Assignment 2: Decision Trees Algorithm

Implementation details

1. Perform cross-validation

To ensure that we will be able to use the same folds for our cross validations in future coursework without having to save the indices, we decided to write a general function. This, given a data set D, the total number of folds N, and the fold number Y, splits the data into N folds and returns the Yth fold for validation and the rest for training. If the number of examples is not evenly divisible by the number of folds (as it was not in our case, having 1004 examples and using 10 folds), we attach the remaining examples to the validation set. For example, calling this function with D being 1004 examples, N=10 and Y=2, the validation fold becomes the examples from index 100 to 204 and the training fold takes the rest. Thus, actually doing the 10-fold cross validation means calling this function 10 times and changing the Y parameter.

```
for i = 1:fold
    [ training, validation ] = crossValidationSplit(fold, data, i);
end
```

2. Select the best attribute in each node

Our implementation dictates the best attribute is the one that results in the greatest information gain. We implemented the information gain and entropy functions as described in the lectures and course book. We did however try to optimise our id3 algorithm by inspecting how our best attribute split the data. If the best attribute failed to split the examples, we realised it was unnecessary to continue the recursion until all attributes ran out and terminated the process by adding a leaf with the modal target value straight away.

3. Compute the average results

We create a cell array to hold the 10 confusion matrices produced by the 10 fold cross validation. Then we take the average of the 10 matrices using MATLAB's mean and reshape functions.

```
mean(reshape(cell2mat(matrices), [ size(matrices{1}), length(matrices) ]),
ndims(matrices{1})+1) ;
```

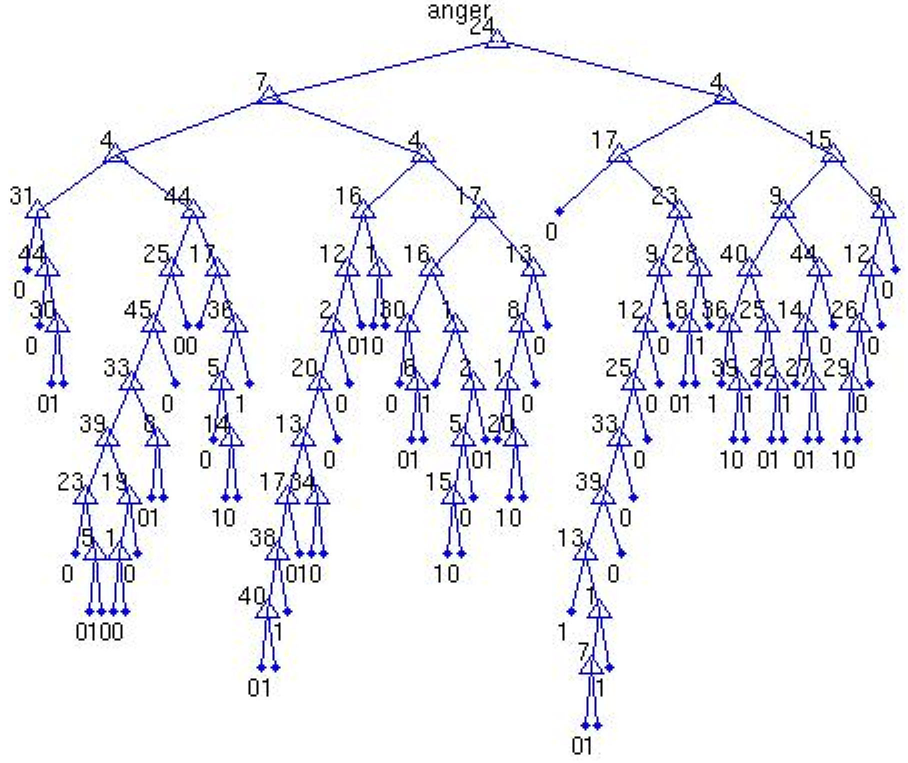
`cell2mat(matrices)` takes the cell array of the 10 matrices and combines them side by side. We then use the reshape function to change the combined matrix into a 3D matrix, the third dimension being made up of the multiple confusion matrices.

`reshape(cell2mat(matrices), [ size(matrices{1}), length(matrices) ])`

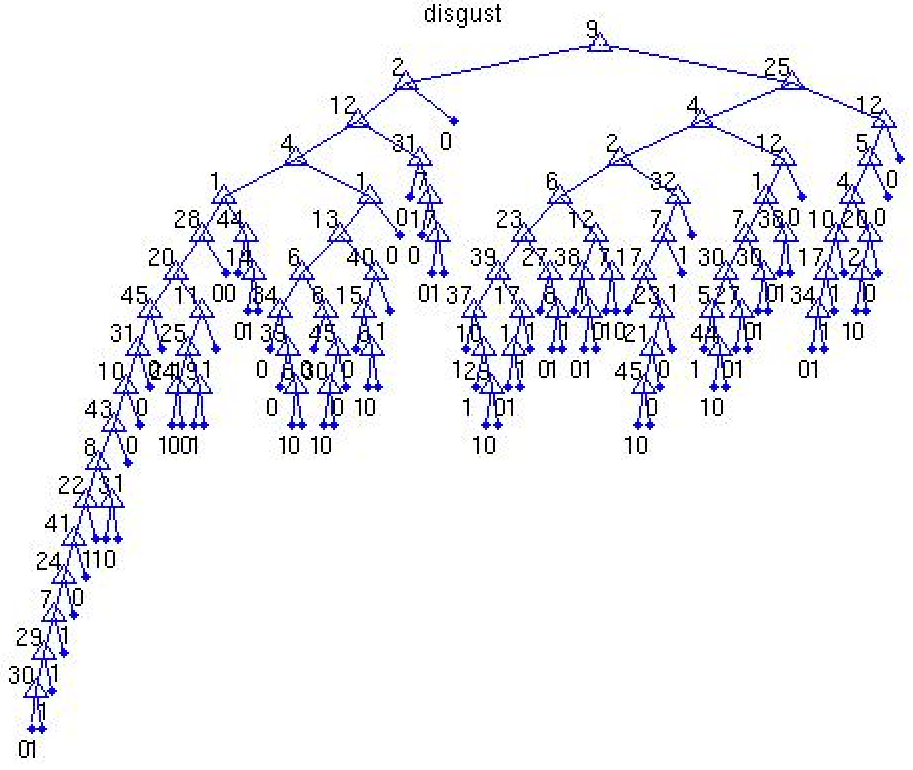
We then use the mean function to take the average in each dimension.

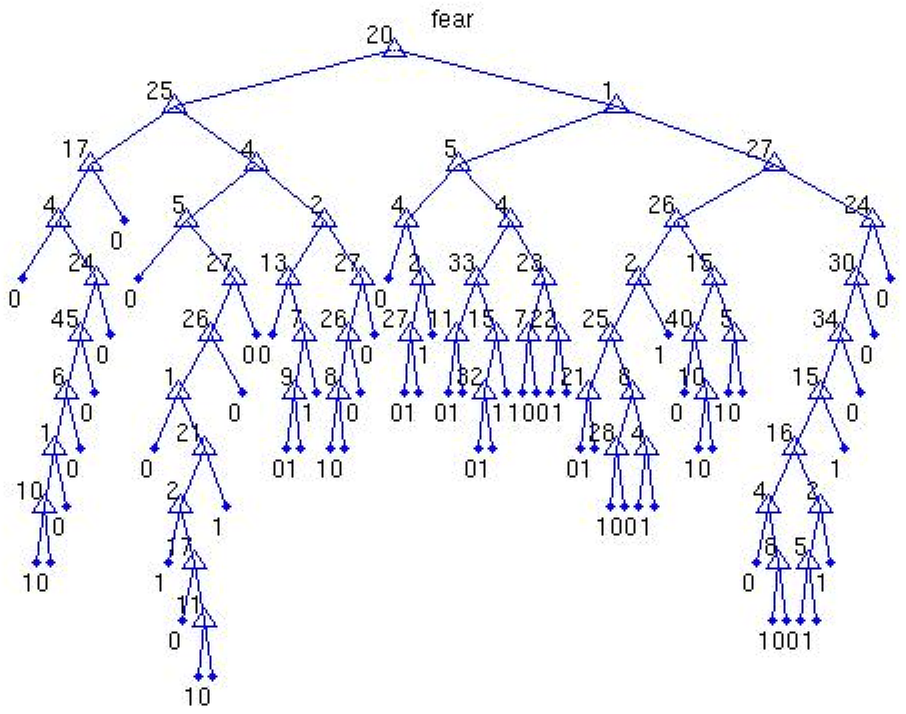
Tree figures

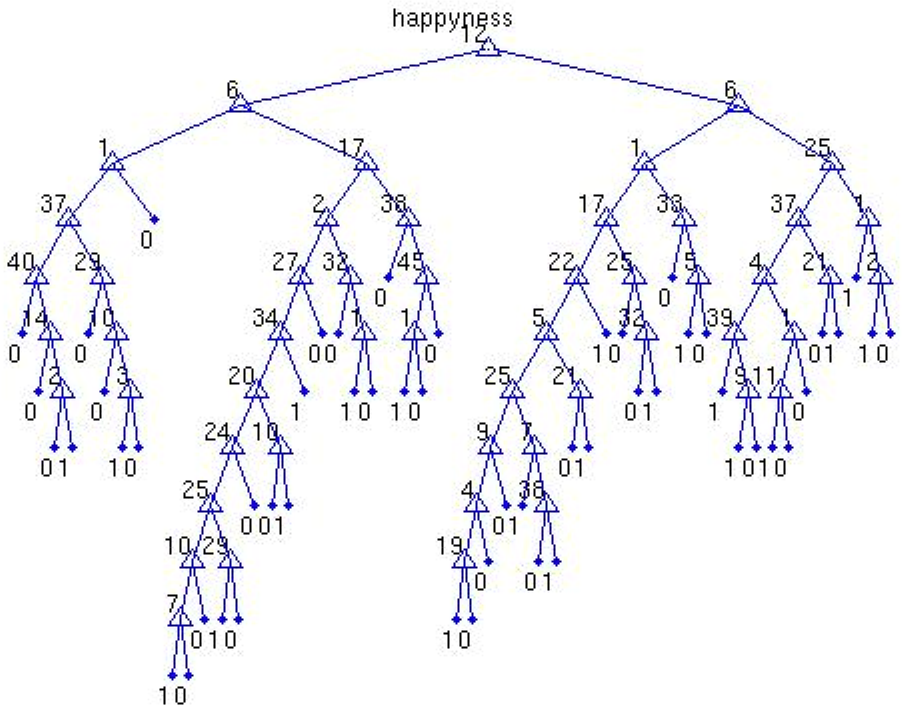
anger

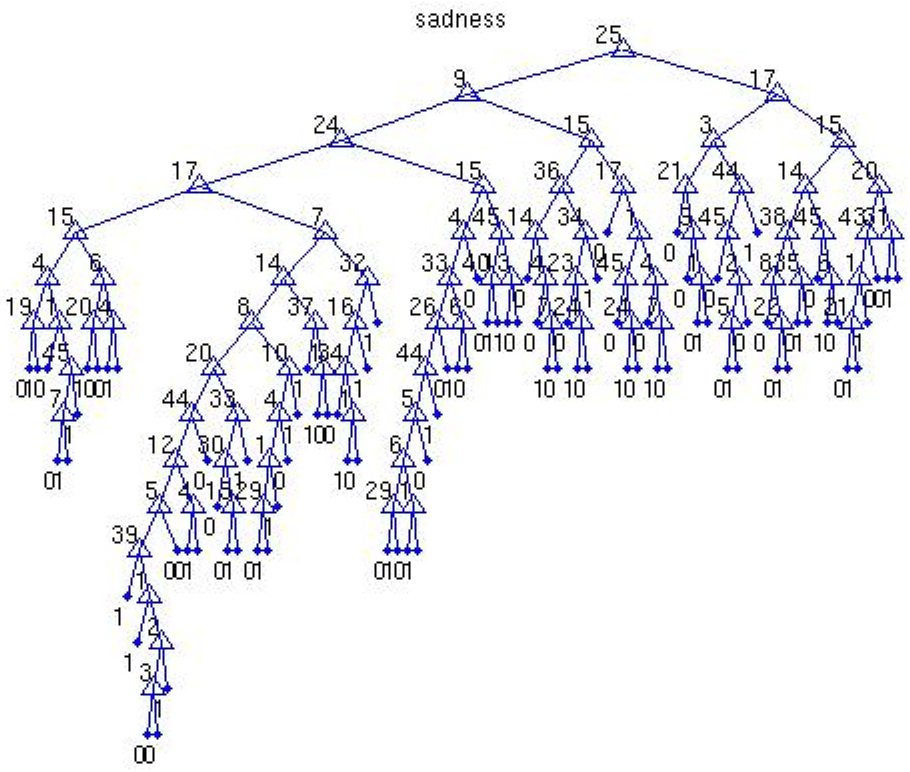


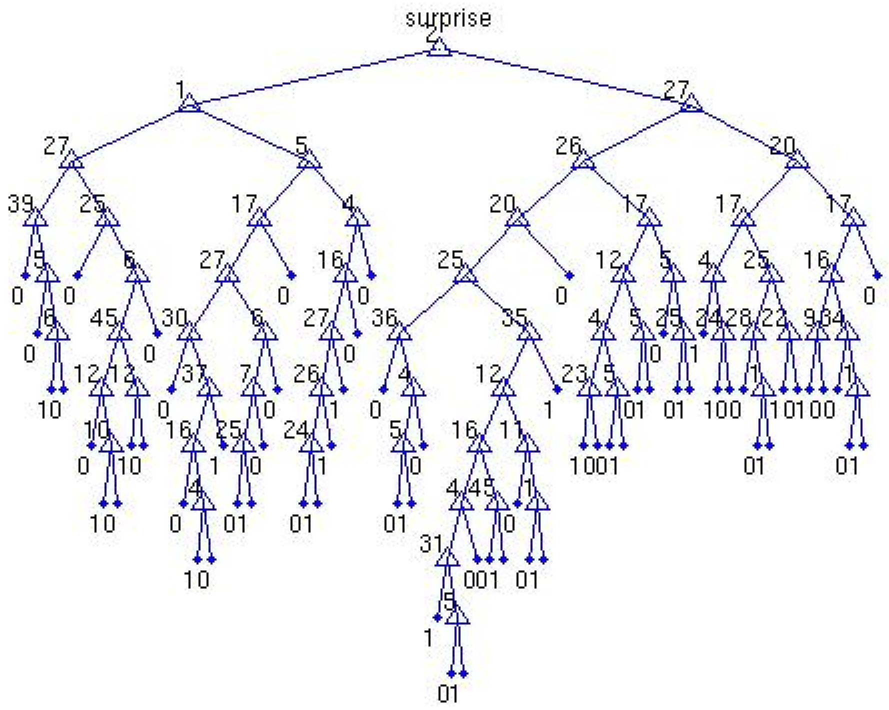
disgust




fear


happyness


sadness


surprise

Confusion matrix

The combined matrices are (average of confusion matrices of the 10-fold validation):

	clean data	noisy data
Approach 1 random selection	$\begin{matrix} 8.6 & 2.0 & 0.7 & 0.6 & 1.3 & 0.4 \\ 1.7 & 13.6 & 0.5 & 1.9 & 1.3 & 1.2 \\ 0.5 & 0.3 & 8.0 & 0.8 & 1.4 & 1.1 \\ 0.1 & 0.9 & 1.0 & 18.3 & 1.3 & 0.8 \\ 1.8 & 1.5 & 0.6 & 1.3 & 7.7 & 0.7 \\ 0.2 & 0.8 & 1.4 & 0.7 & 1.1 & 17.9 \end{matrix}$	$\begin{matrix} 2.0 & 1.1 & 1.7 & 1.0 & 2.2 & 0.8 \\ 1.3 & 12.4 & 1.6 & 1.8 & 1.0 & 0.8 \\ 1.7 & 1.9 & 10.0 & 2.2 & 1.3 & 1.8 \\ 0.9 & 1.5 & 1.4 & 15.7 & 0.5 & 1.3 \\ 2.1 & 0.9 & 0.8 & 0.7 & 5.3 & 1.2 \\ 1.0 & 1.5 & 1.8 & 0.9 & 1.2 & 15.7 \end{matrix}$
Approach 2 scoring leaves	$\begin{matrix} 9.0 & 1.4 & 0.8 & 0.5 & 1.4 & 0.5 \\ 1.6 & 15.0 & 0.4 & 0.9 & 1.0 & 1.3 \\ 0.8 & 0.5 & 8.0 & 0.4 & 0.9 & 1.5 \\ 0.7 & 0.8 & 0.6 & 19.2 & 0.9 & 0.2 \\ 1.9 & 1.6 & 0.6 & 0.9 & 7.8 & 0.8 \\ 0.5 & 0.8 & 1.3 & 1.0 & 0.7 & 17.8 \end{matrix}$	$\begin{matrix} 2.1 & 0.9 & 1.9 & 1.1 & 2.0 & 0.8 \\ 1.4 & 13.4 & 1.7 & 1.4 & 0.4 & 0.6 \\ 1.1 & 1.3 & 11.2 & 2.2 & 1.5 & 1.6 \\ 1.0 & 1.2 & 0.8 & 16.7 & 0.5 & 1.1 \\ 1.5 & 0.9 & 0.7 & 1.0 & 5.8 & 1.1 \\ 1.0 & 1.1 & 1.3 & 1.1 & 1.1 & 16.5 \end{matrix}$

Recall/precision/F Measure/Classification rate

Clean and Noisy side by side comparison:

Approach 1: random selection (to 3 s.f)

		Random	Random	Scoring	Scoring
Class	Classification Measures	Clean data	Noisy data	Clean data	Noisy data
1 - Anger	Recall	0.632	0.227	0.662	0.239
	Precision	0.667	0.222	0.621	0.259
	F1	0.649	0.225	0.641	0.249
2 - Disgust	Recall	0.673	0.656	0.743	0.709
	Precision	0.712	0.642	0.746	0.713
	F1	0.692	0.649	0.744	0.711
3 - Fear	Recall	0.661	0.529	0.661	0.593
	Precision	0.656	0.578	0.684	0.636
	F1	0.658	0.552	0.672	0.614
4 - Happiness	Recall	0.817	0.737	0.857	0.784

	Precision F1	0.775 0.796	0.704 0.720	0.838 0.848	0.711 0.746
5 -Sadness	Recall Precision F1	0.566 0.546 0.556	0.482 0.461 0.471	0.574 0.614 0.593	0.527 0.513 0.520
6 - Surprise	Recall Precision F1	0.810 0.810 0.810	0.710 0.727 0.719	0.805 0.805 0.805	0.747 0.760 0.753
	CR	0.713	0.605	0.738	0.650

Analysis of the cross validation experiments

Firstly, it is evident for both prediction techniques that the classification metrics are always worse for noisy data than for the clean data, but that in almost all cases, our scoring resolution method is an improvement on random selection. It is also notable that the performance gain between the methods is greater in the noisy set than the clean set. The measures of class anger and sadness is significantly lower in the noisy data. This could because the tree trained is affect by the noise in the dataset thus it can not be correctly recognised.

Ambiguity Question

Both resolution methods have three cases of classification to cover. The first case occurs when no tree positively classifies an example, the second when only one tree positively classifies an example and the third when multiple trees positively classify an example. The easiest case to solve is case 2, and in both approaches we handled this by selecting the only class that positively identified.

Approach 1: Random selection
Our first approach was to pick an emotion at random if no tree had identified the emotion, or to pick at random from the set of positive emotions, if multiple trees positively classified the example. We decided to implement a randomised approach because it's simple and would serve as a good comparator to a more considered second implementation. One problem with random selection is that not all examples are classified equally, at least, there are three different base cases in our ID3 algorithm and on inspection one might consider some base cases as a stronger classification than others. For example, it is reasonable to suggest that the base case in which all training items share a common target is a stronger classification

than the base case in which there are no attributes left to divide the remaining targets but they still disagree. Another problem with random selection is that it doesn't consider the complexity of the classification hypothesis - by this we mean the depth of the leaf within a tree. It might be important to consider whether we should favour complex or simple hypothesis before we select at random.

Approach 2: Scoring leaves
We developed our second approach to try and address some of the issues we discussed in approach 1. We altered our id3 algorithm so that we could assign a confidence value to the leaves of our decision trees. A confidence value allows us to select purposefully from multiple instances of positive classification. The score assigned to leaves created from unanimous examples was equal to the number of examples that reached the case. If two trees both returned a positive from two leaves both created this way then the winner would be the one we have seen more examples for, which makes sense. In both other cases, running out of attributes or examples, we assigned a score equal to the mean of the example binary targets i.e the proportion of positive examples. This means that when deciding between two such leaves we will always favour the one that had a higher proportion of positive examples as it is more likely to be accurate.

Approach 3: Leaf complexity
An approach we considered but did not implement resolved ambiguity by following Occam's razor; that is, always selecting results from the shallowest leaf because the depth of leaves directly corresponds to the complexity of a hypothesis. However, it is debatable whether a simpler hypothesis is necessarily always better.

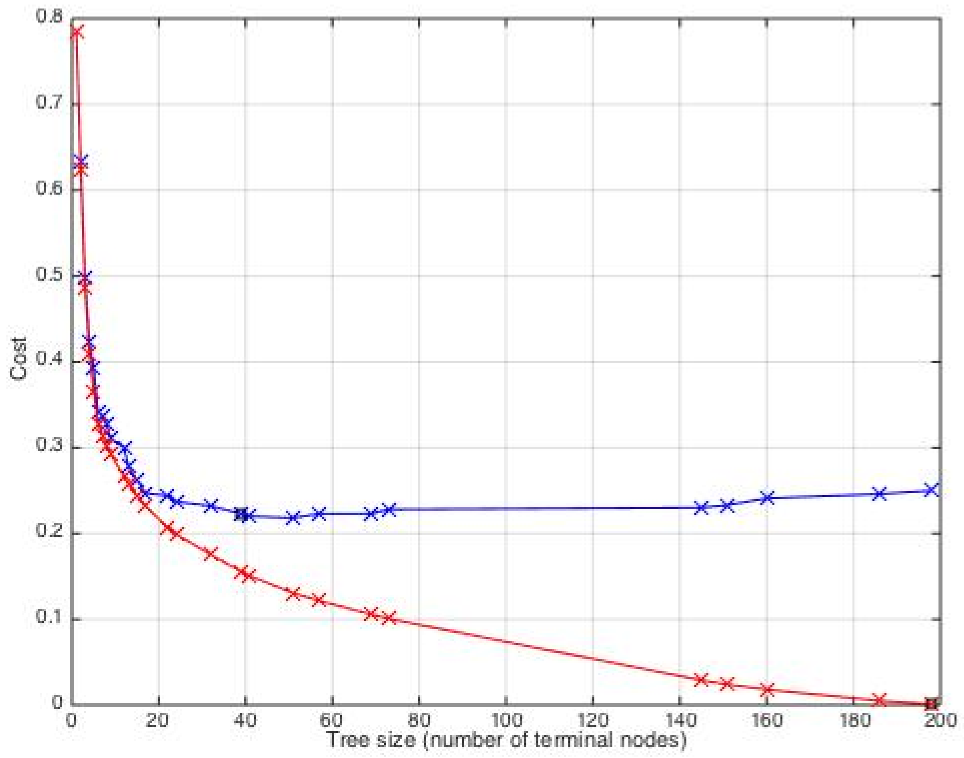
Conclusion: scoring improves tree measures, but not as significantly as we would have hoped. The improvement is greater in the noisy data set than in the clean data set. We think this is because the biggest advantage is when we select a unanimous leaf over leaves created by taking modal values. In the clean data far fewer leaves are created via contradiction (running out of attributes) because by definition the data is 'clean'. In the noisy set, we see 'modal' leaves occurring much more often and so our selection bias is more effective.

Noisy-Clean Datasets Question

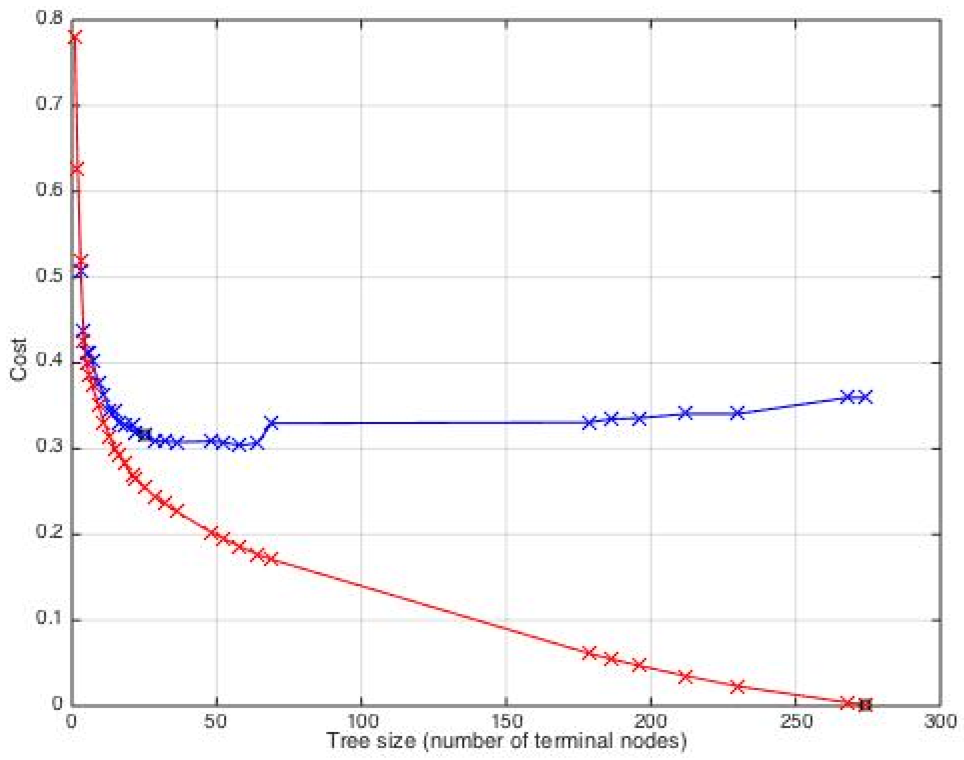
The noisy data is just so because it contains many example of contradictory data - training items that share all attribute values but differ in their classification. The consequence of this is that when the trees are constructed, it's more frequently the case that a leaf is created because there are no attributes left that can split the examples. In such instance the leaf takes the modal value and can't truly represent the classifications of all it's examples. When a tree has more leaves like this it's intuitive that it should more frequently make mistakes. The table suggests our scoring algorithm performs better on the noisy data than the random method which makes sense because it favours leaves constructed when all the examples had matching classifications. The scoring algorithm also favours the leaves that had a higher proportion of consistent classifications which we believe makes it more probable that the prediction will be correct. However, when you look at the results, it’s apparent that our improved method was really only slightly better than our random choice method.

Pruning Question

Clean data:



Noisy data:



The pruning_example function first generates a decision tree, then measures the error rate of the tree on both training and unseen data and on both clean and noisy data sets. The function calculates the performance of the tree at successive levels of pruning. That is, it generalises or prunes the tree using different ‘quotas’ of nodes and calculates performance measures each time.
We can see that when the tree size is small and has only one or two nodes then the tree is too general and the error rate is large. One or two attributes is not enough to accurately predict a classification. At the other end of the scale, we see a zero error rating on training data. This is because the tree is now an exact representation of the training data in tree form. There is no generalisation and the tree will accurately classify any item of training data. It follows that with this many nodes the tree fails to generalise on unseen data and performs relatively far worse than with training data.

Trees induced from noisy data are much more likely to overfit the data. In trees with a smaller amount of nodes, it is much more likely for these nodes to be irrelevant attributes, which may fit the noise in the data. These ‘outliers’ can have a negative effect on the accuracy of unseen data.

Two graphs are then drawn on the same axis, of tree size (the number of terminal/leaf nodes) against the cost/error rate of the tree. The blue line represents the nodes and costs from using cross-validation, to test the cost/error rate on unseen data. While the red lines represents those from resubstitution. The resubstitution cost is based on the same sample that was used to create the original tree, so it fits can fit training data when the number of nodes increases, but under estimates the likely cost of applying the tree to unseen data.

From the information in the graph, the optimal tree sizes (lowest cost) can be said to be as follows:

	Clean dataset	**Noisy dataset**
Optimal tree size (no. terminal nodes)	52	56