(ans_list1)
```

```
height_mm:
diam_mm:
length_mm:
is_male:
[[2316, 3176], [2266, 3176], [2230, 3176], [1864, 3176]]
```

### (b) Compute the information-theoretic heuristic, and order the features by it.

In [199…

```
# For the A:
# the H(A) = -(4/8 * log2(4/8) + 4/8 * log2(4/8)) = 1
# the Remainder(A) = (2/8 * H(A1) + 6/8 * H(A2))
# H(A1) = -(0/2 * log2(0/2) + 2/2 * log2(2/2)) = 0
# H(A2) = -(2/6 * log2(2/6) + 4/6 * log2(4/6)) = 0.918
# Remainder(A) = 0.6885

# Gain(A) = H(A) = Remainder(A) = 1-0.6885 = 0.3115

#For the B:
# Using the same way of computing, the H(A) = H(B) = 1
# the Remainder(B) = (4/8 * H(B1) + 4/8 * H(B2))
# H(B1) = -(3/4 * log2(3/4) + 1/4 * log2(1/4)) = 0.81
# H(B2) = -(3/4 * log2(3/4) + 1/4 * log2(1/4)) = 0.81
# Remainder(B) = 0.81

# Gain(B) = H(B) = Remainder(B) = 1-0.81 = 0.19


def get_gain(l):
    total_0 = 0
    length = int(len(l)/2)
    for i in range(0, length):                                  #all branches 0 (x_train)
        total_0 = total_0 + l[i]

    total_1 = 0
    for i in range(length, len(l)):
        total_1 = total_1 + l[i]                               #all branches 1 (x_train)


    m_0 = l[0] + l[3]                                           #all 0
    m_1 = l[1] + l[4]                                           #all rectangle
    m_2 = l[2] + l[5]                                           #all X
    total_m = m_0 + m_1 + m_2

    m0_0 = l[0]
    m0_1 = l[1]
    m0_2 = l[2]

    m1_0 = l[3]
    m1_1 = l[4]
```

```
    m1_2 = l[5]

    H = -(m_0/total_m * np.log2(m_0/total_m) + (m_1/total_m) * np.log2(m_1/total_m) + (m_2/total_m) * np.log2(m_2/total_m))


    H_1 = -(m0_0/total_0 * np.log2(m0_0/total_0) + m0_1/total_0 * np.log2(m0_1/total_0) + m0_2/total_0 * np.log2(m0_2/total_0))
    H_2 = -(m1_0/total_1 * np.log2(m1_0/total_1) + m1_1/total_1 * np.log2(m1_1/total_1) + m1_2/total_1 * np.log2(m1_2/total_1))

    R = (total_0/total_m) * H_1 + (total_1/total_m) * H_2
    G = H-R
    return G

ans_list = []
for i in range(len(lst)):
    ans = get_gain(lst[i])
    ans_list.append(ans)

print("height_mm: ")
print("diam_mm: ")
print("length_mm:  ")
print("is_male: ")
ans_list.sort(reverse=True)
print(ans_list)
```

```
height_mm:
diam_mm:
length_mm:
is_male:
[0.17302867291002477, 0.1500706886802703, 0.13543816377043694, 0.024516482271752293]
```

In [123…

```
# # test on question 1
# x_train = [[0],[0],[0], [0], [1], [1], [1], [1]]
# y_train = [0, 0, 0, 1, 0, 1, 1, 1]
# m = get_elements(0)
# new_lst = []
# for i in range(len(m)):
#     if i == 2 or i == 5:
#         continue
#     new_lst.append(m[i])

# print(new_lst)
```

```
# total_0 = 0
# length = int(len(new_lst)/2)

# for i in range(0, length):
#     total_0 = total_0 + new_lst[i]

# total_1 = 0
# for i in range(length, len(new_lst)):
#     total_1 = total_1 + new_lst[i]

# m_0 = new_lst[0] + new_lst[2]                                    #all 0
# m_1 = new_lst[1] + new_lst[3]                                    #all 1
# total_m = m_0 + m_1

# print(total_0)
# print(total_1)
# H = -(m_0/total_m * np.log2(m_0/total_m) + (m_1/total_m) * np.log2(m_1/total_m))
# print(H)


# m0_0 = new_lst[0]
# m0_1 = new_lst[1]
# m1_0 = new_lst[2]
# m1_1 = new_lst[3]




# H_1 = -(m0_0/total_0 * np.log2(m0_0/total_0) + m0_1/total_0 * np.log2(m0_1/total_0))
# H_2 = -(m1_0/total_1 * np.log2(m1_0/total_1) + m1_1/total_1 * np.log2(m1_1/total_1))
# print(H_1)
# print(H_2)

# R = (total_0/total_m) * H_1 + (total_1/total_m) * H_2
# G = H-R

# print(G)
```

```
[3, 1, 1, 3]
4
4
1.0
0.8112781244591328
0.8112781244591328
0.18872187554086717
```

# 3 Generate decision trees for full- and restricted-feature data

## (a) Print accuracy values and generate tree images.

In [200…
```
com_x_test = np.loadtxt('./data_abalone/x_test.csv', delimiter=',', skiprows=1)
com_x_train = np.loadtxt('./data_abalone/x_train.csv', delimiter=',', skiprows=1)
com_y_test = np.loadtxt('./data_abalone/y_test.csv', delimiter=',', skiprows=1)
com_y_train = np.loadtxt('./data_abalone/y_train.csv', delimiter=',', skiprows=1)
```

In [187…
```
import sklearn.tree

classifier = sklearn.tree.DecisionTreeClassifier(criterion= 'entropy')
classifier.fit(x_train, y_train)

train_score = classifier.score(x_train, y_train)
test_score = classifier.score(x_test, y_test)

print("train_score is: ")
print(train_score)
print("test_score is: ")
print(test_score)
```

```
train_score is:
0.7326826196473551
test_score is:
0.722
```

In [188…
```
from sklearn.tree import export_graphviz

fit_value = classifier.fit(x_train, y_train)
graph = sklearn.tree.export_graphviz(fit_value, out_file=None)

final_pdf = graphviz.Source(graph)
final_pdf.render("simplified_final_pdf")
```

Out[188]:

```
'simplified_final_pdf.pdf'
```

In [189…

```
com_classifier = sklearn.tree.DecisionTreeClassifier(criterion='entropy')
com_classifier.fit(com_x_train, com_y_train)

com_train_score = com_classifier.score(com_x_train, com_y_train)
com_test_score = com_classifier.score(com_x_test, com_y_test)

print("train_score is: ")
print(com_train_score)
print("test_score is: ")
print(com_test_score)
```

```
train_score is:
1.0
test_score is:
0.196
```

In [190…

```
com_fit_value = classifier.fit(com_x_train, com_y_train)
com_graph = sklearn.tree.export_graphviz(com_fit_value, out_file=None)

com_final_pdf = graphviz.Source(com_graph)
com_final_pdf.render("complicated_final_pdf")
```

Out[190]:

```
'complicated_final_pdf.pdf'
```

### (b) Discuss the results seen for the two trees

*TODO*

In [ ]:

```
# I can see the score is different. for the complicated version of data, it is a 1 for train_score and 0.2 for test_score, which me
# this is a overfitting. The score for the simplified version of data seems normal.

# the two trees are different that one is pretty simple and has 2 branches with 3 features. Meanwhile, the complicated tree has ton
# branches and features, which can produce a very large decision tree.

# The mistake that the simple version of data make is that it has two branches (two leaves) every time the tree spread out. There i
# mistake that for example, the most left bottom value, it suppose to be 0 on label 1 and 2. However, there still value on the labe
# which is different to the ideal situation.
```