hw05

April 25, 2021

0.1 Name and ID

Mattia Danese

0.2 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.

0.2.1 Import required libraries.

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

import sklearn.tree
import graphviz
```

0.3 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.

0.3.1 1 Compute both heuristics for toy data.

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

[6]:
```
def counting(x_data, y_data):
    importances = []
    num_features = np.shape(x_data)[1]
    i = 0
    while i < num_features:
        correct = 0
        j = 0
        while j < len(y_data):
```

```
            if(x_data[j][i] == y_data[j]):
                correct += 1
            j += 1
        i += 1
        importances.append(1 - (correct / len(y_data)))

    return importances
```

[7]:
```
toy_data_y = [0,0,0,0,1,1,1,1]
toy_data_x = [[1,1], [1,1], [0,1], [0,0],
              [0,1], [0,0], [0,0], [0,0]]

importances = counting(toy_data_x, toy_data_y)
print("A: %.3f" % importances[0])
print("B: %.3f" % importances[1])
```

```
A: 0.750
B: 0.750
```

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

[25]:
```
def calc_entropy(n):
    if n == 0:
        return 0
    else:
        return -1 * n * np.log2(n)
```

[31]:
```
def info(x_data, y_data, n_features, n_classes):
    gains = []

    # gets H(examples)
    e = 0
    for i in range(n_classes):
        total = 0

        for y in y_data:
            if y == i:
                total += 1
        total /= len(y_data)

        e += (calc_entropy(total))

    #gets remainder

    for f in range(n_features):

        #gets the amount of data points at each branch (2 branches per feature)
        branch_totals = [0,0]
```

```
        idx_1 = []
        idx_0 = []
        for i in range(len(y_data)):
            if x_data[i][f] == 1:
                branch_totals[1] += 1
                idx_1.append(i)
            else:
                branch_totals[0] += 1
                idx_0.append(i)

        #gets the amount of data points of each class are at each branch
        class_counts = [0 for x in range(n_classes)]

        for c in range(n_classes):
            for i in range(len(idx_0)):
                if y_data[i] == c:
                    class_counts[c] += 1

        #calculates the entropy of branch 0
        r0 = []
        for i in range(n_classes):
            r0.append(calc_entropy((class_counts[i] / branch_totals[0])))

        class_counts = [0 for x in range(n_classes)]

        for c in range(n_classes):
            for i in range(len(idx_1)):
                if y_data[i] == c:
                    class_counts[c] += 1

        #calculates the entropy of branch 1
        r1 = []
        for i in range(n_classes):
            r1.append(calc_entropy((class_counts[i] / branch_totals[1])))


        #calculates the complete remainder
        r = ((branch_totals[0] / len(y_data)) * sum(r0)) + ((branch_totals[1] /␣
↪len(y_data)) * sum(r1))

        gains.append(e-r)

    return gains
```

[36]:
```
# TODO
toy_data_y = [0,0,0,0,1,1,1,1]
toy_data_x = [[1,1], [1,1], [0,1], [0,0],
```

```
              [0,1], [0,0], [0,0], [0,0]]


importances = info(toy_data_x, toy_data_y, 2, 2)
print("B: %.3f" % importances[1])
print("A: %.3f" % importances[0])
```

```
B: 1.000
A: 0.311
```

(c) Discussion of results. For the counting based heuristic, both features have equal importance as they both correctly classify 3 out of 4 data points. The information-theoretic heuristic, however, shows that feature B is more important than feature A when it comes to classifying data points. For datasets of this size, the difference, in terms of trees produced, between using both heuristics is most likely negligible, that is there probably won't be a great difference in efficiency. For much larger datasets, however, using the information-theoretic heuristic will probably save a lot of time and be more efficient as it can already identify, on a very small dataset, that feature B is more important.

0.3.2 2 Compute both heuristics for simplified abalone data.

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

[33]:
```
# TODO

simp_train_x = np.loadtxt('./data_abalone/small_binary_x_train.csv',␣
 ↪delimiter=',', skiprows=1)
simp_train_y = np.loadtxt('./data_abalone/3class_y_train.csv', delimiter=',',␣
 ↪skiprows=1)

importances = counting(simp_train_x, simp_train_y)
print("height_mm: %.3f" % importances[0])
print("diam_mm:   %.3f" % importances[1])
print("is_male:   %.3f" % importances[2])
print("length_mm: %.3f" % importances[3])
```

```
height_mm: 0.413
diam_mm:   0.298
is_male:   0.287
length_mm: 0.271
```

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

[38]:
```
# TODO
importances = info(simp_train_x, simp_train_y, 4, 3)

print("height_mm: ", importances[2])
print("diam_mm:   ", importances[0])
print("is_male:   ", importances[1])
```

```
print("length_mm: ", importances[3])
```

```
height_mm:  -0.0005752180469202894
diam_mm:    -0.002175693090761932
is_male:    -0.0024105805899838906
length_mm:  -0.002431707127539484
```

0.3.3 3 Generate decision trees for full- and restricted-feature data

(a) Print accuracy values and generate tree images.

[50]:
```
# TODO
simp_test_x = np.loadtxt('./data_abalone/small_binary_x_test.csv',␣
 ↪delimiter=',', skiprows=1)
simp_test_y = np.loadtxt('./data_abalone/3class_y_test.csv', delimiter=',',␣
 ↪skiprows=1)

model = sklearn.tree.DecisionTreeClassifier(criterion = "entropy")
model.fit(simp_train_x, simp_train_y)

acc_simp_train = model.score(simp_train_x, simp_train_y)
acc_simp_test = model.score(simp_test_x, simp_test_y)

print("Train Accuracy on Simplified Data:", acc_simp_train)
print("Test Accuracy on Simplified Data:", acc_simp_test)
print(" ")

dot_data = sklearn.tree.export_graphviz(model, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("simplified_tree")

######################################################################

train_x = np.loadtxt('./data_abalone/x_train.csv', delimiter=',', skiprows=1)
train_y = np.loadtxt('./data_abalone/y_train.csv', delimiter=',', skiprows=1)
test_x = np.loadtxt('./data_abalone/x_test.csv', delimiter=',', skiprows=1)
test_y = np.loadtxt('./data_abalone/y_test.csv', delimiter=',', skiprows=1)

model2 = sklearn.tree.DecisionTreeClassifier(criterion = "entropy")
model2.fit(train_x, train_y)

acc_full_train = model2.score(train_x, train_y)
acc_full_test = model2.score(test_x, test_y)

print("Train Accuracy on Full Data:", acc_full_train)
print("Test Accuracy on Full Data:", acc_full_test)

dot_data = sklearn.tree.export_graphviz(model2, out_file=None)
```

```
graph = graphviz.Source(dot_data)
graph.render("full_tree")
```

```
Train Accuracy on Simplified Data: 0.7326826196473551
Test Accuracy on Simplified Data: 0.722

Train Accuracy on Full Data: 1.0
Test Accuracy on Full Data: 0.182
```

[50]: `'full_tree.pdf'`

(b) Discuss the results seen for the two trees The training are testing accuracies of the simplified data are 0.733 and 0.722, respectively, while those of the full data are 1.0 and 0.182. The training accuracy of the full data being 1.0 shows that this tree is extremely overfit and is very precise relative to its training data. This is also supported by the testing accuracy of the full data being very low at just 0.182 as this shows that the tree is very bad a generalizing over new data (which makes sense since it is very overfit). Due to the full dataset being much bigger than the simplified dataset and that the tree of the full data is very overfit, the trees produced vary greatly in size. The tree for the simplified data has a depth of 5 and 16 leaves while the tree for the full data is large enough to the point where I cannot count the leaves.

[ ]: