Nick Murphy

nmurph03

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 (https://piazza.com/tufts/spring2021/comp135).

Import required libraries.

In [4]:
```
import numpy as np
import pandas as pd

import sklearn.tree
```

In [5]:
```
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 [6]:
```
def counting_heuristic(toy_x, toy_y):
    features = list()
    num_ft = np.shape(toy_x)[1]
    i = 0
    while i < num_ft:
        correct = 0
        r = 0
        while r < np.size(toy_y):
            if(toy_x[r,i] == toy_y[r]):
                correct += 1
            r+=1
        features.append(correct/len(toy_y))
        i+=1
    return features
```

In [7]:
```
toy_x = np.array([[1,1,0,0,0,0,0,0],[1,1,1,0,1,0,0,0]])
toy_x = np.transpose(toy_x)
toy_y = np.array([1,1,1,1,0,0,0,0])
toys_list = counting_heuristic(toy_x, toy_y)
print("A: %3f" %toys_list[0])
print("B: %3f" %toys_list[1])
```

```
A: 0.750000
B: 0.750000
```

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

In [8]:
```
def entropy(val):
    if val == 0:
        res = 0
    else:
        res = -1 * val * np.log2(val)
    return res
```

In [156]:
```
def remainder(toy_x, toy_y, num_ft, i):
    total = len(toy_x)
    num_pos = 0
    num_neg = 0
    rem = 0
    in_pos = 0
    in_neg = 0
    r = 0
    while r < total:
        if(toy_x[r,i] == 1):
            num_pos +=1
            if(toy_y[r] == 1):
                in_pos +=1
        elif(toy_x[r,i] == 0):
            num_neg +=1
            if(toy_y[r] == 0):
                in_neg +=1
        r+=1
    pos = num_pos/(total)
    neg = num_neg/(total)
    rem += pos * (entropy(in_pos/num_pos) - entropy(in_neg/num_neg))
    rem += neg * (entropy(in_pos/num_pos) - entropy(in_neg/num_neg))
    return rem
```

In [157]:
```
def infotheory_heuristic(toy_x, toy_y):
    features = list()
    num_ft = np.shape(toy_x)[1]
    i=0
    while i < num_ft:
        num_pos = 0
        num_neg = 0
        r = 0
        while r < np.size(toy_y):
            if(toy_x[r,i] == 1):
                num_pos +=1
            elif(toy_x[r,i] == 0):
                num_neg +=1
            r+=1
        pos = num_pos/(num_pos + num_neg)
        neg = num_neg/(num_pos + num_neg)
        ent = (entropy(pos) + entropy(neg))/len(toy_y)
        rem = remainder(toy_x,toy_y,num_ft, i)
        gain = ent - rem
        features.append(gain)
        i+=1
    return features
```

In [158]:
```
toys_list = infotheory_heuristic(toy_x, toy_y)
print("A: %3f" %toys_list[0])
print("B: %3f" %toys_list[1])
```

```
A: 0.491385
B: 0.125000
```

(c) Discussion of results.

If we built a tree using these heuristics, the tree using the information theory based heuristic would provide insight into the difference in information gained from the two different features, while the counting based heuristic would not.

2 Compute both heuristics for simplified abalone data.

Load Data

In [62]:
```
x_train_s = np.loadtxt('./data_abalone/small_binary_x_train.csv', delimiter
x_test_s = np.loadtxt('./data_abalone/small_binary_x_test.csv', delimiter='
y_train_s = np.loadtxt('./data_abalone/3class_y_train.csv', delimiter=',',
y_test_s = np.loadtxt('./data_abalone/3class_y_test.csv', delimiter=',', sk
```

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

In [179]:
```
def remainder_p2(toy_x, toy_y, num_ft, i):
    total = len(toy_x)
    num_z = 0
    num_one = 0
    num_two = 0
    rem = 0
    in_one = 0
    in_two = 0
    in_z = 0
    r = 0
    while r < total:
        if(toy_x[r,i] == 1):
            num_one +=1
            if(toy_y[r] == 1):
                in_one +=1
        elif(toy_x[r,i] == 0):
            num_z +=1
            if(toy_y[r] == 0):
                in_z +=1
        elif(toy_x[r,i] == 2):
            num_two +=1
            if(toy_y[r] == 2):
                in_two +=1
        r+=1
    one = num_one/total
    two = num_two/total
    zero = num_z/total
    rem += one * (entropy(in_one/num_one) - entropy(0) - entropy(in_z/num_z
    rem += two * (entropy(in_one/num_one) - entropy(0) - entropy(in_z/num_z
    rem += zero * (entropy(in_one/num_one) - entropy(0) - entropy(in_z/num_
    return rem
```

In [180]:
```
def infotheory_heuristic_p2(toy_x, toy_y):
    features = list()
    num_ft = np.shape(toy_x)[1]
    i=0
    while i < num_ft:
        num_z = 0
        num_one = 0
        num_two = 0
        r = 0
        while r < np.size(toy_y):
            if(toy_x[r,i] == 1):
                num_one +=1
            elif(toy_x[r,i] == 0):
                num_z +=1
            elif(toy_x[r,i] == 2):
                num_two +=1
            r+=1
        total = num_one + num_z + num_two
        one = num_one/total
        two = num_two/total
        zero = num_z/total
        ent = entropy(one) + entropy(two) + entropy(zero)
        rem = remainder_p2(toy_x,toy_y,num_ft, i)
        gain = ent - rem
        features.append(gain)
        i+=1
    return features
```

In [181]:
```
toys_list = counting_heuristic(x_train_s, y_train_s)
print("is_male: %3f" %toys_list[0])
print("length_mm: %3f" %toys_list[1])
print("diam_mm: %3f" %toys_list[2])
print("height_mm: %3f" %toys_list[3])
```

```
is_male: 0.586902
length_mm: 0.702141
diam_mm: 0.713476
height_mm: 0.729219
```

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

In [182]:
```
toys_list = infotheory_heuristic_p2(x_train_s, y_train_s)
print("is_male: %f" %toys_list[0])
print("length_mm: %f" %toys_list[1])
print("diam_mm: %f" %toys_list[2])
print("height_mm: %f" %toys_list[3])
```

```
is_male: 0.957996
length_mm: 0.911196
diam_mm: 0.915771
height_mm: 0.919150
```

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

(a) Print accuracy values and generate tree images.

In [12]:
```
# Load data
x_train = np.loadtxt('data_abalone/x_train.csv', skiprows=1, delimiter=',')
x_test = np.loadtxt('data_abalone/x_test.csv', skiprows=1, delimiter=',')
y_train = np.loadtxt('data_abalone/y_train.csv', skiprows=1, delimiter=',')
y_test = np.loadtxt('data_abalone/y_test.csv', skiprows=1, delimiter=',')
```

In [26]:

```
import csv
opened = open('data_abalone/small_binary_x_train.csv','r')
reader = csv.reader(opened)
features = next(reader)
print(features)
dt = sklearn.tree.DecisionTreeClassifier(criterion = 'entropy')
dt = dt.fit(x_train_s,y_train_s)
dt_test_score = dt.score(x_test_s,y_test_s)
dt_train_score = dt.score(x_train_s,y_train_s)
print ("Accuracy in testing data set is: ", dt_test_score)
print ("Accuracy in training data set is: ", dt_train_score)
dot_data = sklearn.tree.export_graphviz(dt,out_file=None,filled = True, fea
graph = graphviz.Source(dot_data)
graph.render("simplified_dataset")
graph
```

```
['is_male', 'length_mm', 'diam_mm', 'height_mm']
Accuracy in testing data set is:  0.722
Accuracy in training data set is:  0.7326826196473551
```

Out[26]:

In [27]:
```
x_train_read = csv.reader(open('./data_abalone/x_train.csv','rt'))
feature_list = next(x_train_read)
x_train = np.loadtxt('data_abalone/x_train.csv', skiprows=1, delimiter=',')
x_test = np.loadtxt('data_abalone/x_test.csv', skiprows=1, delimiter=',')
y_train = np.loadtxt('data_abalone/y_train.csv', skiprows=1, delimiter=',')
y_test = np.loadtxt('data_abalone/y_test.csv', skiprows=1, delimiter=',')
dt = sklearn.tree.DecisionTreeClassifier(criterion = 'entropy')
dt = dt.fit(x_train,y_train)
dt_test_score = dt.score(x_test,y_test)
dt_train_score = dt.score(x_train,y_train)
print ("Accuracy in testing data set is: %.4f"%dt_test_score)
print ("Accuracy in training data set is: %.4f"%dt_train_score)
dot_data = sklearn.tree.export_graphviz(dt,out_file=None,filled = True, fea
graph = graphviz.Source(dot_data)
graph.render("massive_dataset")
graph
```

Out[27]:

(b) Discuss the results seen for the two trees

We can see from these results that the 4-feature model performed better on testing data than the 8-feature model. However, we see massive underfitting in the 8-feature model, as accuracy on the training data is 100% while on the testing data it is 19%. This problem may be because of the extra features and the representation of the data, as the 4-feature data is in a much simpler form.

In [ ]:

In [ ]:

In [ ]:

In [ ]: