Assignment 3

Marius Aarsnes

March 2018

1 Introduction

In this exercise you will implement a decision tree learning algorithm. Please make sure to read the exercise completely before starting the implementation

2 Tasks

You should implement two different versions of $IMPORTANCE$ function (used by pseudo-code in Figure 18.5):

1. Allocating a random number as importance to each attribute.
2. Define Importance as the expected information gain, as discussed in the lecture.

To Compare the two versions of $IMPORTANCE$, you should examine each of them by doing the following steps:

1. Learn a decision tree from the data in *training.txt*
2. Document the tree you got in your report
3. Classify all examples in the test-set (given the text-file *test.txt*), and calculate the accuracy of the learner by comparing to the correct classification of the examples in the test-set

Discuss your findings:

1. What can you conclude about the results you have obtained? Which $IMPORTANCE$ is better, and why?
2. Do you get the same result if you run the random $IMPORTANCE$ several times?
3. What happens when you run the learned based on Information Gain several times?

3 Results

When making a solution for this assignment i found that making some small adjustments made an impact on the actual result of the classification. The change was if splitting attributes were allowed to appear multiple times in the tree or not.

The Example in the book implies that this is not a possibility with the example used. However, after a quick Google search (Stack Overflow) it seems like it is possible/allowed to do this as long as they do not appear multiple times in the same branch (i.e. a splitting attribute cannot be descendent of itself).

Therefore, the *DecisionTreeLearning* Class in the source code has a bool *allow_reuse_atr* so that this can be toggled on and off.

```
subtree = self._decision_tree_learning(exs, list(attributes),
                                       examples)
```

(a) Building a subtree for a node, allowing for reuse of attributes.

```
subtree = self._decision_tree_learning(exs, attributes,
                                       examples)
```

(b) Building a subtree for a node, ***not*** allowing for reuse of attributes.

Figure 1: Two code snippets showing difference when allowing and not allowing the reuse of splitting-attributes

Finally, note that the attributes are null indexed, i.e. they range from 0 to 6, and not 1 to 7. Also, the possible values for each attribute are either 0 or 1, not 1 or 2. I made this change because it makes it easier to enumerate the attributes and checking for the possibility of classification, simply adding all 1s instead of counting both 1s and 2s.

The source code can be found in the accompanying zip-folder.

3.1 Random Importance

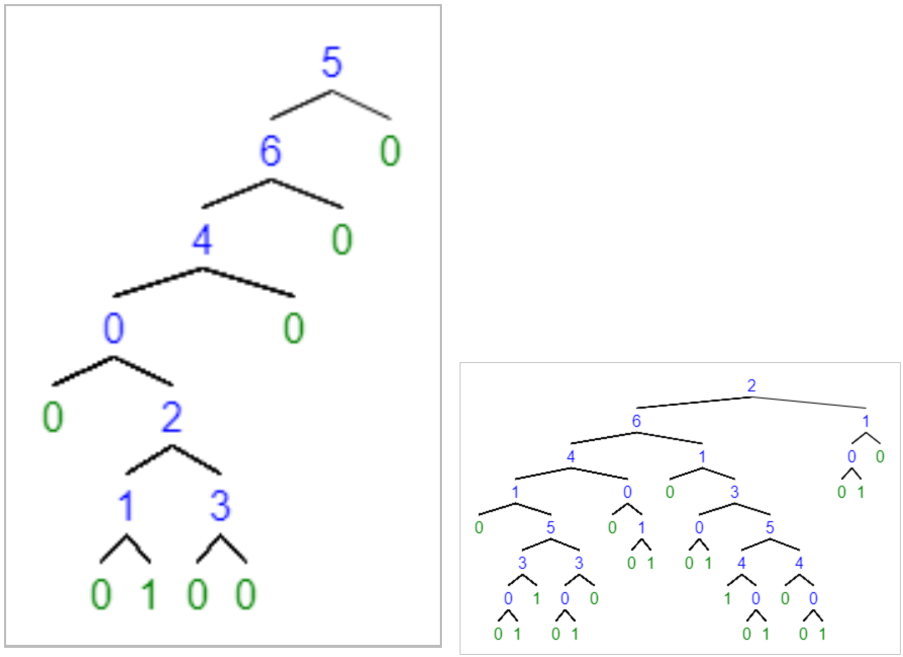
Figure 2 shows two decision trees using random importance. Figure 2a shows a tree without the reuse of splitting-attributes, while Figure 2b shows a tree where the reuse of splitting-attributes has been used.

Figure 3 shows two sets of results from testing random trees, training them for each time. Figure 3a shows scores when not reusing splitting-attributes, while Figure 3b shows scores when reusing splitting-attributes.

Generally, the use of random importance actually makes a decent resulting tree. The lowest score i have been able to get using random importance is 17/28, this goes for both with and without reusing attributes. Using random importance without the reuse of splitting attributes produces pretty stable results, most often between 21-23/28, peaking at 25/28. When allowing for the reuse of

splitting-attributes the results vary a bit more, but it is actually able to get a perfect score, 28/28.

Note: The trees and the results are not necessarily related, they are just examples of running the code. Using random importance creates different trees each time and therefore different results, i.e. this method is not deterministic.



(a) Decision tree using random importance, without reusing splitting attributes.

(b) Decision tree using random importance, with reusing splitting attributes.

Figure 2: Two trees made using random importance

```
Random score: 23.000000
Random score: 25.000000
Random score: 21.000000
Random score: 24.000000
Random score: 21.000000
```

(a) Results using random importance, without reusing splitting attributes.

```
Random score: 17.000000
Random score: 26.000000
Random score: 23.000000
Random score: 28.000000
Random score: 23.000000
```

(b) Results using random importance, with reusing splitting attributes.

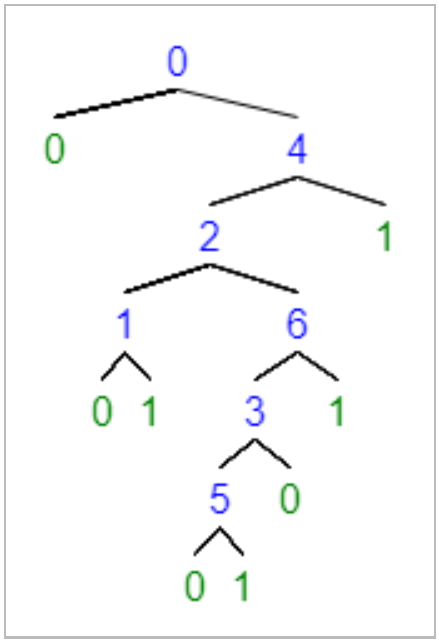
Figure 3: Two result-sets with random importance

3.2 Information Gain Importance

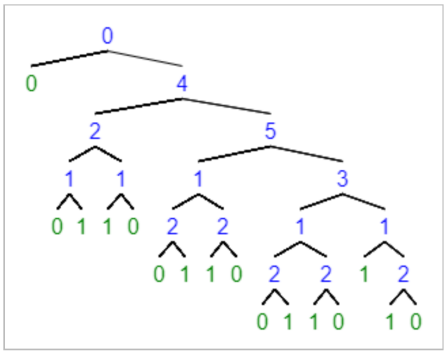
Figure 4 shows two decision trees using random importance. Figure 4a shows a tree without the reuse of splitting-attributes, while Figure 4b shows a tree where the reuse of splitting-attributes has been used.

Figure 5 shows two sets of results from testing random trees, training them for each time. Figure 5a shows scores when not reusing splitting-attributes, while Figure 5b shows scores when reusing splitting-attributes.

Using information gain, either with or without the reuse of splitting-attributes gives the same result each time, 20/28 and 26/28 for each of the forms, no matter how many times you train.



(a) Decision tree using information gain importance, without reusing splitting attributes.



(b) Decision tree using information importance, with reusing splitting attributes.

Figure 4: Two trees made using information gain importance

```
Information gain score: 20
Information gain score: 20
Information gain score: 20
Information gain score: 20
Information gain score: 20
```

(a) Results using information gain importance, without reusing splitting attributes.

```
Information gain score: 26
Information gain score: 26
Information gain score: 26
Information gain score: 26
Information gain score: 26
```

(b) Results using information gain importance, with reusing splitting attributes.

Figure 5: Two result-sets with information gain importance

3.3 Information Gain VS Random

No reuse of splitting-attributes: It is a little hard to make a decision in this case. When using information gain the result is locked as 20/28, while with random it ranges from 17/28 to 25/28. Even with this variation, the random importance generally is very stable and more often than not produces results slightly better than information gain. It may seem counter intuitive that picking random attributes is better than using a heuristic, but in this particular case i would argue that it is.

With reuse of splitting-attributes: In this case we see a bigger variation in the results from the random importance. Also, the results from using information gain is much better. Therefore, when allowing for splitting-attributes to appear multiple times in the decision tree, i would argue that the information gain approach is better than using random importance.

3.4 Multiple Runs

As mentioned with random importance you get a different tree and different score each time. At least more or less - different trees may sometimes generate the same score.

Using information gain on the other hand generates the same decision tree and gets the same score each time. This makes sense seeing as the information gain approach uses a heuristic on entropy to decide which attribute to split on. It is a deterministic approach, whereas picking random is not.