Decision Tree Learning.

Group Members:

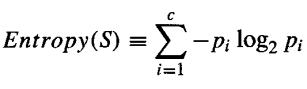
Bharat Raghunathan (2015AAPS263H)

V Thejas (2015A7PS142H)

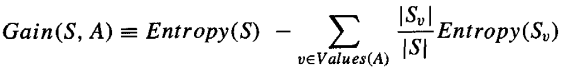
Nitish Ravishankar (2015A7PS152H)

Introduction:

The Decision Tree is initially built using the ID3 (Iterative Dichotomiser 3) algorithm which depends on a single statistical measure of ENTROPY or INFORMATION GAIN (of each attribute) given by



where c is the no. of possible values.



Entropy for one attribute with respect to another attribute.

The attribute with Minimum Entropy maximizes Information Gain, and hence, the attribute with maximum entropy is placed at the root of the Decision Tree.

Subsequently, the attributes with higher information gain(s) and lower entropies are placed closer to the root of the tree.

ID3 does not guarantee an optimal solution (may get stuck in local optima) because it follows a greedy approach and does not backtrack. Due to this, ID3 has the tendency to overfit the training data. The attributes with higher information gain are preferentially selected and there is an Inductive Bias (not easily evident) that follows.

**Approximate Inductive Bias of Decision Trees**: Shorter Trees are preferred over longer trees and trees that place high information gain attributes closer to the root are preferred over those that do not.

This directly follows from a principle:

**Ockham’s Razor:** Prefer Shorter Hypotheses over Longer ones.

ID3 exhibits what is called a Preferential Bias, wherein it searches all possible hypotheses but prefers one hypotheses over another with respect to a statistical measure.

In order to reduce overfitting tendencies, we have tried to do reduced-error pruning, which tries to prune/remove subtrees and replace them with leaf node(s) by backtracking till there is no reduction in the prediction accuracy (accuracy either improves or stays same).

After this, we have also tried to implement Random Forests, which builds an entire tree of possible decision trees (represented by a vector of decision trees) fitting a particular training example or an ensemble of decision trees and then takes a majority vote to classify a given test instance as positive or negative and we have also measured accuracy, precision, recall and F-Measure for each of these to evaluate the performance of each of these algorithms’ implementations.

Handling continuous valued attributes:

A split value ‘x’ is chosen for the continuous valued attribute, and the data pertaining to the node is divided into two, with <=x and >x values for the attribute. A std::pair<int,bool> with first as the attribute value and using value of the target for the data in the bool part is constructed, and sorted in increasing order of first value. At each index where there is a change in second value(target), information gain with x=(pair[i-1].first+pair[i].first)/2 is calculated and the x giving maximum information gain is selected to be the split value for the node.

Function used: splitContinuous(Treenode\* node,int ano, pair<int,int> &missing).

Time complexity: O(nlogn) where n is the size of node->data.

Handling missing values in training data:

For handling the “?”s in data, we considered two approaches.

1. **Assign values before constructing the decision tree:** We initially considered only those instances of training data, that were complete without a single “?”, and pushing the others into a waiting queue. Data missing in continuous valued attributes in waiting queue were set to the arithmetic mean of the values of that attribute in the “complete” training data. For discrete valued attributes, we assigned missing values by random sampling based on the frequency of its occurrence in the complete training data. This sampling is based on Roulette wheel selection, where the probability of selecting an attribute value linearly depends on the frequency of its occurrence in complete training data.
2. **Assign values during tree construction:** A std::pair<int,int> missing variable was added as a data member of the Treenode class that had the average of all the non-missing values, in case of a continuous valued attribute. All possible values for a discrete valued attributes are stored in a std::set<std::string>, hence the strings are stored in lexicographical order. The index of the attribute value occurring most frequently in data pertaining to the treenode is stored. A pair is used to separately calculate the above for positive examples and negative examples.

First approach gave an accuracy of 80.23% and second gave an accuracy of 80.91%.

So we assigned the values during decision tree construction and stored this pair into treenode->missing for handling missing values in testing data in the same way.

Constructing the Decision Tree:

Decision Tree is built using the recursive function makeTree() of Tree class. Terminating condition for recursion is when the current node (node) has only positive or only negative examples or the set of remaining attributes is empty. Otherwise, the index of the attribute among the remaining attributes (from root to the current node) having the maximum information gain is set as node->ano. Then, for each possible value of this attribute a new Treenode is created (child nodes). The data in the node is traversed once and assigned to the appropriate child node based on the value of the ‘ano’ attribute. Finally, makeTree() is called for each node in the vector node->children.

Decision Tree with Reduced Error Pruning:

Reduced Error Pruning is a technique used to correct the overfitting of decision trees. For this, the original training data is split into a training set and validation set. In our program, two thirds of the data has been randomly selected and assigned as training set and the remaining training examples are used for validation. Once these sets are built, the decision tree is built using the training set and accuracy of classification of the validation set is calculated. In Reduced Error Pruning, one at a time, each node of this decision tree is removed and the accuracy is calculated again. If the accuracy has improved, this node is permanently removed from the tree, else it is reinstated.

Random Forest:

Random Forests consist of several randomly build decision trees. A specific number of random training examples are sampled with replacement, and is used to build a decision tree. This process is repeated to build many decision trees. Further in each step of building the trees only a certain random subset of remaining attributes are considered as the attribute set. The prediction of an output is based on the mode of the prediction of all the decision trees.

Hyper Parameters:

**n** - training data size.

Number of random training examples per tree - **sqrt(n)**

Number of trees - **sqrt(n)**

Due to a small number of attributes, better accuracy is achieved by considering the attribute set to be all remaining attributes.

Results:

Some metrics used for evaluating the classifiers in our classification problem.

**Accuracy:** Number of correct classifications/Total number of classifications.

**Precision:** Number of True Positives divided by the number of True Positives and False Positives. Putting it another way, it is the number of positive predictions divided by the total number of positive class values predicted. It is also called the Positive Predictive Value (PPV).

**Recall:** Recall is the number of True Positives divided by the number of True Positives and the number of False Negatives. Putting it another way, it is the number of positive predictions divided by the number of positive class values in the test data. It is also called Sensitivity or the True Positive Rate.

**F-Measure:** (referred to as F-Score or F**1**-score) = (2\*precision\*recall)/(precision+recall).

By testing with our testing dataset of 16,281 examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Accuracy** | **Precision** | **Recall** | **F-Measure** |
| ID3 | 80.91% | 0.5872 | 0.6458 | 0.6151 |
| Reduced Error Pruning | 84.68% | 0.7027 | 0.6092 | 0.6526 |
| Random Forest | 84.34% | 0.6881 | 0.6167 | 0.6504 |

**Note:** For random forest, values may vary with each execution.

Conclusion:

**Appropriate Problems for Decision Trees:**

* Problems where instances are represented by attribute-value pairs.
* The target has discrete output values.
* Disjunctive description maybe required to represent hypotheses.
* Problems where there maybe errors in the training data. (Are robust to errors in classification of training data, as well as in attribute values).
* Problems where the training data may have missing attributes.

**Problems of overfitting** can be effectively reduced by reduced error pruning or majority voting of random forests.

Random forests is effective for problems with **high number of attributes** in training data and high size of training data. In our case, the problem had only 14 attributes and selecting sqrt(no.of remaining attributes) at each node, reduced the size of the decision tree too much, giving accuracy of 81% only. So for problems with large number of attributes and size of training data, random forests gives a better classifier.