TDT4171 Ex4

Harald Husum

April 2016

Code

The code appended to this report is not pretty, and full of magic numbers. But it works for the purpose specified, as long as you have Python 3, and host the data files in the same folder.

Entropy Importance

My Importance subroutine based on information gain is deterministic. Given the same training set, it will produce the same tree every time. This means we will see the same performance on the test set every time as well. In my case 7.143%, or 2/28 errors. See Appendix A for tree structure.

Random Importance

The random Importance subroutine yields wildly varying decision trees on the same data set. This is to be expected, as the attributes chosen will be random.

The same can be said for the results these trees give when classifying the test set. When building the tree 100'000 times, i get an average error of about 22% but highs of 57% and lows of 0%, on the test set.

See Appendix B for typical tree structure.

Conclusion

Even though it happens that the randomized importance function handles the test data perfectly, on average it is significantly worse than an implementation based on information gain. In addition, the latter method creates significantly more compact trees almost all the time. It is safe to say the information gain approach performs better.

Appendix A: Gain Based Tree Structure

```
NODE('test'): 0
  NODE('1')
  NODE('test'): 4
    NODE('test'): 2
      NODE('test'): 1
        NODE('1')
        NODE('2')
      NODE('test'): 1
        NODE('2')
        NODE('1')
    NODE('test'): 5
      NODE('test'): 1
        NODE('test'): 2
          NODE('1')
          NODE('2')
        NODE('test'): 2
          NODE('2')
          NODE('1')
      NODE('test'): 3
        NODE('test'): 1
          NODE('test'): 2
            NODE('1')
            NODE('2')
          NODE('test'): 2
            NODE('2')
            NODE('1')
        NODE('test'): 1
          NODE('2')
          NODE('test'): 2
            NODE('2')
            NODE('1')
```

Appendix B: Randomized Tree Structure

```
NODE('test'): 4
  NODE('test'): 2
    NODE('test'): 3
      NODE('test'): 1
        NODE('1')
        NODE('test'): 0
          NODE('1')
          NODE('2')
      NODE('test'): 1
        NODE('1')
        NODE('test'): 0
          NODE('1')
          NODE('2')
    NODE('test'): 3
      NODE('test'): 6
        NODE('test'): 0
          NODE('1')
          NODE('test'): 5
            NODE('1')
            NODE('2')
        NODE('test'): 0
          NODE('1')
          NODE('test'): 5
            NODE('test'): 1
              NODE('2')
              NODE('1')
            NODE('test'): 1
              NODE('2')
              NODE('1')
      NODE('test'): 1
        NODE('test'): 6
          NODE('test'): 0
            NODE('1')
            NODE('2')
          NODE('test'): 5
            NODE('2')
            NODE('test'): 0
              NODE('1')
              NODE('2')
        NODE('1')
  NODE('test'): 1
    NODE('test'): 6
      NODE('test'): 3
        NODE('1')
        NODE('test'): 5
          NODE('test'): 2
            NODE('1')
            NODE('test'): 0
              NODE('1')
              NODE('2')
          NODE('test'): 0
            NODE('1')
            NODE('2')
      NODE('test'): 2
        NODE('1')
        NODE('test'): 0
          NODE('1')
          NODE('2')
    NODE('test'): 6
      NODE('test'): 0
        NODE('1')
        NODE('test'): 3
          NODE('test'): 5
            NODE('2')
            NODE('test'): 2
              NODE('2')
              NODE('1')
          NODE('test'): 2
            NODE('2')
            NODE('1')
      NODE('test'): 0
        NODE('1')
        NODE('test'): 3
          NODE('test'): 5
            NODE('test'): 2
              NODE('2')
              NODE('1')
            NODE('2')
          NODE('test'): 5
            NODE('test'): 2
              NODE('2')
              NODE('1')
            NODE('2')
```