TDT4171 Artificial Intelligence Methods
Exercise 4
Decision Tree Learning Algorithms

Stian Hvatum (hvatum)
MTDT

March 12, 2012

Contents

1 Intro

In this assignment I have implemented the Decision Tree Learner Algorithm from AIMA Figure 18.5. I have chosen to implement the algorithm in C, because we are learning C in different cources at school right now, and C is a good and efficient language to implement this kind of algorithms, with good support for math and data structures.

2 How to build the program

The program is provided with builds for Windows, Linux and OSX. Binaries are located in the `Releases/`*`system`*-folder, and named `tree-learning`, where system is the provided platforms. To compile the source files for a different platform than provided, you can run `make clean && make`. You will then have a fresh binary file in the `bin`-directory. I have tested the code on `login.stud.ntnu.no`, `asti.idi.ntnu.no`, a Windows 7 machine with MSYS and MinGW, and with current version of X-Code on Max OSX Lion. All with GCC, but the code will probably compile with Visual Studio as well.

3 How to run the program

The program is run from the command line. You can supply input files, test data, which algorithm (Random or Information Gain) and iterations. How to supply each of the arguments are given by the program help file:

```
Help for Decision Tree Learner
         bin/tree-learning.exe [ARGUMENTS] INPUT FILES

Arguments:       Short form:     Desc:
--iterations n   -i n             n is the number of times the input files are read.
--random         -r               If set, the random-gain is used instead
                                  of the information gain algorithm
--test file      -t file          file is the a file to check our tree against
--help           -h               Displays this help text and exits

Example:
         bin/tree-learning.exe -r -i 100 -t test.txt training.txt training2.txt
```

4 Results

4.1 Resulting tree from Information Gain

This is the resulting tree gained from the training set. Each *Val* is an edge, each node is either a *Class* or an *Attribute*, where *Class*es are leaf nodes. This tree is correct on 26 of the 28 examples in the test-set. The tree matches all examples in its own training set.

```
Attrib:1
 | Val:1 Class:1
 | Val:2 Attrib:5
 |   | Val:1 Attrib:3
 |   |   | Val:1 Attrib:2
 |   |   |   | Val:1 Class:1
 |   |   |   | Val:2 Class:2
 |   |   | Val:2 Attrib:2
 |   |   |   | Val:1 Class:2
 |   |   |   | Val:2 Class:1
 |   | Val:2 Attrib:6
 |   |   | Val:1 Attrib:2
 |   |   |   | Val:1 Attrib:3
 |   |   |   |   | Val:1 Class:1
 |   |   |   |   | Val:2 Class:2
 |   |   |   | Val:2 Attrib:3
 |   |   |   |   | Val:1 Class:2
 |   |   |   |   | Val:2 Class:1
 |   |   | Val:2 Attrib:4
 |   |   |   | Val:1 Attrib:2
 |   |   |   |   | Val:1 Attrib:3
 |   |   |   |   |   | Val:1 Class:1
 |   |   |   |   |   | Val:2 Class:2
 |   |   |   |   | Val:2 Attrib:3
 |   |   |   |   |   | Val:1 Class:2
 |   |   |   |   |   | Val:2 Class:1
 |   |   |   | Val:2 Attrib:2
 |   |   |   |   | Val:1 Class:2
 |   |   |   |   | Val:2 Attrib:3
 |   |   |   |   |   | Val:1 Class:2
 |   |   |   |   |   | Val:2 Class:1
```

4.2 Resulting tree from Random

Here we see one possible tree from the Random algorithm. There Random trees are usually very long ones, so I ran the program a few time to get one that is short enough to fit on one page. This tree is correct on 21 of the 28 examples in the test-set, but this is about arbitary for each random tree. Though every Random tree is 100% match on its own training set, just as the Information Gain trees.

```
Attrib:1
 | Val:1 Class:1
 | Val:2 Attrib:4
 |   | Val:1 Attrib:5
 |   |   | Val:1 Attrib:2
 |   |   |   | Val:1 Attrib:3
 |   |   |   |   | Val:1 Class:1
 |   |   |   |   | Val:2 Class:2
 |   |   |   | Val:2 Attrib:3
 |   |   |   |   | Val:1 Class:2
 |   |   |   |   | Val:2 Class:1
 |   |   | Val:2 Attrib:7
 |   |   |   | Val:1 Attrib:2
 |   |   |   |   | Val:1 Class:1
 |   |   |   |   | Val:2 Attrib:3
 |   |   |   |   |   | Val:1 Class:2
 |   |   |   |   |   | Val:2 Class:1
 |   |   |   | Val:2 Attrib:6
 |   |   |   |   | Val:1 Attrib:2
 |   |   |   |   |   | Val:1 Attrib:3
 |   |   |   |   |   |   | Val:1 Class:1
 |   |   |   |   |   |   | Val:2 Class:2
 |   |   |   |   |   | Val:2 Attrib:3
 |   |   |   |   |   |   | Val:1 Class:2
 |   |   |   |   |   |   | Val:2 Class:1
 |   |   |   |   | Val:2 Attrib:3
 |   |   |   |   |   | Val:1 Attrib:2
 |   |   |   |   |   |   | Val:1 Class:1
 |   |   |   |   |   |   | Val:2 Class:2
 |   |   |   |   |   | Val:2 Class:2
 |   |   | Val:2 Attrib:5
 |   |   |   | Val:1 Attrib:6
 |   |   |   |   | Val:1 Attrib:3
 |   |   |   |   |   | Val:1 Attrib:7
 |   |   |   |   |   |   | Val:1 Attrib:2
 |   |   |   |   |   |   |   | Val:1 Class:1
 |   |   |   |   |   |   |   | Val:2 Class:2
 |   |   |   |   |   |   | Val:2 Attrib:2
 |   |   |   |   |   |   |   | Val:1 Class:1
 |   |   |   |   |   |   |   | Val:2 Class:2
 |   |   |   |   |   | Val:2 Attrib:2
 |   |   |   |   |   |   | Val:1 Class:2
 |   |   |   |   |   |   | Val:2 Class:1
 |   |   |   |   | Val:2 Attrib:7
 |   |   |   |   |   | Val:1 Class:1
 |   |   |   |   |   | Val:2 Attrib:2
 |   |   |   |   |   |   | Val:1 Attrib:3
 |   |   |   |   |   |   |   | Val:1 Class:1
 |   |   |   |   |   |   |   | Val:2 Class:2
 |   |   |   |   |   |   | Val:2 Class:1
 |   |   |   | Val:2 Attrib:3
 |   |   |   |   | Val:1 Attrib:2
 |   |   |   |   |   | Val:1 Class:1
 |   |   |   |   |   | Val:2 Class:2
 |   |   |   |   | Val:2 Attrib:6
 |   |   |   |   |   | Val:1 Attrib:2
 |   |   |   |   |   |   | Val:1 Class:2
 |   |   |   |   |   |   | Val:2 Class:1
 |   |   |   |   |   | Val:2 Attrib:7
 |   |   |   |   |   |   | Val:1 Attrib:2
 |   |   |   |   |   |   |   | Val:1 Class:2
 |   |   |   |   |   |   |   | Val:2 Class:1
 |   |   |   |   |   |   | Val:2 Class:2
```

4.3 Random vs. Information Gain

The Random trees also have "random" accuracy, but they are generally less accurate than those created by the Information Gain algorithm. Occationally the Random-created ones yields

a better accuracy, but this is due to the fact that Information Gain uses a heuristic on entropy to calculate which attributes are the most important ones, and a random selection of attributes might a seldom time perform better at finding the right ones.

I will state that the Information Gain algorithm is the better one, as it is usually generates better learning trees than the Random algorithm.

4.4 Multiple runs with Random

If I run Random multiple times with same input, I will get a different result as good as every time. The Random algorithm is seeded with current time as the program starts, and will choose a different `most-important`-attributes on each run where seconds on the host computer clock has changed[1] since last run.

4.5 Multiple runs with Information Gain

If I multiply the input for the Information Gain algorithm, I still end up with the exact same output. This is due to the fact that there are no more unique examples, and the new equal ones are eaten in chunks by the second check in the algorithm: `if` *`all examples`* `have same class then return the class`. The examples are in that way treated as a mathematical set rather than a collection. If we increase the training set with new unique examples rather than reusing the old ones, we would gain better results. Eg. if we concatenate the training set and the test set, we will get another learning tree which is 100% accurate on both those sets.

5 Small Code Guide

The program is split in several `.c`-files, each representing a part of the program. Most of the math stuff is done in `importance.c`. Here you will find the two implementations, Information Gain and Random. The **B**-function is also found here, named `func_b`. There is also som algorithm-dependent math in `plurality_value.c`, where we simply find which class is most representeted in a set of examples.

The main algorithm for building the tree is found in `decision_tree_learning.c`, and contains a lot of help-functions to build the tree. The main function is `decision_tree_learning`, which except for some C-specific malloc's and free's are a pure copy of Fig 18.5 from AIMA.

Code found in `types.c` is mainly data structures to build the examples and the trees, while the code in `main.c` is the reading of input files, initializing stacks and tree-pointers and cleanup upon completion.

[1] This is implementation dependent. A real Random-gain would be different on each run