Assignment 2: Decision Trees Algorithm
— Report —

Robert Jaworzyn, Aodhgan Gleeson, Ben Fadero, Levi Epstein.
{rj616, amg315, bof16, lbe16 }@doc.ic.ac.uk

Teaching helper: Jiankang Deng
Course: CO395, Imperial College London

13th February, 2017

1 Implementation Details

1.1 Selecting the best attribute in each node

The ID3 algorithm is a decision tree learning algorithm which selects attributes for nodes based on their information gain. The attribute with the highest information gain at any point is the best candidate for a node.

This algorithm generally works with positive and negative examples for a given target. However, the data in question deals with 6 different targets (emotions, in this case). It is therefore necessary to train 6 separate trees, counting each emotion as a binary target - the decision tree for happiness, for example, would treat happy examples as positive and all other examples as negative.

To implement this concept in MATLAB, a function called *chooseBestDecisionAttribute* was created. This function was applied to each emotion, selecting the attribute with the highest information gain in the following way:

- The number of positive and negative examples are counted, and the total entropy is calculated using these values.
- The function then iterates through each attribute, calculating its information gain. The information gain is the reduction in total entropy caused by partitioning the set of data according to the attribute in question.
- The function stores the highest information gain calculated across all attributes, and this attribute is selected as the best attribute for the node.

1.2 Performing Cross-Validation

In general, cross-validation is performed by splitting the data into k folds, using k-1 folds for training and validation, and the final fold for testing. This process is then repeated k times, using a different fold as the testing fold each time. Splitting the data this way helps to prevent overfitting by giving a representation of a tree's performance on unseen data.

For the purposes of this task, the data was divided into 10 roughly evenly-sized folds (9 for training and 1 for testing). Using the training data, a tree was trained for each of the 6 emotions, and the perfomance of each tree was tested by comparing its predictions against the data provided. This was implemented by doing the following:

- A row of attributes from the testing data was passed through the trained trees (1 to 6), thus giving a prediction of which emotion the row of attributes corresponds to. This process was repeated for all rows in the testing data. Cases in which a set of attributes could fit multiple emotions will be discussed in section 4.2.

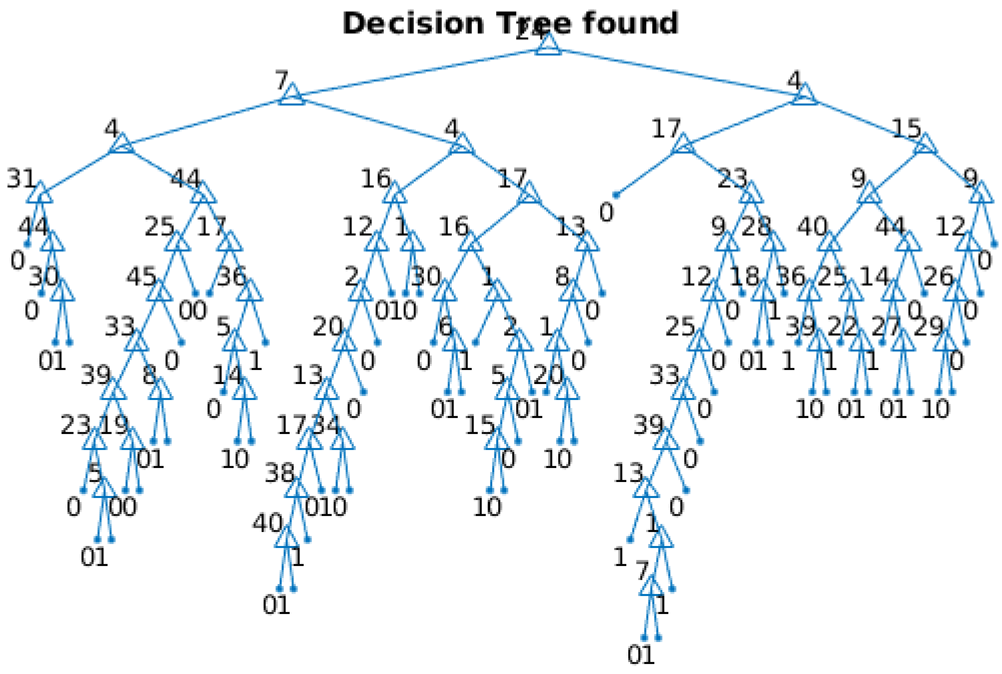
- The predictions were then compared against the actual data by creating a confusion matrix (see 3.1.1).

1.3 Computing the Average Results

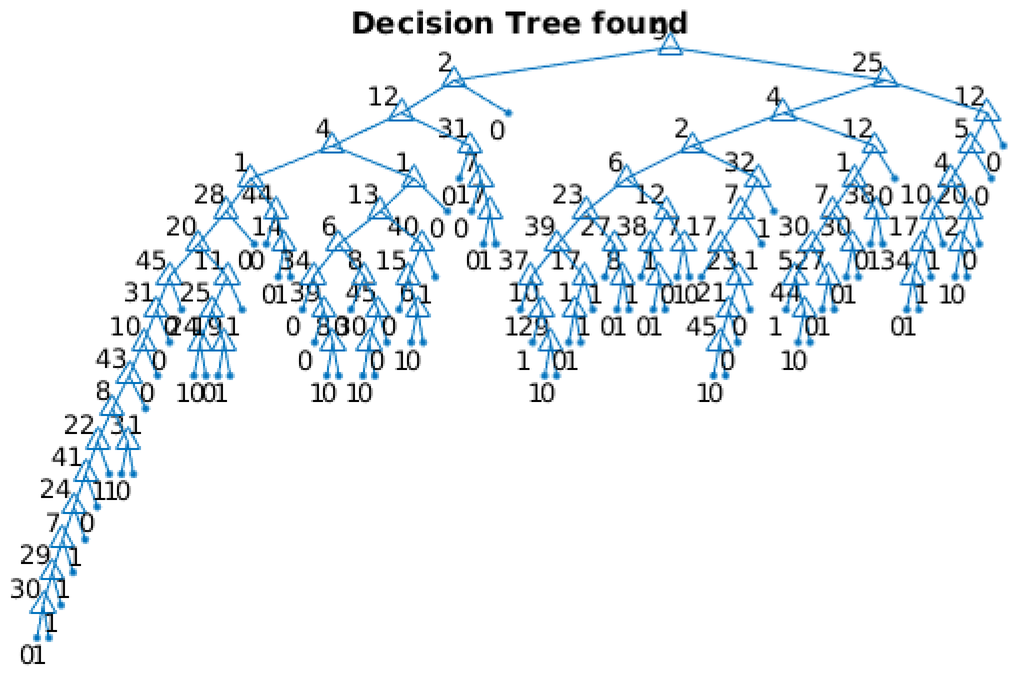
A confusion matrix was created for each of the iterations of the 10-fold cross-validation. The ten confusion matrices were then summed together to give a cumulative confusion matrix, from which the average precision, recall, F1 and classification rate could be calculated using the appropriate formulae.

2 Tree Figures

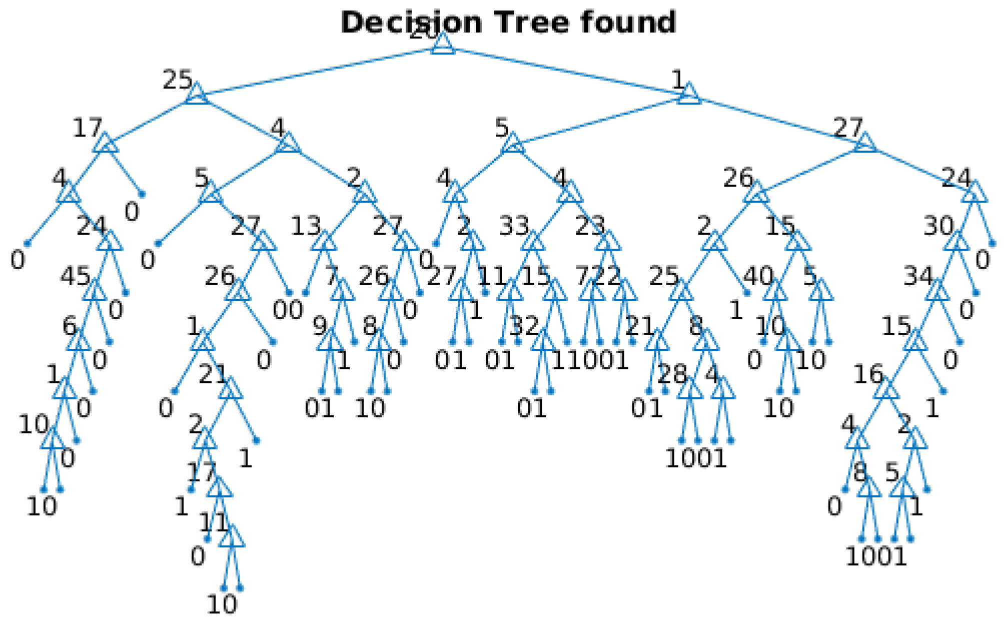
2.1 Emotion 1



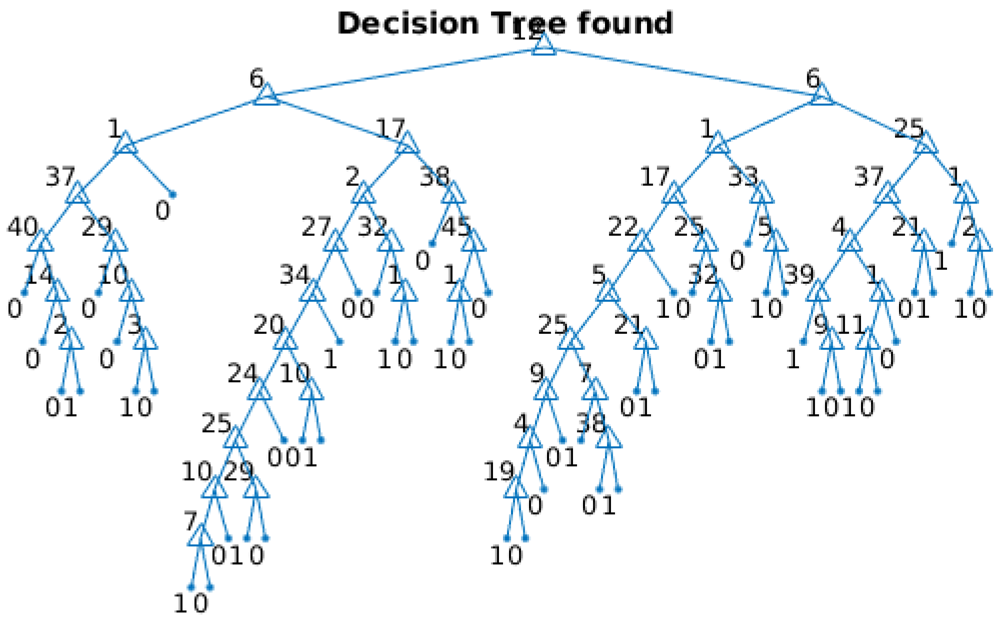
2.2 Emotion 2



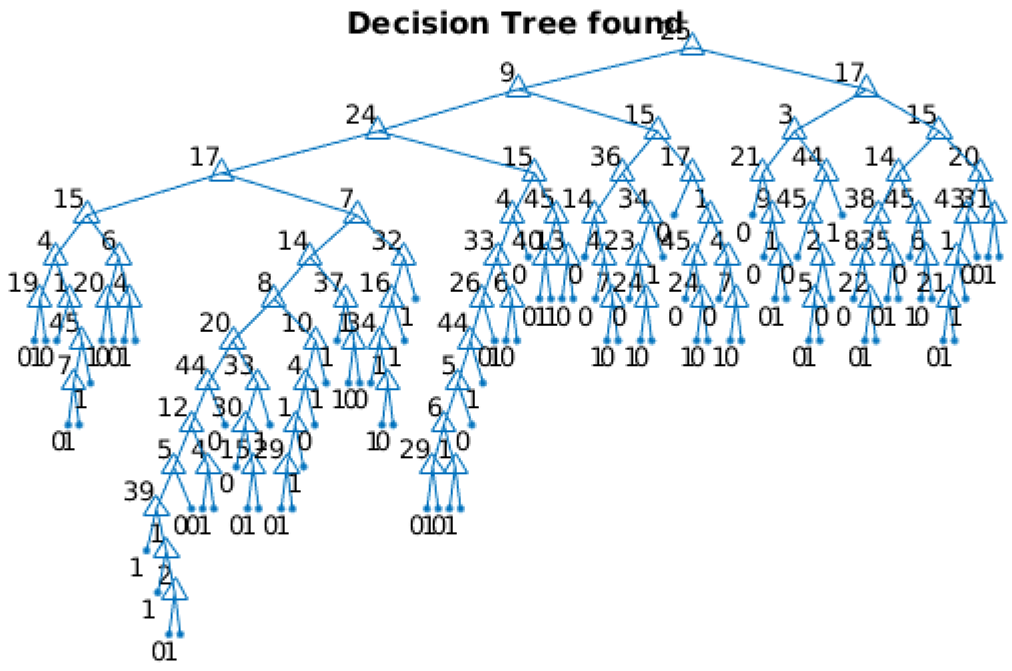
2.3 Emotion 3



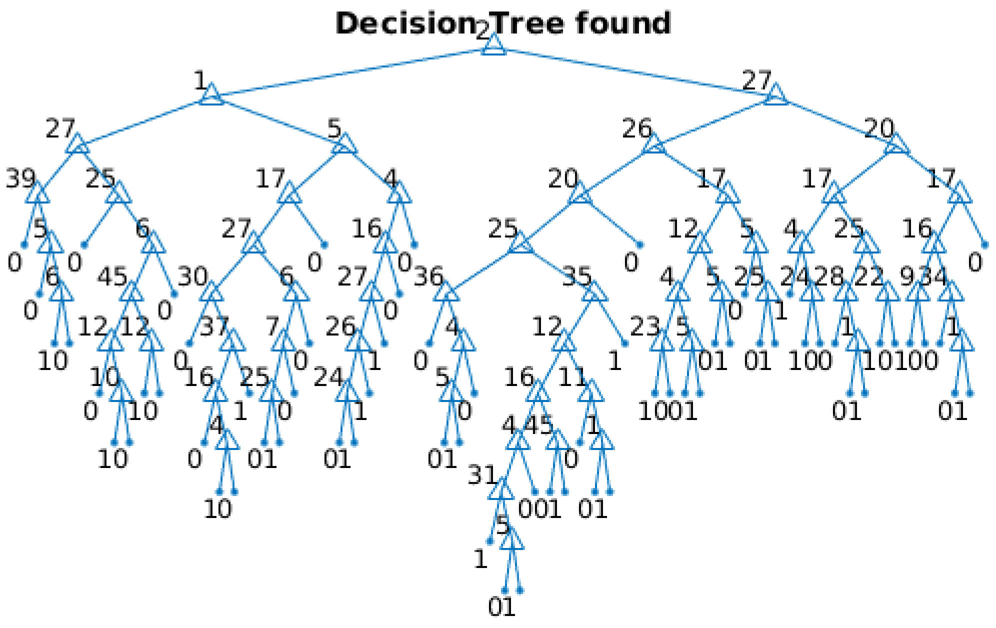
2.4 Emotion 4



2.5 Emotion 5



2.6 Emotion 6



3 Results of the Evaluation

3.1 For the Clean Data Set

3.1.1 The Total Confusion Matrix

		Predicted Class					
		1	2	3	4	5	6
Actual Class	1	93	11	4	7	13	4
	2	20	136	7	11	13	11
	3	5	5	83	1	7	18
	4	4	11	4	179	11	7
	5	23	11	12	12	70	4
	6	4	4	16	7	10	166

The classes 1-6 represent the 6 different emotions (anger, disgust, fear, happiness, sadness and surprise). Looking at the emotion 1 class as an example, the confusion matrix shows that of the 132 actual examples of emotion 1, the system predicted 93 correctly as emotion 1 (anger), 11 as emotion 2 (disgust), 4 as emotion 3 (fear), 7 as emotion 4 (happiness), 13 as emotion 5 (sadness) and 4 as emotion 6 (surprise). This means that there were 93 true positives (TPs) and 39 false negatives (FNs) for the emotion 1 class. There were 136, 83, 179 , 70 and 166 TPs for emotions 2, 3, 4, 5, 6 respectively.

3.1.2 Classification Rate

The average classification rate = 72.41%. This measures the number of correctly predicted examples divided by the total number of examples.

3.1.3 Precision Rate

The average precision rates for each of the 6 classes (in percentages):

Class					
1	2	3	4	5	6
62.42	76.40	65.87	82.49	56.45	79.05

3.1.4 Recall Rate

The average recall rates for each of the 6 classes (in percentages):

Class					
1	2	3	4	5	6
70.45	68.69	69.75	82.87	53.03	80.19

3.1.5 F1-measure

The F1-measure for each of the 6 classes (in percentages):

Class					
1	2	3	4	5	6
66.19	72.34	67.76	82.68	54.69	79.63

3.2 For the Noisy Data Set

3.2.1 The Total Confusion Matrix

	Predicted Class						
		1	2	3	4	5	6
Actual Class	1	21	12	17	12	17	9
	2	14	126	16	19	6	6
	3	11	21	97	26	15	17
	4	7	14	12	159	4	13
	5	19	10	6	10	55	10
	6	14	8	15	17	12	154

3.2.2 Classification Rate

The average classification rate = 61.14%

3.2.3 Precision Rate

The average precision rates for each of the 6 classes (in percentages):

Class					
1	2	3	4	5	6
24.42	69.57	59.51	64.43	50.46	73.68

3.2.4 Recall Rate

The average recall rates for each of the 6 classes (in percentages):

Class					
1	2	3	4	5	6
23.86	67.38	51.87	74.08	50.00	70.00

3.2.5 F1-measure

The F1-measure for each of the 6 classes (in percentages):

Class					
1	2	3	4	5	6
24.14	66.67	55.43	70.35	50.23	71.79

4 Questions

4.1 Noisy-Clean Datasets Question

There is a difference in performance when training and testing using a clean data set in comparison with a noisy data set. When looking at the classification performance measures above (confusion matrix, classification rate, recall and precision rates, F1-measure) for the clean and noisy data sets, we can see that the performance of the classifiers when using the clean data set is superior to the performance when using the noisy data set.

- The average classification rate is 11.27% higher using clean data set. This number may vary slightly each time the program is run (see 4.2).
- The precision rates using the clean data set are generally higher for each of the six classes.
- The recall rates using the clean data set are generally higher for each of the six classes.
- The F1-measures using the clean data set are generally higher for each of the six classes.

The clean data is considered to be completely accurate, whereas the noisy data contains some errors and incorrect information. The noisy data set used here could contain some incorrectly detected AUs and some AUs may be missing.

Since there may be some inccorrectly detected or missing AUs in the data, this would impact the performance of the learning algorithm. If the system is learning from data that contains several errors, it could increase the possibilty of the making incorrect predictions. This could lead to a smaller amount of true positives and larger amount of false negatives classified for each class. This is why the performance measures are generally worse when using the noisy data data set.

In essence, less accuracy in the data used for training will result in less accuracy in classification of examples, and this is evident in the results.

4.2 Ambiguity Question

During testing, the case in which multiple emotions are predicted for a set of attributes must be accounted for, so that only one emotion is given. Three methods for doing this were devised: weighted selection, random selection and recursion depth selection. The advantages and disadvantages, as well as the performance of each method will be evaluated. For each of the methods, a random emotion was chosen if the example did not give rise to a positive example for any of the trees; because of this, the performance of the decision trees varies marginally each time the program is run and new trees are created. For the clean data, 11.94% of examples could not be classified, whilst 16.98% could not be classified using the noisy data.

4.2.1 Weighted Selection

The first attempt at selecting one emotion (in the case that more than one was found) was to overwrite any previously found emotion each time another one was found for a set of attributes. For example, if the trees for emotions 1-6 were tested iteratively - starting at emotion 1 and ending at emotion 6 - and emotions 1, 2, 3 and 4 were found to be positive for a given set of attributes, emotion 4 would be chosen as the prediction. Similarly, if the trees for the aforementioned set of attributes were tested in the reverse order - from 6 to 1 - and the same emotions were found to be positive, emotion 1 would be chosen as the prediction. When the trees were tested using weighted selection, iterating through the trees in an ascending order, the predictions can be expected to be weighted towards emotion 6, and thus it is reasonable to expect that the trees would perform better, relatively, the closer they were to emotion 6.

4.2.2 Random Selection

Another possible method is to randomly select one of the found emotions. The effect this method would have on performance is unpredictable, and varies each time the program is run, even more so than either of the other two methods.

4.2.3 Recursion Depth Selection

The *testTrees* function is implemented using recursion: given an example, it recursively searches down each tree according to the value of the AU at each node, until a leaf node is found. Due to the fact that smaller trees generally respond better to unseen data than longer ones (see https://uk.mathworks.com/help/stats/classification-trees-and-regression-trees.html), which are prone to overfitting, it can then be reasonably assumed that the trees that require fewer recursive calls will have higher performance on unseen data, and ergo should be chosen over trees that require a greater number of calls.

4.2.4 Performance on the Clean and Noisy Data Sets

Below are the performance measures on the clean and noisy data set, using the the three different methods described above.

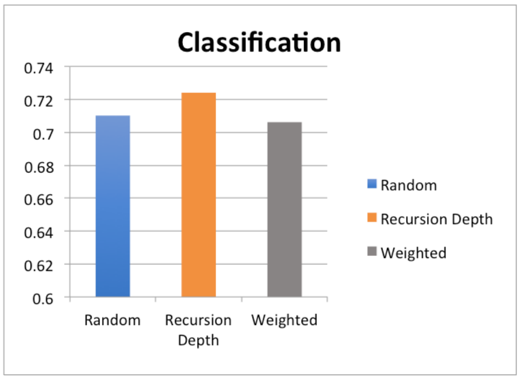


Figure 1: Classification rate (clean data set)

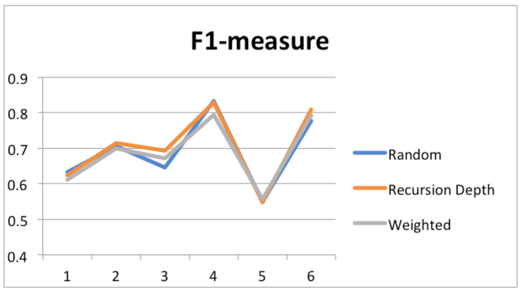


Figure 2: F1-measure (clean data set)

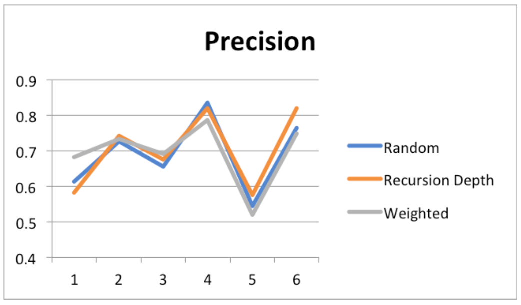


Figure 3: Precision rate (clean data set)

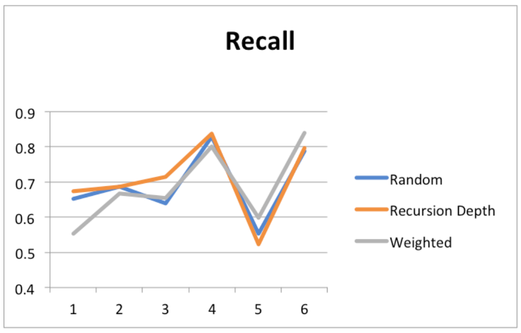


Figure 4: Recall rate (clean data set)

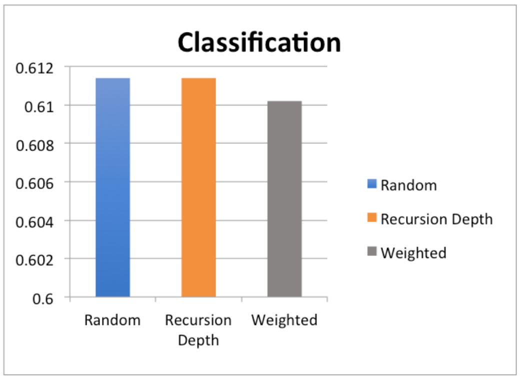


Figure 5: Classification rate (noisy data set)

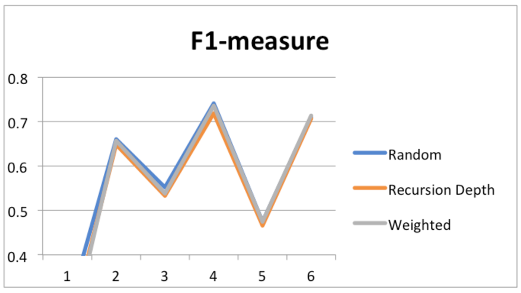


Figure 6: F1-measure (noisy data set)

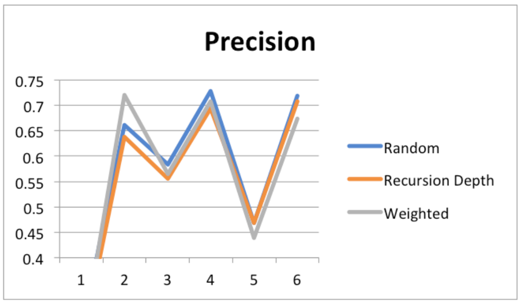


Figure 7: Precision rate (noisy data set)

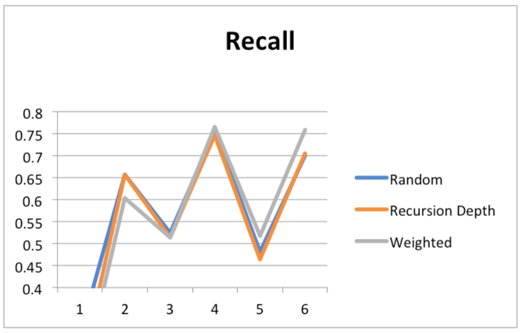


Figure 8: Recall rate (noisy data set)

4.3 Pruning Question

If a decision tree is too large, it is at risk of overfitting the data it has been trained on, meaning that it will not necessarily perform well on unseen data. Pruning is a technique that can be used to combat this, by removing subtrees and branches from a tree that do not aid as much in classifying examples, meaning that a smaller decision tree can be used without affecting performance.

The *classregtree* function constructs an unpruned decision trees with two branches from each node. The *test* function then checks how well the tree performs. In this case, two methods of testing the tree are used: resubstitution and 10-fold cross-validation. Resubsitution simply involves training and testing a tree on the exact same set of data, whilst 10-fold cross-validation - as is discussed in 1.2 - involves splitting the data into training and testing sets (9 parts training and 1 part testing, repeating the process 10 times, with a different testing set each time). Following this, the test results are displayed as a line graph using MATLAB's built in *plot* function, with the number of terminal (leaf) nodes of the tree on the x-axis, and the cost (the relative performance of the tree, as measured by testing it) on the y-axis.

4.3.1 Evaluation

For both the clean and noisy datasets, a similar pattern can be observed: for the trees tested using the resubstition method (the red line in the graphs below), the cost is inversely proportional to the number of terminal nodes. When the tree is unpruned, there is no cost, as it is trained and tested on the same data. Consequently there is a high chance that the tree overfits the data, and would perform poorly on unseen data. As the tree is pruned further, it overfits the data to a lesser degree, and its performance becomes more representative of what it would be on unseen data.

The change in cost is for trees trained using cross-validation (the blue line in the graphs below) is negligable for most levels of pruning. This is because cross-validation aims to simulate testing trees on unseen data (see 1.2, and will therefore have an inherently higher cost, as it does not overfit the data. There is however, a visible threshold at roughly 25 terminal nodes: pruning trees to contain fewer terminal nodes than this has a clear effect on the cost of the trees, as the tree becomes underfitted. This is also true for trees tested using resubstitution.

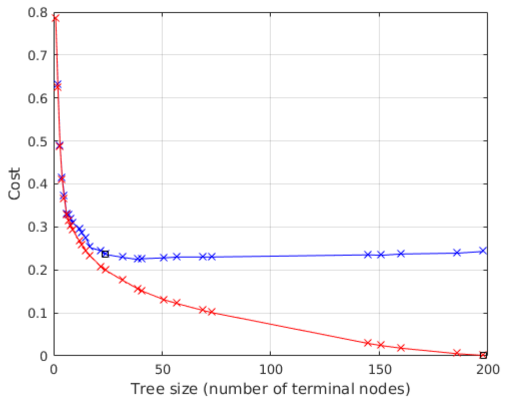


Figure 9: Pruning results on clean data.

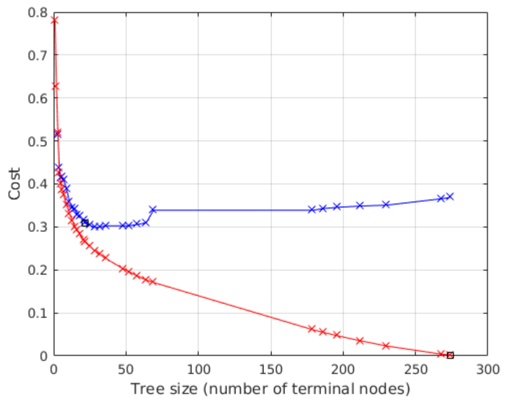


Figure 10: Pruning results on noisy data.