Name and ID

Tanay Nistala (1379385)

HW04 Code

You will complete the following notebook, as described in the PDF for Homework 04 (included in the download with the starter code). You will submit:

1. This notebook file (`hw04.ipynb`), `implementation.py`, and two files for both trees images, i.e., `full`, `full.pdf`, `simple`, and `simple.pdf` (PDFs and text files generated using `graphviz` within the code). HINT: `render()`, and it should be clear when to use it, i.e., #3). Compress all files mentioned and submit 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 [ ]:
```
import numpy as np
import pandas as pd

import sklearn.tree
import graphviz

from implementation import information_remainder, counting_heuristic, set_er

%load_ext autoreload
%autoreload 2
```

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.

In [ ]:
```
feature_names = np.array(["A", "B"])
feature_len = 2
```

```
classes = [0, 1]

x_set = np.array([[1, 1],
                  [1, 1],
                  [0, 1],
                  [0, 0],
                  [0, 1],
                  [0, 0],
                  [0, 0],
                  [0, 0]])
y_set = np.array([0, 0, 0, 0, 1, 1, 1, 1])
```

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

In [ ]:
```
# TODO counting_heuristic()

# TODO sort the feature names by their correct counts
# e.g., sort_correct = [best, second_best, ..., worst]
# e.g., sort_names = ["A", "B"] or ["B","A"]
sort_correct = [ counting_heuristic(x_set, y_set, i, classes) for i in range
sort_names = [ feature_names[i] for i in np.argsort(sort_correct) ]

# Print the sorted features along with their correct predictions count in th
longest = max(len(name) for name in sort_names)
for name, correct in zip(sort_names, sort_correct):
    print("%*s: %d/%d" % (longest, name, correct, len(x_set)))
```

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

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

In [ ]:
```
# TODO information_remainder()

# TODO sort the feature names by their gains
sort_gains = [ information_remainder(x_set, y_set, i, classes) for i in rang
sort_names_by_gains = [ feature_names[i] for i in np.argsort(sort_gains) ] #

longest = max(len(name) for name in sort_names_by_gains)
for name, gain in zip(sort_names_by_gains, sort_gains):
    print("%*s: %.3f" % (longest, name, gain))
```

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

(c) Discussion of results.

While the counting-based heuristic suggests that features A and B both result in similar accuracies over the dataset, the information-theoretic heuristic suggests that feature B, with an information gain of ~0.31, provides more information for the classification of data as compared to feature A with an information gain of ~0.19.

2 Compute both heuristics for simplified abalone data.

In [ ]:
```
# load the data into np arrays

# TODO:fix the empty lists below
# full-feature abalone data
x_train = pd.read_csv("data_abalone/x_train.csv")
x_test = pd.read_csv("data_abalone/x_test.csv")
y_train = pd.read_csv("data_abalone/y_train.csv")
y_test = pd.read_csv("data_abalone/y_test.csv")

# TODO:fix the empty lists below
# simplified version of the data (Restricted-feature)
simple_x_train = pd.read_csv("data_abalone/small_binary_x_train.csv")
simple_x_test = pd.read_csv("data_abalone/small_binary_x_test.csv")
simple_y_train = pd.read_csv("data_abalone/3class_y_train.csv")
simple_y_test = pd.read_csv("data_abalone/3class_y_test.csv")

# get useful information
# TODO:fix the empty lists below
full_feature_names = x_train.columns.to_numpy() # features names of full-fea
simple_feature_names = simple_x_train.columns.to_numpy() # features names of
classes_abalone = np.sort(simple_y_test['rings'].unique()) # unique set of c
class_names = ["Small", "Medium", "Large"] # name of the classes

print(full_feature_names)

simple_x_train = simple_x_train.to_numpy()
simple_x_test = simple_x_test.to_numpy()
simple_y_train = simple_y_train.to_numpy()
simple_y_test = simple_y_test.to_numpy()
```

```
['is_male' 'length_mm' 'diam_mm' 'height_mm' 'whole_weight_g'
 'shucked_weight_g' 'viscera_weight_g' 'shell_weight_g']
```

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

In [ ]:
```
# TODO counting_heuristic()

# TODO sort the feature names by their correct counts
sort_correct_abalone = [ counting_heuristic(simple_x_train, simple_y_train,
sort_names_abalone = [ simple_feature_names[i] for i in np.argsort(sort_corr

sort_correct_abalone.reverse()
sort_names_abalone.reverse()

# Print the sorted features along with their correct predictions count in th
longest = max(len(name) for name in sort_names_abalone)
for name, correct in zip(sort_names_abalone, sort_correct_abalone):
    print("%*s: %d/%d" % (longest, name, correct, len(simple_x_train)))
```

```
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 [ ]:
```python
# information_remainder()

# sort the feature names by their gains
sort_gains_abalone = [ information_remainder(simple_x_train, simple_y_train,
sort_names_by_gains_abalone = [ simple_feature_names[i] for i in np.argsort(

sort_gains_abalone.reverse()
sort_names_by_gains_abalone.reverse()

longest = max(len(name) for name in sort_names_by_gains_abalone)
for name, gain in zip(sort_names_by_gains_abalone, sort_gains_abalone):
    print("%*s: %.3f" % (longest, name, gain))
```

```
height_mm: 0.173
  diam_mm: 0.150
length_mm: 0.135
  is_male: 0.025
```

3) Generate decision trees (criterion='entropy', random_state=42) for full- and simple-feature data

(a) Train and eval on entire train and test sets. Print accuracy values and generate tree images.

Render the tree diagram, naming it "full." A text file and PDF should be created and saved (i.e., `full` and `full.pdf`) - include both in submission.

In [ ]:
```python
# calculate accuracies
classifier = sklearn.tree.DecisionTreeClassifier(criterion="entropy", random
classifier.fit(x_train, y_train)

train_accuracy = classifier.score(x_train, y_train)
test_accuracy = classifier.score(x_test, y_test)
print(f"Accuracy (train): {train_accuracy:.3f}")
print(f"Accuracy  (test): {test_accuracy:.3f}")

# generate tree image
dot_data = sklearn.tree.export_graphviz(classifier, out_file=None,
                                        feature_names=full_feature_names)
graphviz.Source(dot_data).render("full")
```

```
Accuracy (train): 1.000
Accuracy  (test): 0.202
```

Out[ ]:
```
'full.pdf'
```

(b) Restricted-feature (aka simple) data.

Train and eval on simple train and test sets. Same as above, accept this time use the `simple` set. Render the tree diagram, naming it "simple." A text file and PDF should be created and saved (i.e., `simple` and `simple.pdf`) - include both in submission.

```
In [ ]: # TODO calculate out accuracies
classifier_simple = sklearn.tree.DecisionTreeClassifier(criterion="entropy",
classifier_simple.fit(simple_x_train, simple_y_train)

simple_train_accuracy = classifier_simple.score(simple_x_train, simple_y_tra
simple_test_accuracy = classifier_simple.score(simple_x_test, simple_y_test)
print(f"Accuracy (train): {simple_train_accuracy:.3f}")
print(f"Accuracy  (test): {simple_test_accuracy:.3f}")

# TODO generate tree image
dot_data = sklearn.tree.export_graphviz(classifier_simple, out_file=None,
                                        feature_names=simple_feature_names,
                                        class_names=class_names)
graphviz.Source(dot_data).render("simple")
```

```
Accuracy (train): 0.733
Accuracy  (test): 0.722
Out[ ]: 'simple.pdf'
```

(c) Discuss the results seen for the two trees

The full tree is clearly much more complex than the simple tree, and appears to "overfit" to the data in a way as it achieves a perfect training accuracy but does abysmally on the test set. The simple tree appears to fit the data well since it achieves similar accuracies on the train and test sets, but it only outputs classes "Small" and "Medium" which suggests that the model is unable to model the data well enough to separate the large abalone from the rest.