**README – ID3.java**

This is the documentation for the ID3 library in the ml package. It contains details about the ID3 implementation, a list of the publicly available functions that can be used and some test details I conducted for various datasets (using this library).

The Iterative Dichotomiser 3 Algorithm (ID3) is used in supervised machine learning problems. It operates by constructing a Decision Tree – which essentially splits the entire range of test cases into sub-cases depending on the value of certain attributes.

The next few paragraphs provide details of my implementation of the ID3 algorithm.

Node Definition

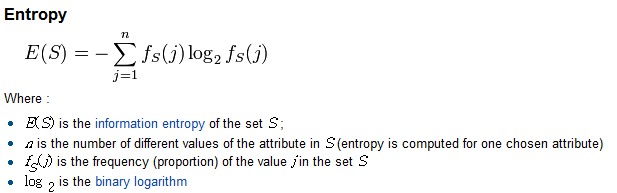
Essentially, each node in the decision tree contains the following details:

* A list of training test cases.
* The best attribute to split this node upon (if not leaf Node).
* Whether it is a terminal or leaf Node: i.e. all testcases of this node have the same output class label.
* The majority output class label – i.e. the output class label that occurs most frequently in the test cases of the node.

Best-Split Attribute

As mentioned earlier, each node contains an attribute that is the best way to split the testcases contained in the node. This is determined using the concept of **Information Gain**

First, the Entropy of the testcases with respect to the output class label is computed. Entropy is defined as follows (diagram taken from Wikipedia) -



**Diagram showing the Entropy formula – taken from Wikipedia**

After calculating the entropy of the node’s test cases, we compute the entropy for each split instance from these test cases (for a particular split attribute). The difference between the entropy of the split test cases and the original test cases gives the information gain. A split that results in the greatest information gain is most desirable.

Pruning:

I have implemented two types of pruning – Pre-Pruning and Post-Pruning. You can decide which one to use (or even if you want to use both together) based on running empirical tests on your dataset.

a. Pre-Pruning is implemented while the tree is being constructed (and hence the name). Essentially it uses the Chi-squared test (with significance value = 0.05) to check if a split (even though has the highest information gain) is significant. If yes, proceed with the algorithm. If not, then that node is not split further and it is labeled with the majority output label of its test cases.

b. Pessimistic Post-pruning is used after the tree is constructed. It traverses through each node of the tree and checks if the pessimistic error rate after the split is greater than the pessimistic error rate before the split. If so, the split is not statistically warranted and hence that node’s subtree is pruned.

I have tried to study pruning’s (both methods) effects on various datasets (which are reported in the Results section).

Handling Continuous Variables:

I have implemented a method that can split the continuous variables into any number n (where n > 0 and n <= t/2; where t = # of test cases – The t/2 restriction is applied because if not – then some bins will have single values, which is pretty much discrete). Essentially it splits the variables into n equal sized bins, depending on the attribute value. In the results, I have tried to see if this split value plays a role in accuracy.

Missing attributes:

For Information gain calculation purposes, I do not consider missing attribute values. However for splitting purposes, I assign the missing attributes the value of the maximum occurring attribute value.

Outputting the tree:

I have used Java’s AWT (Abstract Windowing Toolkit) to draw the tree to an output .ps file. In a later section I have provided some images output by the ID3 library.

**Library Functions Appendix**

**Only the public functions are listed here.**

**1. public** **ID3(**String fName**,** **int** numAttributes**,** **int** testCases**,** String delimiter**,** **int** limitSplits**) - Constructor**

throws: IOException**,** FileNotFoundException

Arguments:

String fName: File containing the training data.

int numAttributes: number of attributes in the training data (not counting the output)

int testCases: number of test cases to be considered from the training file.

String delimiter: delimiter between the attributes in a test case.

int limitSplits: number of splits for continuous valued attributes.

Returns:

An Instance of the ID3 class.

**2. public** **float** **findAccuracy(**String fTestName**,** **int** validationSize**,** String delim**)**

throws: IOException**,** FileNotFoundException

Arguments:  
String fTestName: name of the file containing the test data.

Int validationSize: number of test data instances.

Sting delim: delimiter to seperate attributes in a single instance of the test data.

Returns:

A float number representing the accuracy of the training dataset for this test data set.

Accuracy = # of test cases where output correctly predicted / # of test cases.

**3. public** **int** **returnNumNodes()**

Arguments:

None

Returns:

number of nodes in the decision tree.

4. **public** **void** **drawDecisionTree(**String outFile**,** **int** NodeSize**)**

Throws:

IOException, FileNotFoundException

Parameters:

String outFile: Name of the output file. It draws the tree to outFile.ps

int NodeSize: Diameter of a single Node in pixels.

Returns:

Nothing.

5. **public** String **getOutput(**String str**)**

Parameters;

String str: A single line representing a test case, without the output class label.

Returns

The Output class label.

6. **public** **void** **postPruneTree()**

Parameters: None

Returns: None.

Used to post-prune an already constructed tree.

7. **public** **void** **createDecisionTree(boolean** isPrePruning**)**

Parameters:

boolean isPrePruning: Set to true if pre-pruning required. Else set to false.

Returns: None.

**RESULTS**

All accuracy values are reported (with 95% confidence interval) to be in the range provided. (measured in %)

The Rows are the values for: no pruning, pre-pruning and post-pruning respectively.

The Columns are the values for the number of splits for continuous variables: splitValue = 2,3,5 and 10.

Each Dataset first contains a table for accuracy, and then a table for average number of nodes.

1. **Iris Dataset (**[**http://archive.ics.uci.edu/ml/datasets/Iris**](http://archive.ics.uci.edu/ml/datasets/Iris)**)**

Note: For this dataset, I did not exactly perform 10-cross validation. This is because, if I perform 10-cross validation, the size of the validation dataset will be around 15 (<30). So binomial distribution would not really apply to this dataset.

Hence I used a 30-size validation set, and did 5-cross validation. Then I randomly changed the dataset order and again performed a 5-cross validation.

**Accuracy table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **splitValue = 2** | **splitValue = 3** | **splitValue = 5** | **splitValue = 10** |
| **No pruning** | **64.444 +- 9.392** | **61.556 +- 9.686** | **64.00 +- 9.440** | **62.222 +- 9.662** |
| **Pre-pruning** | **27.556 +- 8.279** | **57.778 +- 9.920** | **64.00 +- 9.044** | **60.444 +- 9.762** |
| **Post-pruning** | **67.556 +- 8.992** | **64.00 +- 9.468** | **63.55 +- 9.521** | **60.00 +- 9.801** |

**Average Number of nodes in the tree** (For split value = 3**)**

|  |  |
| --- | --- |
| **No pruning** | **11.0** |
| **Pre-pruning** | **10.9** |
| **Post-pruning** | **8.2** |

1. **Congressional Voting Records DataSet (**[**http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records**](http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records)**)**

Here there is no need to evaluate across multiple split values, since all of the attributes are discrete.

**Accuracy table**

|  |  |
| --- | --- |
| **No pruning** | **79.4 +- 6.802** |
| **Pre-pruning** | **79.622 +- 6.751** |
| **Post-pruning** | **81.278 +- 6.34** |

**Average Number of Nodes in the Tree**

|  |  |
| --- | --- |
| **No pruning** | **26.1** |
| **Pre-pruning** | **1.0** |
| **Post-pruning** | **1.0** |

1. **Mushroom DataSet(**[**http://archive.ics.uci.edu/ml/datasets/Mushroom**](http://archive.ics.uci.edu/ml/datasets/Mushroom)**)**

Again, here there is no need to have split size, since all attributes are discrete.

**Accuracy table**

|  |  |
| --- | --- |
| **No pruning** | **47.893 +- 10.37** |
| **Pre-pruning** | **47.689+- 10.371** |
| **Post-pruning** | **47.7 +- 10.368** |

**Average number of nodes in the tree**

|  |  |
| --- | --- |
| **No pruning** | **46.9** |
| **Pre-pruning** | **46.8** |
| **Post-pruning** | **25.4** |

1. **Heart Disease dataset(**[**http://archive.ics.uci.edu/ml/datasets/Heart+Disease**](http://archive.ics.uci.edu/ml/datasets/Heart+Disease)**)**

Combining the processed VA, Hungarian, Switzerland and Cleveland datasets – to form a single dataset of heart diseases from the four hospitals.(total 921 test cases).

**Accuracy table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **splitValue = 2** | **splitValue = 3** | **splitValue = 5** | **splitValue = 10** |
| **No pruning** | **44.72 +- 10.253** | **42.867 +- 10.157** | **43.744 +- 10.203** | **46.477 +- 10.326** |
| **Pre-pruning** | **44.3944 +- 10.212** | **45.3111 +- 10.284** | **42.256 +- 10.114** | **45.611 +- 10.27** |
| **Post-pruning** | **43.811 +- 10.214** | **43.6556 +- 10.208** | **45.589 +- 10.297** | **47.6222 +- 10.357** |

**Average Number of nodes in the tree** (For split value = 3)

|  |  |
| --- | --- |
| **No pruning** | **32.1** |
| **Pre-pruning** | **30.7** |
| **Post-pruning** | **5.5** |

.

**CONCLUSIONS:**

1. For large datasets – pruning definitely helps in significantly reducing the size of the decision tree. Also, this does not compromise the accuracy of the trees, only those subtrees that are statistically insignificant are pruned.
2. Post-Pruning is definitely a better option compared to pre-pruning. The former helps to reduce the size of the tree better and also provides an equally good accuracy rate. However pre-pruning has the advantage over post-pruning of being computationally less complex.
3. In certain cases like the Congressional Voting dataset, it can be observed that pre-pruning itself reduces the size of the tree to 1. Hence it might be a good strategy to first try pre-pruning the database and then go to post-pruning if necessary. In the datasets I tried, only voting was extremely favorable to pre-pruning. In the other cases, pre-pruning did not fare significantly better than no-pruning.
4. Performance of a dataset (in terms of accuracy) on the basis of “split value”, is highly dependent on the problem domain. For instance, in many cases a great difference was not noticeable for different “split values”

However pre-pruning with a split value of 2 (for Iris database) performed very badly and gave only an accuracy of 27.55%. It is probable that this happened because the small split value did not segregate the test cases too well to get a good information gain.

1. So the best strategy would be to initially perform pre-pruning and see if the size of the tree reduces below a threshold. If it does, there is no need to perform post-pruning (as it is computationally complex). However the added computational complexity will be justified if pre-pruning does not reduce the size below a threshold. In that case we perform post-pruning.

The “split value” is a parameter that would be hard to ascertain – and empirical tests might be required to determine a good split value. However from the 4 datasets above, it does not seem to be a very significant factor in accuracy.

**Output Produced by the Program**

The following diagrams show the normal decision tree (no pruning) and the post-pruned tree produced by the program for the Iris data set.

