T-61.3050 Term Project, final report
Using decision trees for spam classification

Jori Bomanson (81819F)
jori.bomanson@aalto.fi

Sami J. Lehtinen (44814P)
sjl@iki.fi

November 20, 2011

1 Abstract

We programmed and trained a univariate binary classification tree to detect spam email. We studied the importance of different training set sizes, as well as pre- and postpruning techniques. We observed that a single tree is limited by a maximum accuracy, and further improvements will require introducing other methods.

2 Rationale

The choice of using a decision tree to solve a task with boolean variables seemed natural. Such a tree could definitely match whatever complexity there would be in the training data. In fact, we figured our success would be determined mainly by how we would manage to avoid overfitting.

Decision trees were also tempting in that they are much easier to visualize than, e.g., naive Bayes classifiers. For example, see figure 8.

One reason to study decision trees was that we assumed many groups would take on naive Bayesian classifiers, and we wanted to do something a bit different.

When we started the project, we actually did not realize that we could use existing libraries for the job. This led us to implement a decision tree algorithm by hand instead of just taking, e.g., an existing $C5.0$ [3] implementation.

3 Principles of decision trees

Decision trees are a non-parametric method for data classification. They do not try to approximate any model behind the data, instead relying on a tactic of "similar inputs produce similar outputs".

Basically, our algorithm implements the classification tree construction algorithm in [1, p. 179]. There is one change, however. We thought it would be a good idea to use the split entropy when checking for minimum entropy. The pseudocode is listed in figure 1.

3.1 Calculating impurity

We chose to measure impurity with the entropy function [1, p. 176][1].

[1] Alpaydin cites "(Quinlan 1986)" for the impurity algorithm.

$$\text{Impurity on a node m} \quad \mathcal{I}'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p^i_{mj} \log p^i_{mj}$$

$$\text{where} \quad 0 \log 0 \equiv 0$$

$$p^i_m = P(\text{Instance reaching node m belongs to class i})$$

$$K = 2 \tag{1}$$

```
proc generate_tree(data) ≡
  impurity ← MAX
  for i := 1 to num_columns do
      im ← calculate_impurity(data, column)
      if im < impurity
        then
             impurity ← im
             split_column ← i
      fi
  od
  if impurity < θ
    then
         comment: label will be the majority class in the data.
         create_leaf_node(column, label)
         exit
  fi
  X_tagged, X_untagged ← split(data, column)
  comment: Generate both child nodes by splitting with the chosen attribute, i.e., column.
  generate_tree(X_tagged)
  generate_tree(X_untagged)
.
proc calculate_impurity(data, column) ≡
  X_tagged, X_untagged ← split(data, column)
  return impurity1(X_tagged, X_untagged)
.
```

Figure 1: Pseudo-code for tree generation algorithm.

3.2 Pruning the tree

Decision trees can be trained to perfectly match the training data [1, p. 182]. This will lead the model to gross overfitting, which manifests as a large generalization error. This can be avoided by pruning the tree. There are two non-exclusive approaches to pruning, prepruning and postpruning.

Without pruning, a tree with a training set size of 1000 looks like figure 2.

Still, against the test set the non-pruned tree faired surprisingly well, achieving an accuracy of over 94 percent.

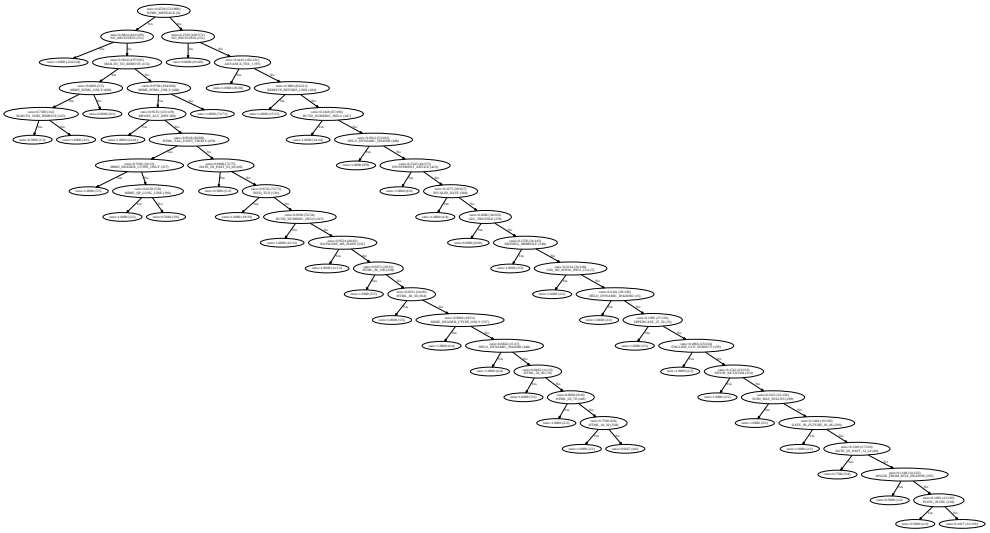

Figure 2: Tree from a training set of 1000 elements that has not been pruned.

Trees with various training set sizes and prunings are depicted in figures 6, 7, 8, and 9.

A more detailed view of a tree is in figure 10.

3.2.1 Prepruning

In prepruning, the recursion of the training algorithm is stopped once certain limits are reached. Examples include impurity and number of data rows.

In our first versions of the classifier we only did prepruning, which resulted in very small and tidy trees, but the accuracy left something to be desired (94-95 percent). We used one of the suggested approaches from [1], where we make the tree as pure as possible (no prepruning) and handle overfitting by postpruning the tree.

Comparing tables in figures 4 and 5, removing prepruning did not have a large effect for the final accuracy. This is probably because our chosen prepruning parameters were so tight already.

We implemented prepruning by having three different stopping criteria in our algorithm to limit the depth of the resulting tree. The first ensures that there is always at least one feature to base a decision on, the second that there are enough messages and the third that there is enough entropy to warrant further recursion.

We ended up using rather marginal prepruning requirements. We required a minimum of 2‰ of rows to have reached a node and at least 4% impurity in order to split it further. This was enough to cut down the size of the resulting trees and speed up the training process, but left significant emphasis to the

Size	Pruning	Training time	Accuracy	Precision	Recall	Perplexity
200	post	10	0.919	0.984	0.892	1.247
200	both	11	0.931	0.981	0.914	1.332
500	post	19	0.939	0.984	0.923	1.203
500	both	16	0.936	0.983	0.920	1.258
800	post	28	0.952	0.984	0.943	1.214
800	both	28	0.948	0.984	0.938	1.207
1000	post	38	0.953	0.984	0.944	1.215
1000	both	38	0.944	0.985	0.930	1.165
2000	post	118	0.961	0.984	0.956	1.194
2000	both	113	0.966	0.983	0.965	1.175
4000	post	381	0.971	0.982	0.973	1.141
4000	both	380	0.970	0.985	0.970	1.136
6000	post	683	0.971	0.985	0.971	1.128
6000	both	668	0.977	0.991	0.974	1.138
8000	post	1112	0.977	0.988	0.978	1.111
8000	both	1033	0.971	0.982	0.975	1.117

Figure 3: Effects of pruning on accuracy and other characteristics with various training data sizes.

Size	Training time	Accuracy	Precision	Recall	Perplexity
200	10	0.919	0.984	0.892	1.247
500	19	0.939	0.984	0.923	1.203
800	28	0.952	0.984	0.943	1.214
1000	38	0.953	0.984	0.944	1.215
2000	118	0.961	0.984	0.956	1.194
4000	381	0.971	0.982	0.973	1.141
6000	683	0.971	0.985	0.971	1.128
8000	1112	0.977	0.988	0.978	1.111

Figure 4: Effect of training data size on postpruned trees (part of data from table in figure 3).

Size	Training time	Accuracy	Precision	Recall	Perplexity
200	11	0.931	0.981	0.914	1.332
500	16	0.936	0.983	0.920	1.258
800	28	0.948	0.984	0.938	1.207
1000	38	0.944	0.985	0.930	1.165
2000	113	0.966	0.983	0.965	1.175
4000	380	0.970	0.985	0.970	1.136
6000	668	0.977	0.991	0.974	1.138
8000	1033	0.971	0.982	0.975	1.117

Figure 5: Effect of training data size on pre- and postpruned trees (part of data from table in figure 3).

postpruning step described in section 3.2.2.

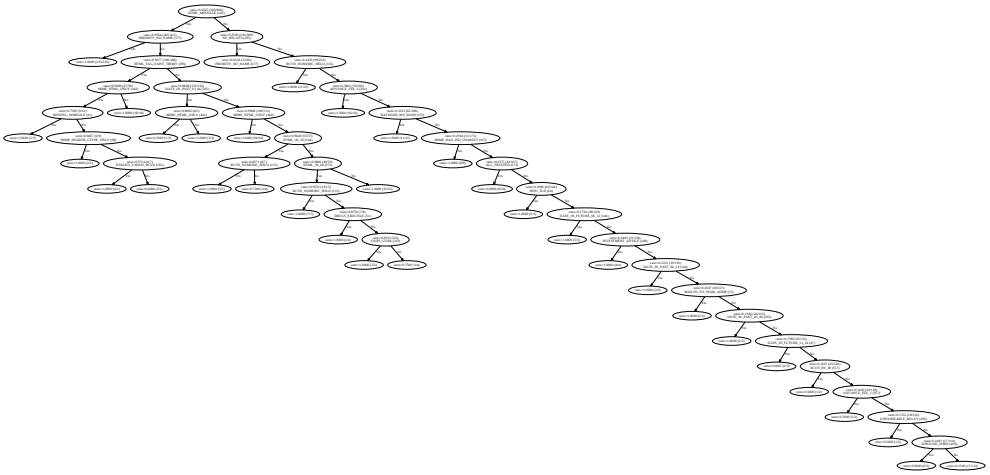

Figure 6: Prepruned tree from a training set of 1000 elements.

3.2.2 Postpruning

Postpruning is used to reduce generalization error by using a special pruning data set to check whether removing a subtree from the decision tree will result in the same or smaller error against the pruning set. The complexity of such subtrees is deemed unjustified and we expect to avoid overfitting by substituting them with simple leaf nodes, labeled according to the majority class of the training instances reaching them [1, p. 183].

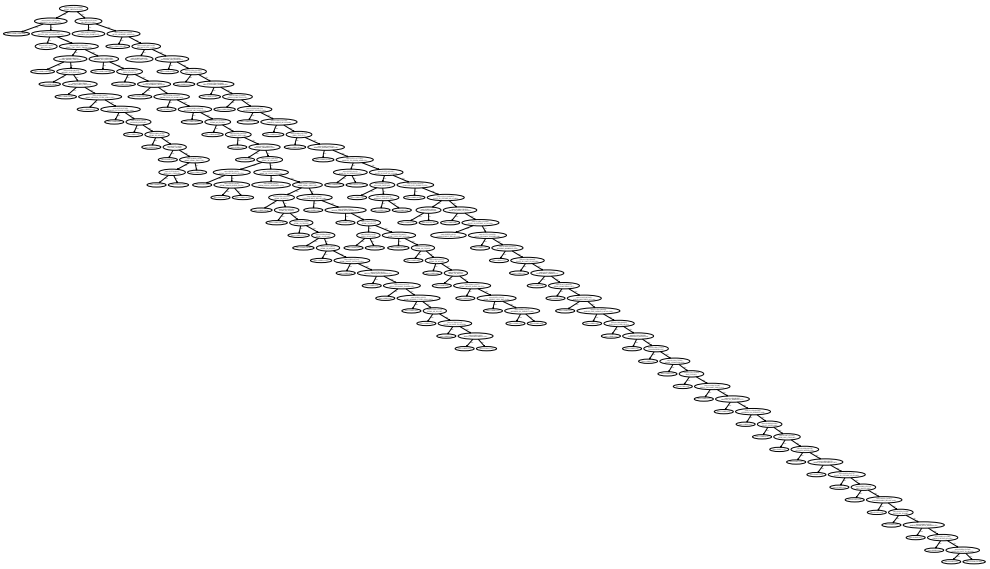

Figure 7: Prepruned tree from a training set of 6000 elements.

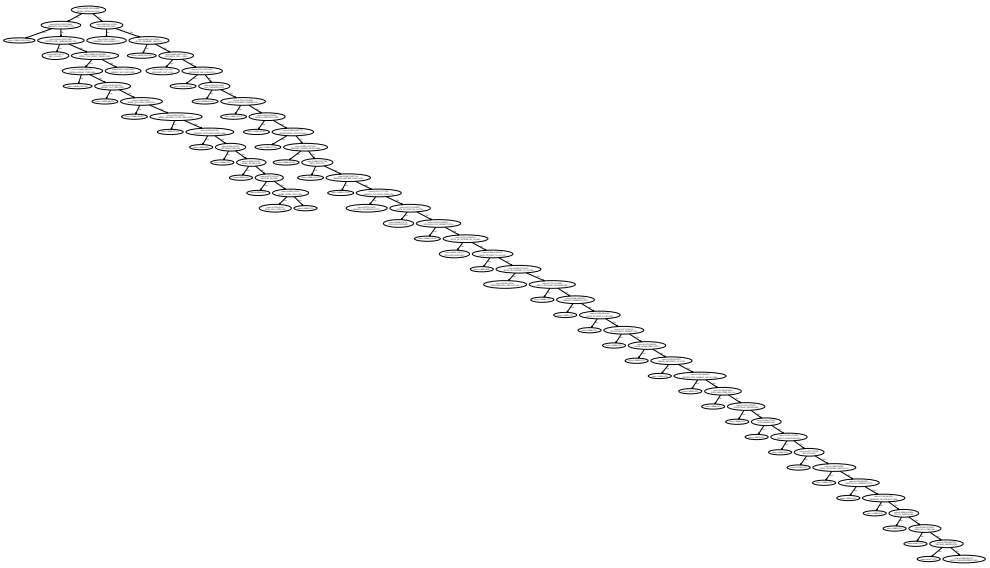

Figure 8: Pre- and postpruned tree from a training set of 6000 elements.

Trees that are both pre- and postpruned take a small hit in accuracy in comparison to only postpruned trees, but are slightly faster to train according to our observations.

4 Validation of our approach

4.1 Accuracy, precision, recall, and perplexity

The criteria for measuring the goodness of the classifier are the same as the ones used in the data challenge. As described in [2], accuracy is the fraction of test set messages classified correctly. The precision is the ratio of correctly classified spam messages to all messages classified as spam. Recall is the ratio of correctly classified spam messages to all spam messages.

Perplexity is calculated with

$$P = \exp(-\operatorname{mean}(\ln P_i)) \tag{2}$$

where P_i is the probability of the email message being of the correct class.

For the perplexity calculation, we return the probability that a message is spam based on the ratio of spam messages left at the leaf node that ends up deciding whether the message is spam or not. This seems a decent metric for calculating perplexity, at least when comparing to what other groups reported in the data challenge.

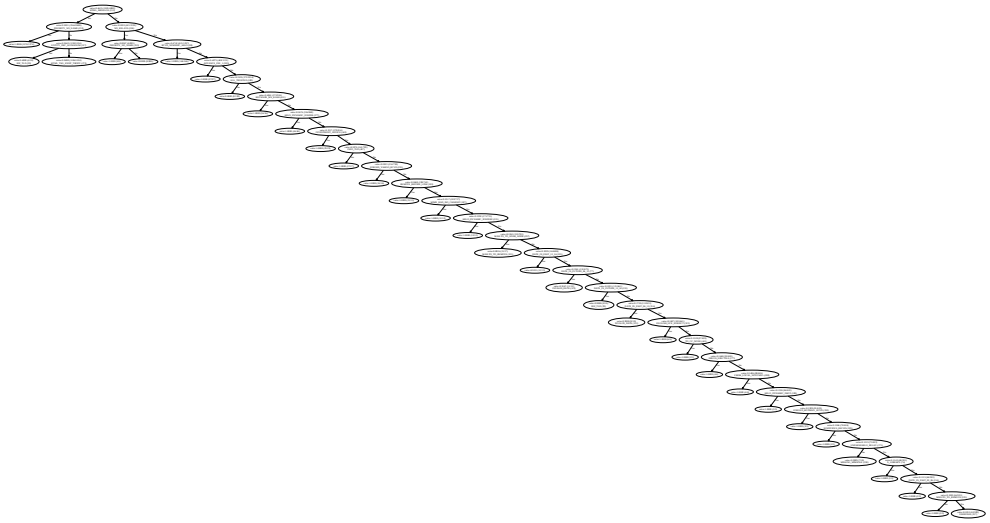

Figure 9: Postpruned tree from a training set of 6000 elements.

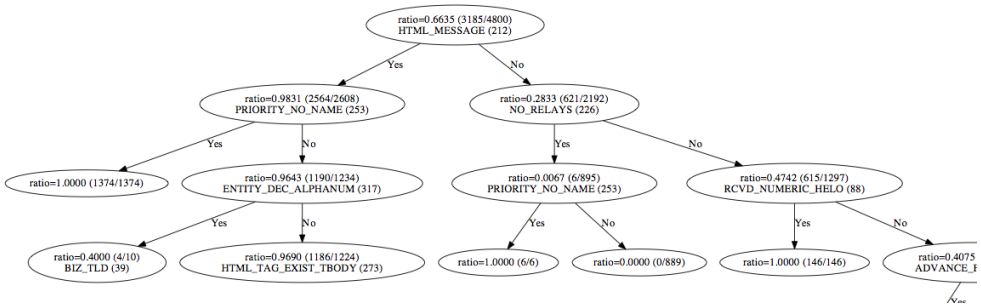

Figure 10: Detail from a postpruned tree from a training set of 6000 elements.

4.2 K-fold cross validation

We implemented 10-fold cross validation to our classifier. This gave more stable values for generalization error when compared to repeatedly training the classifier without the cross validation.

One part of the training data set was used for postpruning the tree, see section 3.2.2.

4.3 Dummy model

We can compare our classifier against a dummy model of the data, where the dummy would just classify messages as spam based on the prior probability of a message being spam or ham. In our case, the dummy would output “SPAM” with approximately 70% chance, without looking at the features of the messages at all. Such a classifier would have an accuracy of approximately 58%. A

classifier which would just always output "SPAM" would have an accuracy of 70%. Our classifier beats both of these dummies.

5 Results with different training set sizes

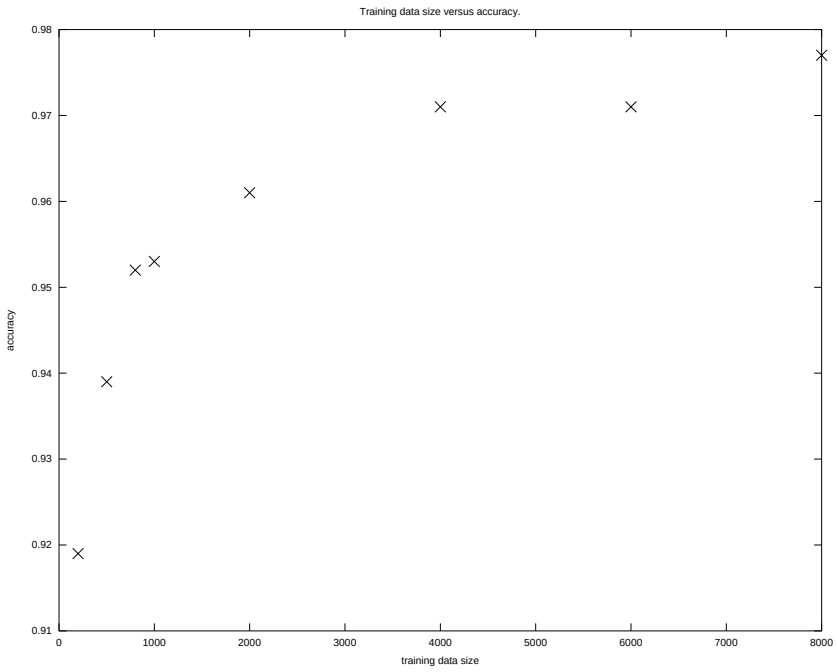

Figure 11: Accuracy plotted as a function of training data size.

As can be seen from figure 3, training the classifier with 8000 items yields only approximately 1.5 percentage points better accuracy than training with 1000 items. From figure 11, the elbow point is somewhere around 2000-3000 items in the training set.

5.1 Training algorithm complexity

Based on figure 3, the training algorithm looks like a $O(n \log n)$ time algorithm. This is definitely not quadratic, although not linear either.

The time values for very small training set sizes have bias, as `numpy.loadtxt()` takes a long time (several seconds), to load the data set.

6 Data challenge

The data challenge led us to realize that our - at the time merely prepruned - decision tree was not the magical black box method that would wipe the table that we hoped it would be. As a consequence, we did the inevitable and gave postpruning as well as K-fold cross validation a try.

7 Conclusions

The chosen approach did not achieve the same high marks for accuracy as the winners of the data challenge. Tweaking of the approach is naturally possible, but it seems that decision trees are more suited in this as a part of combined classifier instead of on their own. This does not mean that the approach is completely without merits; decision trees were applied here without prior knowledge to the distribution of the spam messages. In the data challenge, the winning teams had tweaked the prior probabilities for spam to match the distribution in the data set. In the decision tree model, this is not necessary or even possible.

Looking at the generated decision trees, it is easy to follow the process by eye (at least with the smaller trees). This, we think, helps a lot in validating the approach.

We implemented the algorithm with Python and Numpy ourselves. We think this gave us deeper insight into the inner workings of the algorithm than just using an existing software package.

From the final results, we see (figure 3) that we get the best accuracy, 97.7%, with 6000 data points.

It would have been interesting to see how boosting [1, p. 360] would have affected the results; unfortunately, we did not have time to try it. Because our classifier was already producing quite good results, boosting might have given that extra percentage point or so. Because the classifier already hits the 95% accuracy with relatively small training data size, training an ensemble of trees would have been feasible and would have very likely improved the accuracy further.

8 Comments on project difficulty

Writing the classifier by hand took quite a bit of effort. Even though the algorithm is simple in principle, testing it was perceived hard to automate. As a result, testing was done adhoc by the developer, taking time off writing features (such as boosting).

We underestimated the difficulty of the task, which became apparent when the results of the data challenge were published.

	Classifier	Report
Sami	32h	16h
Jori	22h	10h

Figure 12: Estimated hours for the project

The project was challenging, but not too much so.

References

[1] Ethem Alpaydin, *Introduction to Machine Learning*. Massachusetts Institute of Technology, Cambridge, Massachusetts, 1st edition, 2004. 415 pages. ISBN 0-262-01211-1.

[2] *Term Project 2011: Spam or Ham?*. https://noppa.aalto.fi/noppa/kurssi/t-61.3050/term_project [Web article]. 2011-11-6. [Cited 2011-11-19].

[3] *Data Mining Tools See5 and C5.0*. http://rulequest.com/see5-info.html [Web article]. 2011-01. [Cited 2011-11-20].

A Python code for the classifier

The code is available for scrutiny at GitHub, https://github.com/sjlehtin/spam-or-ham/blob/master/decisiontree.py.