Name and ID

Pengcheng Xu (pxu02)

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.

```
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.

```python
def counting_based_for_A():
    correct_count = 2 + 4 # left (i.e. o) + right (i.e. x) corrects in each
    total_count = 8
    return correct_count / total_count

def counting_based_for_B():
    correct_count = 3 + 3 # eft (i.e. o) + right (i.e. x) corrects in each
    total_count = 8
    return correct_count / total_count

res_map = {}
res_map['A'] = counting_based_for_A()
res_map['B'] = counting_based_for_B()
# sorted the result from best to worst based on accurate value
res_map = sorted(res_map.items(), key=lambda x: x[1], reverse=True)
# print out the result
for f_name, f_value in res_map:
    print(f'{f_name}: {f_value}')
```

```
A: 0.75
B: 0.75
```

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

```
import math

entropy_example = -(4/8 * math.log(4/8, 2) + 4/8 * math.log(4/8, 2))

entropy_remainder_A = -( 2 / 8 * 0 + 6 / 8 * ( -(2/6*math.log(2/6, 2) + 4/6

gain_A = entropy_example - entropy_remainder_A

entropy_remainder_B = -(4/8 * -(3/4 * math.log(3/4,2) + 1/4*math.log(1/4,2)

gain_B = entropy_example - entropy_remainder_B

res_info = {}
res_info["A"] = gain_A
res_info["B"] = gain_B
res_info = sorted(res_info.items(), key=lambda item: item[1], reverse = Tru
for f_name, f_value in res_info:
    print(f'{f_name}: {f_value:.3f}')
```

```
B: 1.811
A: 1.689
```

(c) Discussion of results.

> This means when we use counting-based heuristic, we're gonna use either feature A or B as our root node, since the couting-base criterioin is the same, we cannot differentiate between A and B.
>
> When we use information-based heuristic, we're gonna use B as our root classification node, since we would gain more infomation by using B compared to A.
>
> This shows the information-based heuristic is preferable, since it can help us choose a better one when counting-based heuristic is stuck.

2 Compute both heuristics for simplified abalone data.

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

```
simplified_x_train = pd.read_csv("data_abalone/small_binary_x_train.csv")
simplified_y_train = pd.read_csv("data_abalone/3class_y_train.csv")
```

```
num_data = len(simplified_y_train)
# print("# of data: ", num_data)

# couting_based heuristic for is_male
y_group_ismale_0 = simplified_y_train[simplified_x_train['is_male'] == 0]
y_group_ismale_1 = simplified_y_train[simplified_x_train['is_male'] == 1]

y_group_ismale_0_output_label = y_group_ismale_0.value_counts().index[0][0]
y_group_ismale_1_output_label = y_group_ismale_1.value_counts().index[0][0]

y_group_ismale_0_correct = np.sum(y_group_ismale_0 == y_group_ismale_0_outp
y_group_ismale_1_correct = np.sum(y_group_ismale_1 == y_group_ismale_1_outp

correct_rate_ismale = (y_group_ismale_0_correct + y_group_ismale_1_correct)

# couting_based heuristic for length_mm
y_group_lengthmm_0 = simplified_y_train[simplified_x_train['length_mm'] ==
y_group_lengthmm_1 = simplified_y_train[simplified_x_train['length_mm'] ==

y_group_lengthmm_0_output_label = y_group_lengthmm_0.value_counts().index[0
y_group_lengthmm_1_output_label = y_group_lengthmm_1.value_counts().index[0

y_group_lengthmm_0_correct = np.sum(y_group_lengthmm_0 == y_group_lengthmm_
y_group_lengthmm_1_correct = np.sum(y_group_lengthmm_1 == y_group_lengthmm_

correct_rate_lengthmm = (y_group_lengthmm_0_correct + y_group_lengthmm_1_cc
# couting_based heuristic for diam_mm
y_group_diammm_0 = simplified_y_train[simplified_x_train['diam_mm'] == 0]
y_group_diammm_1 = simplified_y_train[simplified_x_train['diam_mm'] == 1]

y_group_diammm_0_output_label = y_group_diammm_0.value_counts().index[0][0]
y_group_diammm_1_output_label = y_group_diammm_1.value_counts().index[0][0]

y_group_diammm_0_correct = np.sum(y_group_diammm_0 == y_group_diammm_0_outp
y_group_diammm_1_correct = np.sum(y_group_diammm_1 == y_group_diammm_1_outp

correct_rate_diammm = (y_group_diammm_0_correct + y_group_diammm_1_correct)
# couting_based heuristic for height_mm

y_group_heightmm_0 = simplified_y_train[simplified_x_train['height_mm'] ==
y_group_heightmm_1 = simplified_y_train[simplified_x_train['height_mm'] ==

y_group_heightmm_0_output_label = y_group_heightmm_0.value_counts().index[0
y_group_heightmm_1_output_label = y_group_heightmm_1.value_counts().index[0
```

```
y_group_heightmm_0_correct = np.sum(y_group_heightmm_0 == y_group_heightmm_
y_group_heightmm_1_correct = np.sum(y_group_heightmm_1 == y_group_heightmm_

correct_rate_heightmm = (y_group_heightmm_0_correct + y_group_heightmm_1_cc

# print res
res_q2 = {}
res_q2['is_male'] = correct_rate_ismale.to_numpy()[0]
res_q2['length_mm'] = correct_rate_lengthmm.to_numpy()[0]
res_q2['diam_mm'] = correct_rate_diammm.to_numpy()[0]
res_q2['height_mm'] = correct_rate_heightmm.to_numpy()[0]

# print(res_q2)
# sorted the result from best to worst based on accurate value
res_q2 = sorted(res_q2.items(), key=lambda x: x[1], reverse=True)
# print out the result
for f_name, f_value in res_q2:
    print(f'{f_name}: {f_value}')
```

```
height_mm: 0.7292191435768262
diam_mm: 0.7134760705289672
length_mm: 0.7021410579345088
is_male: 0.5869017632241813
```

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

```
# H(Example)
example_num_y0 = np.sum(simplified_y_train==0)
example_num_y1 = np.sum(simplified_y_train==1)
example_num_y2= np.sum(simplified_y_train==2)
H_Example = - (example_num_y0 / num_data * math.log(example_num_y0 / num_da

# information_based heuristic for is_male
num_y0_y_group_ismale_0 = np.sum( y_group_ismale_0 ==0 )
num_y1_y_group_ismale_0 = np.sum( y_group_ismale_0 ==1 )
H_ismale_0 = -(num_y0_y_group_ismale_0/len(y_group_ismale_0) * math.log(num

num_y0_y_group_ismale_1 = np.sum( y_group_ismale_1 ==0 )
num_y1_y_group_ismale_1 = np.sum( y_group_ismale_1 ==1 )
H_ismale_1 = -(num_y0_y_group_ismale_1/len(y_group_ismale_1) * math.log(num

H_ismale = len(y_group_ismale_0) / num_data * H_ismale_0 + len(y_group_ismc
Gain_ismale = H_Example - H_ismale
```

```
# information_based heuristic for length_mm
num_y0_y_group_lengthmm_0 = np.sum( y_group_lengthmm_0 ==0 )
num_y1_y_group_lengthmm_0 = np.sum( y_group_lengthmm_0 ==1 )
H_lengthmm_0 = -(num_y0_y_group_lengthmm_0/len(y_group_lengthmm_0) * math.l

num_y0_y_group_lengthmm_1 = np.sum( y_group_lengthmm_1 ==0 )
num_y1_y_group_lengthmm_1 = np.sum( y_group_lengthmm_1 ==1 )
H_lengthmm_1 = -(num_y0_y_group_lengthmm_1/len(y_group_lengthmm_1) * math.l

H_lengthmm = len(y_group_lengthmm_0) / num_data * H_lengthmm_0 + len(y_grou
Gain_lengthmm = H_Example - H_lengthmm

# couting_based heuristic for diam_mm
num_y0_y_group_diammm_0 = np.sum( y_group_diammm_0 ==0 )
num_y1_y_group_diammm_0 = np.sum( y_group_diammm_0 ==1 )
H_diammm_0 = -(num_y0_y_group_diammm_0/len(y_group_diammm_0) * math.log(num

num_y0_y_group_diammm_1 = np.sum( y_group_diammm_1 ==0 )
num_y1_y_group_diammm_1 = np.sum( y_group_diammm_1 ==1 )
H_diammm_1 = -(num_y0_y_group_diammm_1/len(y_group_diammm_1) * math.log(num

H_diammm = len(y_group_diammm_0) / num_data * H_diammm_0 + len(y_group_diam
Gain_diammm = H_Example - H_diammm

# information_based heuristic for height_mm
num_y0_y_group_heightmm_0 = np.sum( y_group_heightmm_0 ==0 )
num_y1_y_group_heightmm_0 = np.sum( y_group_heightmm_0 ==1 )
H_heightmm_0 = -(num_y0_y_group_heightmm_0/len(y_group_heightmm_0) * math.l

num_y0_y_group_heightmm_1 = np.sum( y_group_heightmm_1 ==0 )
num_y1_y_group_heightmm_1 = np.sum( y_group_heightmm_1 ==1 )
H_heightmm_1 = -(num_y0_y_group_heightmm_1/len(y_group_heightmm_1) * math.l

H_heightmm = len(y_group_heightmm_0) / num_data * H_heightmm_0 + len(y_grou
Gain_heightmm = H_Example - H_heightmm

res_q2b = {}
res_q2b['is_male'] = Gain_ismale.to_numpy()[0]
res_q2b['length_mm'] = Gain_lengthmm.to_numpy()[0]
res_q2b['diam_mm'] = Gain_diammm.to_numpy()[0]
res_q2b['height_mm'] = Gain_heightmm.to_numpy()[0]

# sorted the result from best to worst based on accurate value
res_q2b = sorted(res_q2b.items(), key=lambda x: x[1], reverse=True)
```

```
# print out the result
for f_name, f_value in res_q2b:
    print(f'{f_name}: {f_value:.3f}')
```

```
height_mm: 0.227
diam_mm: 0.205
length_mm: 0.191
is_male: 0.083
```

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

(a) Print accuracy values and generate tree images.

```
from sklearn import tree
# simplified data-set
simp_x_train = pd.read_csv("data_abalone/small_binary_x_train.csv")
simp_y_train = pd.read_csv("data_abalone/3class_y_train.csv")

simp_x_test = pd.read_csv("data_abalone/small_binary_x_test.csv")
simp_y_test = pd.read_csv("data_abalone/3class_y_test.csv")

clf = tree.DecisionTreeClassifier()
clf.fit(simp_x_train, simp_y_train)
simp_accurate_train = clf.score(simp_x_train, simp_y_train)
simp_accurate_test = clf.score(simp_x_test, simp_y_test)

print("simplified version - train accuracy: ", simp_accurate_train)
print("simplified version - test accuracy: ", simp_accurate_test)

dot_data = tree.export_graphviz(clf, out_file=None, feature_names=list(simp
graph = graphviz.Source(dot_data)
graph.render("simp_tree")
# full version
full_x_train = pd.read_csv("data_abalone/x_train.csv")
full_y_train = pd.read_csv("data_abalone/y_train.csv")

full_x_test = pd.read_csv("data_abalone/x_test.csv")
full_y_test = pd.read_csv("data_abalone/y_test.csv")

clf2 = tree.DecisionTreeClassifier()
clf2.fit(full_x_train, full_y_train)
full_accurate_train = clf2.score(full_x_train, full_y_train)
full_accurate_test = clf2.score(full_x_test, full_y_test)

print("full version - train accuracy: ",full_accurate_train)
print("full version - test accuracy: ",full_accurate_test)

dot_data2 = tree.export_graphviz(clf2, out_file=None, feature_names=list(fu
graph2 = graphviz.Source(dot_data2)
graph2.render("full_tree")
```

```
simplified version - train accuracy:  0.7326826196473551
simplified version - test accuracy:  0.722
full version - train accuracy:  1.0
full version - test accuracy:  0.196




'full_tree.pdf'
```

(b) Discuss the results seen for the two trees

- Discuss the results you have just seen. > The simplified version tree is relatively small and balanced, while the full version tree is huge and relatively unbalanced
- What do the various accuracy-score values tell you? > The the accuracy-score of the simplified version tells us that this model performs normal (i.e. training accuracy score is slightly better than testing score), while the accurarcy-score of the full version tells us that this model overfits the data-set, since it fits the training set perfect but screws up on the testing set.
- How do the two trees that are produced differ? > The depth of the simplified version tree is small and well-balanced, and at most of leaves, the gini value is not zero (i.e. all samples in that group don't have the same output). While the full version is really huge, although most of the leaves have the gini value zero, but also most of the leaves usually only include just one sample.
- Looking at the outputs (leaves) of the simplified-data tree, what sorts of errors does that tree make? > The errors it makes is that almost all of its leaves are not pure (i.e. gini value is not zero, or all samples in the leaf do not in the same class). Because ideally we expect all samples in the leaf would end up being in the same output class.