Murt Sayeed

TODO

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]:
```
# TODO
#d = dict() dictionary to store the number of labels in each attribute
d_A = {'True': {'circle':2, 'cross':0}, 'False': {'circle':2, 'cross':4} }
d_B = {'True': {'circle':3, 'cross':1}, 'False': {'circle':1, 'cross':3} }

def guess_dict(d):
    for key in d:
        if d[key]['circle'] >= d[key]['cross']:
            d[key]['guess'] = 'circle'
        else:
            d[key]['guess'] = 'cross'

guess_dict(d_A)
guess_dict(d_B)
```

```python
def compute_score(d):
    num_correct = 0
    num_total = 0
    for key in d:
        num_correct += d[key][d[key]['guess']]
        num_total += d[key]['circle']
        num_total += d[key]['cross']
    score = num_correct/num_total
    return score, num_correct, num_total

score_A, corr_A, total_A = compute_score(d_A)
score_B, corr_B, total_B = compute_score(d_B)

if score_A >= score_B:
    print('Feature A: ', corr_A, '/', total_A, sep='')
    print('Feature B: ', corr_B, '/', total_B, sep='')
else:
    print('Feature B: ', corr_B, '/', total_B, sep='')
    print('Feature A: ', corr_A, '/', total_A, sep='')
```

```
Feature A: 6/8
Feature B: 6/8
```

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

In [3]:

```python
# TODO
d_All = {'circle': 4, 'cross': 4}
d_A = {'True': {'circle':2, 'cross':0}, 'False': {'circle':2, 'cross':4} }
d_B = {'True': {'circle':3, 'cross':1}, 'False': {'circle':1, 'cross':3} }

def Entropy(d):
    circle = d['circle']
    cross = d['cross']
    total = sum(d.values())
    if circle == 0:
        return -1*(cross/total*np.log2(cross/total))
    elif cross == 0:
        return -1*(circle/total*np.log2(circle/total))
    else:
        return -1*(circle/total*np.log2(circle/total) + cross/total*np.log2(cros

def Gain(d, entropy_fullset):
    entr_Tr  = Entropy(d['True'])
    count_Tr = sum(d['True'].values())
    entr_Fa  = Entropy(d['False'])
    count_Fa = sum(d['False'].values())
    total = count_Tr + count_Fa

    return entropy_fullset - (count_Tr / total * entr_Tr + count_Fa / total * en


ent_All = Entropy(d_All)

gain_A = Gain(d_A, ent_All)
gain_B = Gain(d_B, ent_All)

if gain_A >= gain_B:
    print('Feature A: %.3f' % gain_A)
    print('Feature B: %.3f' % gain_B)
```

```
else:
    print('Feature B: %.3f' % gain_B)
    print('Feature A: %.3f' % gain_A)
```

```
Feature A: 0.311
Feature B: 0.189
```

(c) Discussion of results.

TODO Discuss the results: if we built a tree using each of these heuristics, what would happen? What does this mean?

We have to look at two methods and see what can give us better results and information. The counting-based is simple condition and information-theoretic can provide better and accurate information. The counting-based divides the data into two groups that are identical and didn't find a difference b/w the scores of each feature (A & B). These were a same/tie because of the nature of counting-based method. Our main goal should be to have better and more accurate feature out of all and reduce complexity as well. In information-theoretic method, we concluded Feature A having 0.311 of information gain and Feature B having 0.189, where Feature A is more important than Feature B. Thus, the second method of information-theoretic is a better option.

2 Compute both heuristics for simplified abalone data.

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

In [4]:

```
# TODO
X_train_simple = pd.read_csv('data_abalone/small_binary_x_train.csv')
X_test_simple = pd.read_csv('data_abalone/small_binary_x_test.csv')
y_train_simple = pd.read_csv('data_abalone/3class_y_train.csv')
y_test_simple = pd.read_csv('data_abalone/3class_y_test.csv')

features = []

for c in list(X_train_simple.columns):
    var0_all_ys = y_train_simple[X_train_simple[c]==0]
    var1_all_ys = y_train_simple[X_train_simple[c]==1]

    var0_y0 = (var0_all_ys == 0).sum()['rings']
    var0_y1 = (var0_all_ys == 1).sum()['rings']
    var0_y2 = (var0_all_ys == 2).sum()['rings']

    var1_y0 = (var1_all_ys == 0).sum()['rings']
    var1_y1 = (var1_all_ys == 1).sum()['rings']
    var1_y2 = (var1_all_ys == 2).sum()['rings']

    numerator_c = max(var0_y0, var0_y1, var0_y2) + max(var1_y0, var1_y1, var1_y2
    denominator_c = var0_y0 + var0_y1 + var0_y2 + var1_y0 + var1_y1 + var1_y2
    score_c = numerator_c / denominator_c

    features.append((c,score_c, numerator_c, denominator_c))

features.sort(key=lambda x: x[1], reverse=True)

for f in features:
    print(f[0], ': ', f[2], '/', f[3], sep='')
```

```
height_mm: 2316/3176
diam_mm: 2266/3176
length_mm: 2230/3176
is_male: 1864/3176
```

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

In [5]:

```python
# TODO
features_gain = []

def Entropy_aba(*args):
    total = 0
    for arg in args:
        total += arg
    entro = 0
    for arg in args:
        p_cat = arg / total
        entro += (p_cat * np.log2(p_cat))
    return -entro

entro_fullset = Entropy_aba((y_train_simple == 0).sum()['rings'],(y_train_simple

for c in list(X_train_simple.columns):
    var0_all_ys = y_train_simple[X_train_simple[c]==0]
    var1_all_ys = y_train_simple[X_train_simple[c]==1]

    var0_y0 = (var0_all_ys == 0).sum()['rings']
    var0_y1 = (var0_all_ys == 1).sum()['rings']
    var0_y2 = (var0_all_ys == 2).sum()['rings']

    var1_y0 = (var1_all_ys == 0).sum()['rings']
    var1_y1 = (var1_all_ys == 1).sum()['rings']
    var1_y2 = (var1_all_ys == 2).sum()['rings']

    entro_0 = Entropy_aba(var0_y0, var0_y1, var0_y2)
    entro_1 = Entropy_aba(var1_y0, var1_y1, var1_y2)

    ratio_0 = var0_all_ys.size / y_train_simple.size
    ratio_1 = var1_all_ys.size / y_train_simple.size

    gain = entro_fullset - (ratio_0 * entro_0 + ratio_1 * entro_1)

    features_gain.append((c, gain))

features_gain.sort(key=lambda x: x[1], reverse=True)

for f in features_gain:
    print(f[0], ': ', f[1], sep='')
```

```
height_mm: 0.17302867291002477
diam_mm: 0.1500706886802703
length_mm: 0.13543816377043694
is_male: 0.024516482271752293
```

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

(a) Print accuracy values and generate tree images.

In [9]:

```python
# TODO
```

```python
X_train_full = pd.read_csv('data_abalone/x_train.csv')
X_test_full = pd.read_csv('data_abalone/x_test.csv')
y_train_full = pd.read_csv('data_abalone/y_train.csv')
y_test_full = pd.read_csv('data_abalone/y_test.csv')


dec_full = sklearn.tree.DecisionTreeClassifier(criterion='entropy')
dec_full.fit(X_train_full, y_train_full)
train_full_score = dec_full.score(X_train_full, y_train_full)
test_full_score = dec_full.score(X_test_full, y_test_full)

print('FULL DATA-SET')
print('Accuracy Training Set: %.4f' % train_full_score)
print('Accuracy Testing Set: %.4f' % test_full_score)
print('')

dec_simple = sklearn.tree.DecisionTreeClassifier(criterion='entropy')
dec_simple.fit(X_train_simple, y_train_simple)
train_simple_score = dec_simple.score(X_train_simple, y_train_simple)
test_simple_score = dec_simple.score(X_test_simple, y_test_simple)

print('SIMPLIFIED DATA-SET')
print('Accuracy Training Set: %.4f' % train_simple_score)
print('Accuracy Testing Set: %.4f' % test_simple_score)
print('')

dec_full_data = sklearn.tree.export_graphviz(dec_full, out_file=None)
graph = graphviz.Source(dec_full_data)
graph.render("full");

dec_simple_data = sklearn.tree.export_graphviz(dec_simple, out_file=None)
graph_simple = graphviz.Source(dec_simple_data)
graph_simple.render("simple");
```

```
FULL DATA-SET
Accuracy Training Set: 1.0000
Accuracy Testing Set: 0.1840

SIMPLIFIED DATA-SET
Accuracy Training Set: 0.7327
Accuracy Testing Set: 0.7220
```

(b) Discuss the results seen for the two trees

If we analyze the two decision trees, the simplified data-set model is smaller compared to full data-set, where there is much more depth and large number of leaves. It seems the training accuracy (100%) is better in full data-set for obviously reasons since we have more features. But testing accuracy (18.4%) is very poor. This issue is likely due to overfitting and we shouldn't generalizd this method to other cases and examples. We have tried to use full data-set which contains all the feature and every single details, and the method will conclude in overfitting to fit all our inputs from the full data-set. When we have large tree in full data-set with lots of branches and leaves, there has to be overfitting problem. To fix the problem, we should utilize smaller data-set b/w our true condition vs. the prediction. The simplified model has the training accuracy (73.3%) around the same level as testing accuracy (72.2%), which reflects a better model and doesn't cause the overfitting issue. The tree level in terms of leaves, branches and depth is much better in simplified model.