Assignment 2: Decision Trees Algorithm

Implementation details

Constructing Decision Trees

`DecisionTreeLearning` is implemented exactly as specified in the manual by the pseudo-function provided to us, including the relevant helper functions where specified. This includes `chooseBestDecisionAttribute`, which follows the calculations listed below in order to calculate the gain for each attribute, and return which attribute has the highest gain.

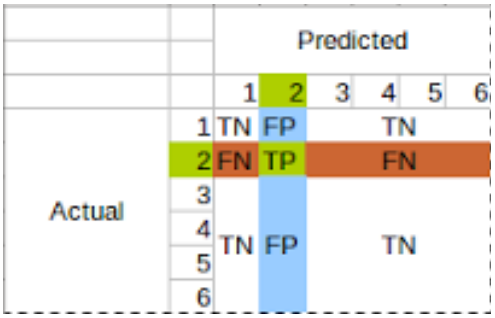
$$Gain(attribute) = I(p,n) - Remainder(attribute)$$
$$I(p,n) = -\frac{p}{p+n} log_2 (\frac{p}{p+n}) - \frac{n}{p+n} log_2 (\frac{n}{p+n})$$

When choosing the best decision attribute, it is possible that there would be multiple attributes which provide the same gain. In this case, we pick the attribute which has the smallest numerical value (as this is the one which the iteration will reach first).

Cross Validation and Evaluation

Our fold function is written such that it will generate an n x 2 matrix, where n is the number of folds. In each row, the first entry contains a cell array of predicted results and the second contains the actual results for each fold. The function calls `partition`, in order to split the provided data into an appropriate number of folds, and then `testTrees` to generate the predicted results from the trees which are created each fold. We implemented two different ways of combining trees to get our results which are detailed in the Answer to the Ambiguity Question section.

We then use our `generateConfusion` function in order to take the complete set of predictions and actual results to create a general confusion matrix containing all the information for each class over every fold. Given a generated matrix our `calculateStats` function calculates the number of true/false positive and true/false negative examples for each class following the strategy shown in Fig. 1.



Using these numbers we calculated the class-specific measures of recall rate, precision rate, f1 measure and classification rate following the provided formulae.

Fig. 1
Identifying true/false positives/negatives for a given class (in this case, class 2)

For more information on each function see Appendix A

Tree figures

The following Tree figures were trained on the entire clean data set. They are used to predict whether a given example is a member of the emotion classes 1=Anger, 2=Disgust, 3=Fear, 4=Happiness, 5=Sadness and 6=Surprise respectively.

Fig 2.1 : Anger

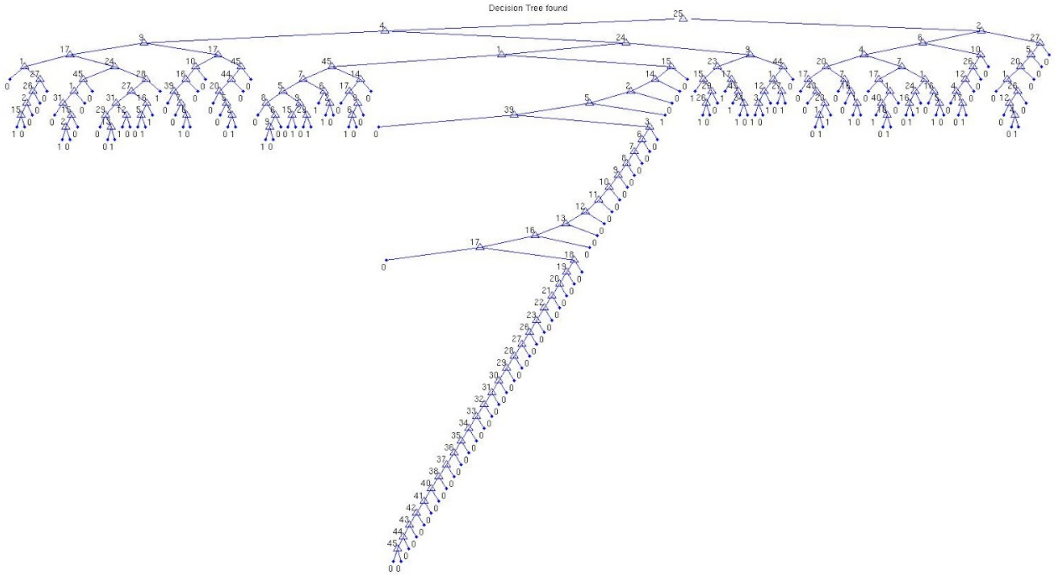


Fig. 2.2 : Disgust

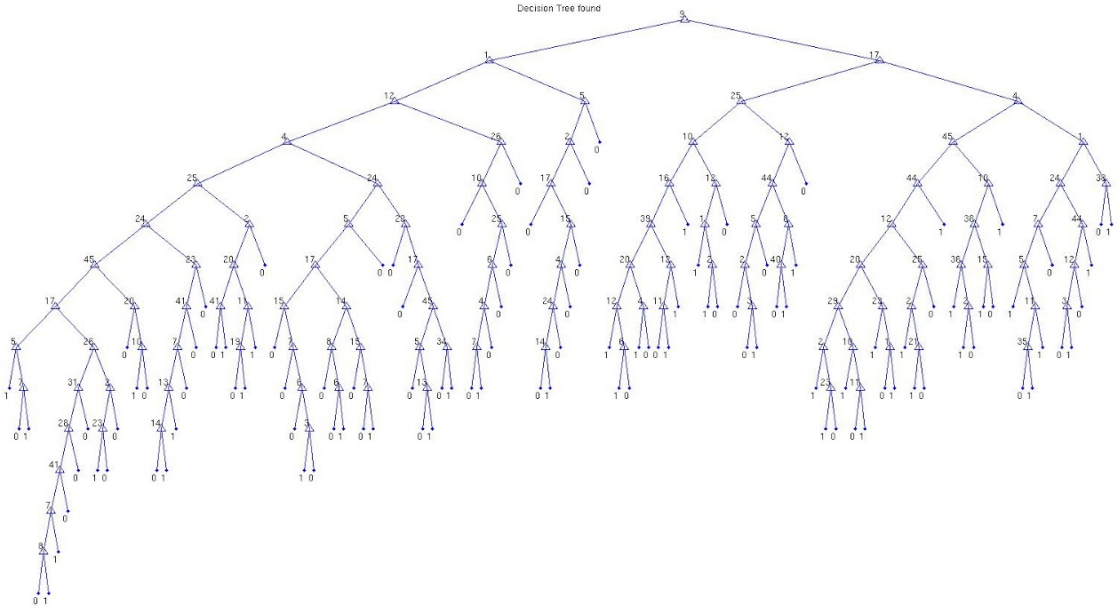


Fig. 2.3 : Fear

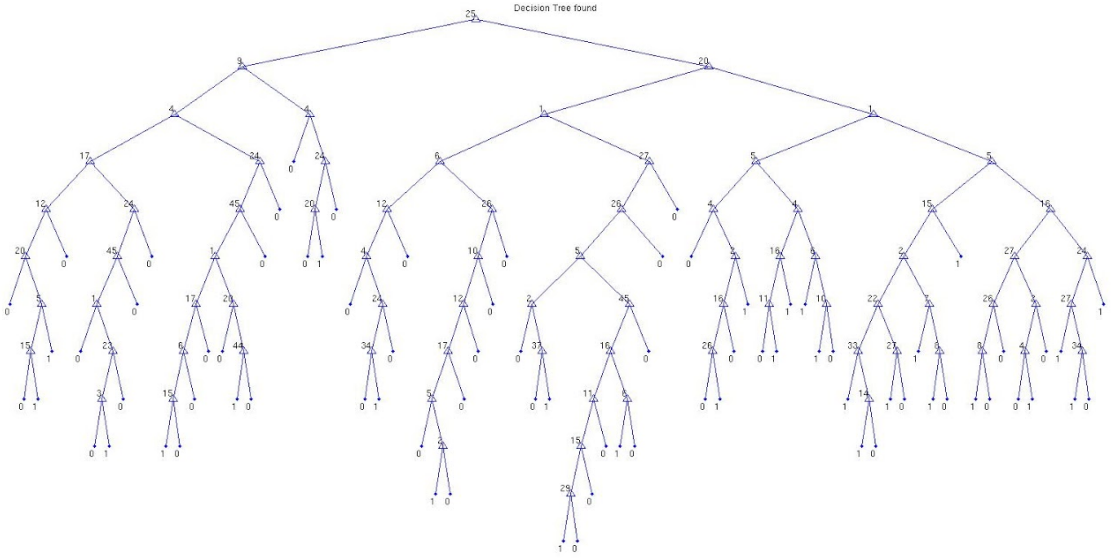


Fig 2.4 : Happiness

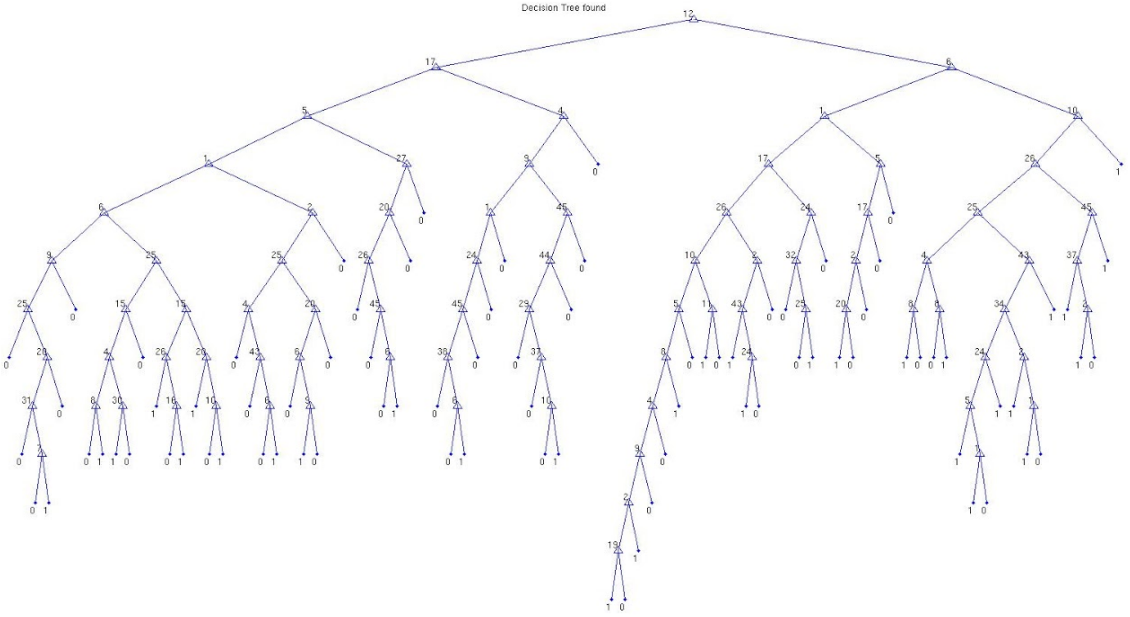


Fig 2.5 : Sadness

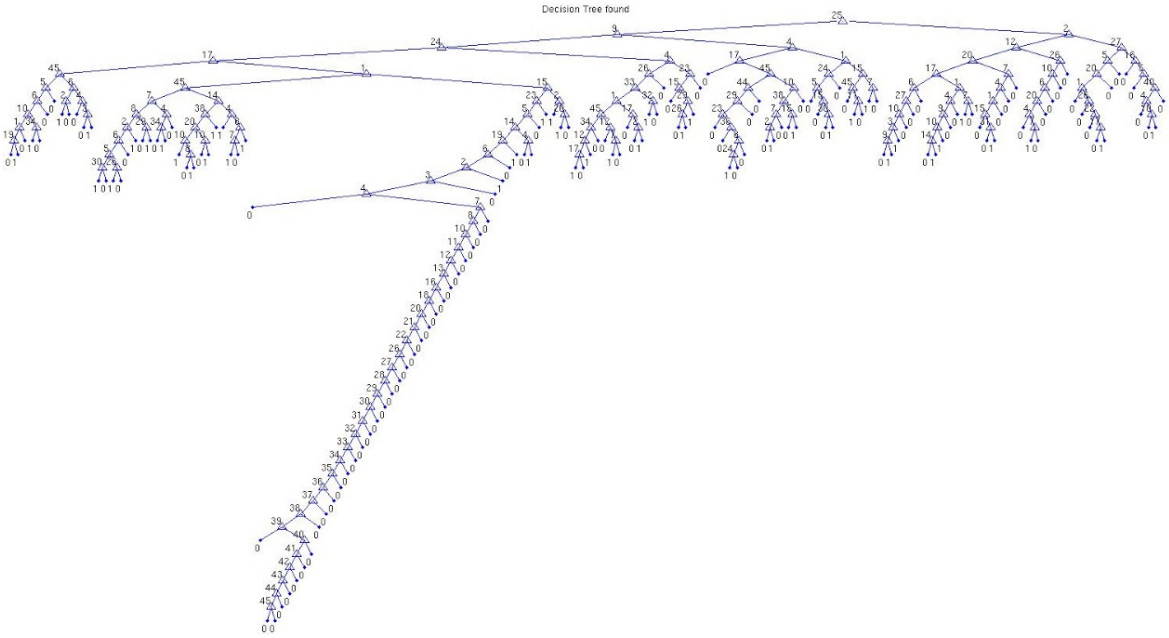
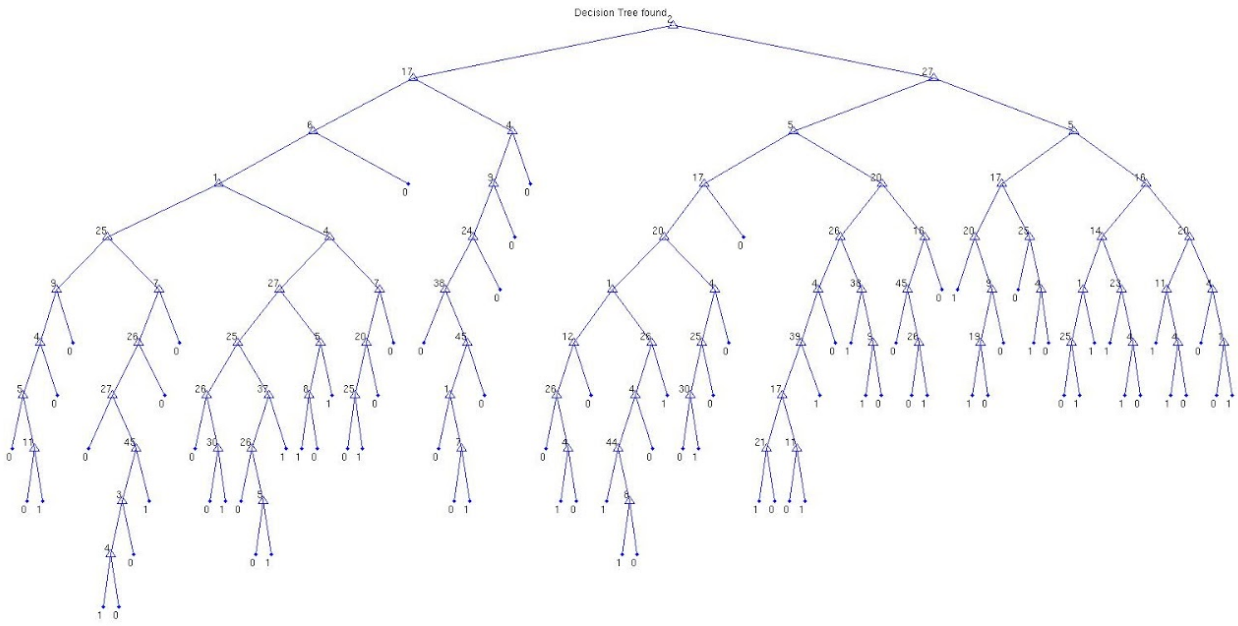


Fig 2.6 : Surprise



Evaluation

The following results show the performance of our decision tree learning algorithm on the complete clean and noisy data sets. These findings correspond to our *first* method of combining trees to create results. For results from the *second* method, and for comparisons between the two, see the Answer to the Ambiguity Question section (page 9).

Note: the following matrices are based on the raw data and are not normalised because the data set is not imbalanced; each class has roughly the same number of examples. Also, the displayed confusion matrices are for testTrees V1 (testTrees V2 matrices omitted for clarity).

Clean Data

Confusion Matrix	Anger	Disgust	Fear	Happiness	Sadness	Surprise
Anger	101	6	3	7	12	4
Disgust	21	152	0	7	14	5
Fear	21	1	77	0	3	18
Happiness	8	3	1	192	9	5
Sadness	35	10	0	3	80	5
Surprise	17	1	7	2	2	181

`testTrees` V1	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Average
Recall Rate	75.9%	76.4%	64.2%	88.1%	60.2%	86.2%	75.2%
Precision Rate	49.8%	87.9%	87.5%	91.0%	66.7%	83.0%	77.6%
F1-measure	60.1%	81.7%	74.0%	89.5%	63.2%	84.6%	75.5%
Classification rate	0.868	0.933	0.947	0.956	0.908	0.935	0.924

Noisy Data

Confusion Matrix	Anger	Disgust	Fear	Happiness	Sadness	Surprise
Anger	45	8	13	5	14	6
Disgust	22	124	24	11	9	2
Fear	34	7	104	20	8	16
Happiness	27	7	7	153	8	10
Sadness	36	4	8	3	50	9
Surprise	18	1	13	5	6	178

`testTrees` V1	Anger	Disgust	Fear	Happiness	Sadness	Surprise	Average
Recall Rate	50.6%	65.6%	55.0%	72.2%	45.5%	80.5%	61.6%
Precision Rate	24.7%	82.1%	62.7%	77.7%	53.8%	80.5%	63.6%
F1-measure	33.2%	72.9%	58.6%	74.8%	49.3%	80.5%	61.6%
Classification rate	0.821	0.909	0.855	0.898	0.898	0.915	0.883

Analysis of the cross validation experiments

On both the clean and noisy datasets, the emotions which have the highest F1 measures are disgust (clean: 81.7%, noisy: 72.9%) , happiness (clean: 89.5%, noisy: 74.8%) and surprise (clean: 84.6%, noisy: 80.5%). This shows that our trees are much better at recognising these emotions (regardless of noise in the dataset) than the other 3.

We can also see that the average classification rate is surprisingly high, at 0.88, for the noisy dataset, although it is consistently higher or equal for every class in the clean dataset (roughly 5% higher on average).

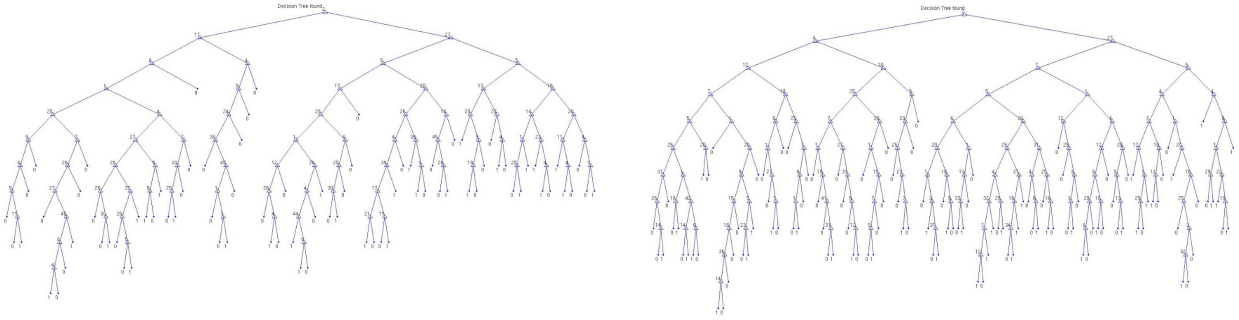
The results show that the precision rate for anger is extremely low at 49.8% for the clean data and just 24.7% for the noisy data set. This is caused by our `testTrees` function, which works by assuming that the example is of class 1 (anger), and then checks trees 2-6 for a match. If it fails to find a match in any of these trees then it defaults to class 1, resulting in a high number of false positives for this class in both datasets.

The matrices show that some pairs of emotions were easily confused. For example, surprise and fear were sometimes mistaken for one another by our algorithm. In the clean data set, 18 examples of fear were classified as surprise and 7 examples of surprise were classified as fear, and in the noisy set 16 examples of fear were classified as surprise and 13 examples of surprise were classified as fear. This is probably due to a large overlap in the AUs presented when showing fear or surprise. The same could be said of sadness and anger because, even though the default class is anger, 12 examples of anger in the clean set and 14 examples in the noisy set were misclassified as sadness by our algorithm.

Questions

Answer to Noisy-Clean Datasets Question

There is a clear difference between the clean and noisy datasets, as evidenced by the ~15% difference in the average values for the various statistical measures. This loss of performance is caused by the noisy dataset introducing a lot more unnecessary or incorrect branches when the trees are created. Evaluation on these trees will then lead to a lot of erroneous data, as some of the branches will be either too specific (fail to match on a positive example, leading to a false negative) or be incorrect (match on a negative example due to noise in the training data). This difference can be demonstrated in the trees for the class Surprise as shown below.



Generated trees for Surprise trained on clean data (left) and noisy data (right)

For specific classes, some are very seriously affected by the clean/noisy dataset switch. Anger is the worst affected, dropping between 30 - 40% on the precision, recall and F1 measures when moving from clean to noisy data. This is likely due to anger being confused with other emotions, particularly sadness, even on the clean data set. This implies that for the data we have there is not a lot of difference between the AUs presented in anger and fear, happiness and sadness, and so when noise is introduced the possibility of recall and precision rates dropping for anger is very high.

The 2 emotions which showed the most resilience between the clean and noisy datasets were disgust (~2% drop on average) and surprise (also ~2% drop on average). This indicates that these 2 emotions are very distinct in terms of the attributes provided, to the extent that even the introduction of noise had little effect on their results.

Finally, fear, happiness and sadness both fell by around 10-15%. Considering the large drop for anger, this indicates that there is a large amount of confusion between these emotions and anger, and also an amount of confusion between each of these classes as well.

Answer to Ambiguity Question

Our first approach for labelling the set of examples has a simplistic nature. Within the `testTree` method, we initialized the predictions for each example with the first emotion, anger. Then, for each example we iterated through the emotions and we tried to find a match; we then accepted the last match for an example and we labelled it accordingly. This approach has the disadvantage that some of the examples which are labelled as anger may not match that emotion; another disadvantage is that in case where multiple emotions are matched for one example, we select an arbitrary one (the one with the highest index). However, the method is simple and therefore represents a good first approach.

We expanded upon this in our second approach, where in case multiple emotions were matched, we selected the one for which we evaluated more attributes in order to find a match; we chose this option because taking into account more attributes could indicate a better solution. When no emotion was matched, we also chose the one for which we evaluated more attributes because this indicates a better match. The advantage of this approach is that in case no match was found or multiple matches were found, it tries to search the match which was the most 'carefully considered'.

`testTrees` V1/V2	**Anger**	**Disgust**	**Fear**	**Happiness**	**Sadness**	**Surprise**	**Average**
Recall Rate	75.9% 71.4%	76.4% 80.9%	64.2% 74.2%	88.1% 83.9%	60.2% 64.7%	86.2% 80.5%	75.2% 75.9%
Precision Rate	49.8% 74.8%	87.9% 75.9%	87.5% 73.6%	91.0% 83.6%	66.7% 60.6%	83.0% 88.0%	77.6% 76.1%
F1-measure	60.1% 73.1%	81.7% 78.3%	74.0% 73.9%	89.5% 83.8%	63.2% 62.5%	84.6% 84.1%	75.5% 75.9%
Classification rate	0.868 0.931	0.933 0.912	0.947 0.938	0.956 0.930	0.908 0.898	0.935 0.937	0.924 0.924

Results for both testTrees approaches on clean data

`testTrees` V1/V2	**Anger**	**Disgust**	**Fear**	**Happiness**	**Sadness**	**Surprise**	**Average**
Recall Rate	50.6% 43.8%	65.6% 78.3%	55.0% 61.9%	72.2% 72.2%	45.5% 46.3%	80.5% 78.3%	61.6% 63.5%
Precision Rate	24.7% 33.9%	82.1% 75.5%	62.7% 60.0%	77.7% 76.5%	53.8% 49.0%	80.5% 86.5%	63.6% 63.6%
F1-measure	33.2% 38.2%	72.9% 76.9%	58.6% 60.9%	74.8% 74.3%	49.3% 47.7%	80.5% 82.2%	61.6% 63.4%
Classification rate	0.821 0.875	0.909 0.912	0.855 0.852	0.898 0.895	0.898 0.889	0.915 0.926	0.883 0.891

Results for both testTrees approaches on noisy data

On both the clean and the noisy datasets, the precision rate for anger was much larger for the second approach (74.8% vs 49.8% on clean data, 33.9% vs 24.7% on noisy data), and the recall rate was smaller because many examples which were mapped to anger in the first approach (when no other match was found) were matched in the second approach to the emotion which fitted them better.

On average, we saw no overall change in the clean dataset; the classification rate for both approaches was identical, and the differences for each measures' average value was very low (~1% at most). On the noisy dataset however there was a much more obvious difference between the two approaches, resulting in the 2nd, branch-length based approach giving better results on average for all measures. However, for both datasets the F1-measure was slightly larger (0.4% for clean data, 1.8% for noisy data) for the second classifier, showing that this classifier has a higher quality.

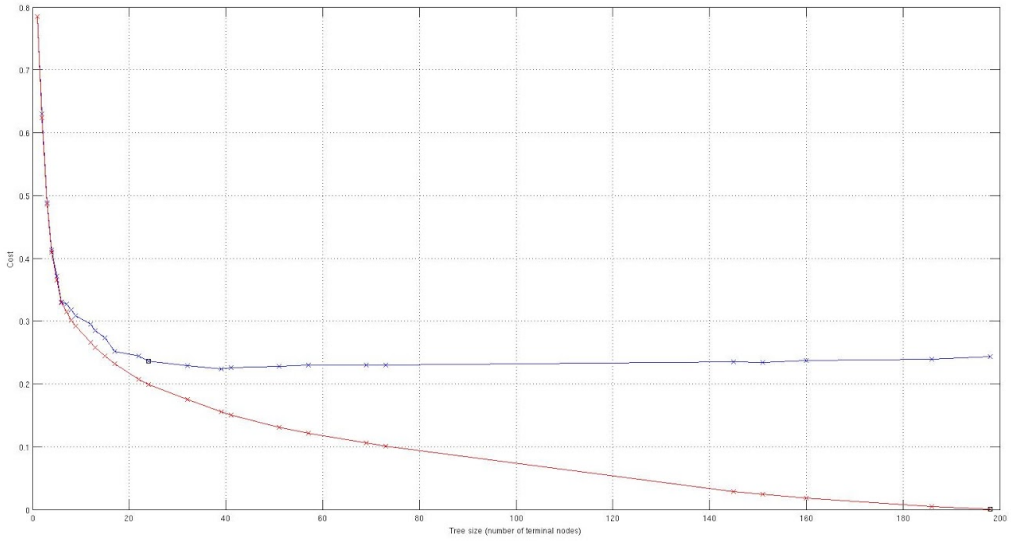
Answer to Pruning Question

The `pruning_example` function will generate a graph, showing the result of pruning a decision tree for the provided data to varying degrees.

`pruning_example` first generates a decision tree for the given data using the MATLAB function `classregtree`. It then prunes this tree to several different sizes and runs 10-fold cross validation on each resulting pruned tree. The graph displayed at the end shows the error rate (Cost) from this evaluation at all the tree sizes that it has pruned the tree to. The graph includes results for both the training data (using MATLAB's `test(tree,'resubstitution'))` and evaluation data (using `test(tree,'cross',x,y)`) for the 10-fold validation.

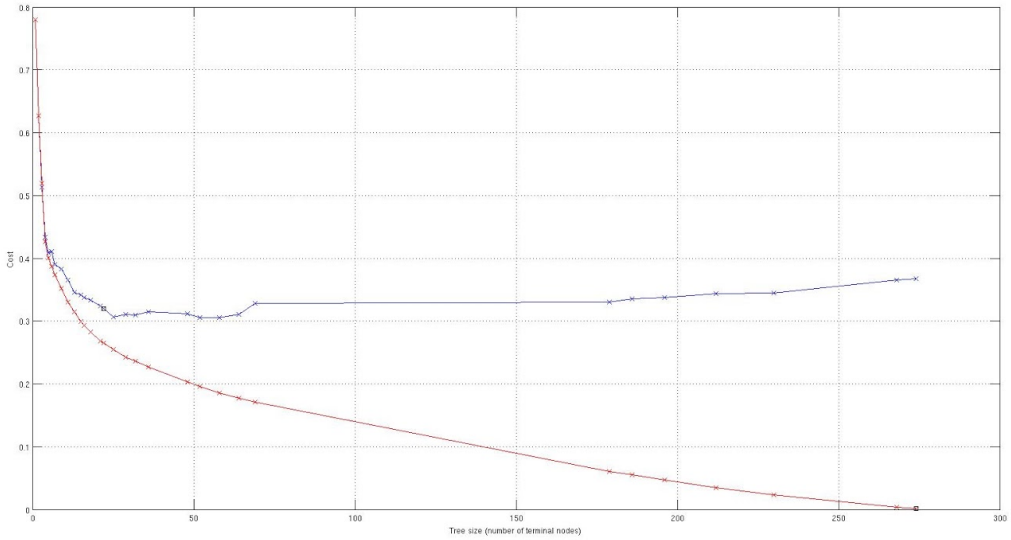
The following figures show the results of running `pruning_example` on the complete clean and noisy data sets.

Fig 3.1



Result of pruning_example on the clean data set

Fig 3.2



Result of pruning_example on the noisy dataset

These curves represent the error rate obtained from validating the resulting pruned trees with both the training data (the red curve) and unseen testing data (the blue curve). Both curves for both data sets show a sharp initial drop in error rate from tree size of 0 to 20 nodes. This is expected as adding a node to a very small decision tree will greatly improve its performance and prevent it from over-generalising. What we observe in both the clean and noisy data sets is a continual reduction in the error rate on training data, reaching 0 for a large number of nodes (around 200 for clean and 270 for noisy data). For validation data the error rate has a minimum value > 0 (~0.22 for clean and ~0.30 for noisy data).

For the clean dataset we can observe the testing data curve (blue) levelling off at a size of around 40 nodes, after which increasing the number of nodes has very little effect on the error rate. This would lead us to conclude that the ideal size to prune a decision tree to for our clean dataset would be 40 nodes. As the error rate does not increase, merely remains constant, this pruning would help only in reducing storage requirements and the time taken to traverse the tree.

We see a different curve in the noisy dataset, however. The testing data curve for this dataset reaches a minimum at around 25 nodes, slightly less than for the clean dataset. However, increasing the number of nodes past this point actually increases the error rate. This indicates that the increased number of nodes results in *overfitting* of the tree to the noisy data, resulting in more errors occurring when the tree is allowed to grow beyond a particular size. The result is that in this case picking 25 nodes not only has the benefits as mentioned for the clean dataset, but also reduces the error rate.

The overfitting is a result of the noise in the dataset; past a certain point it is impossible to differentiate between different emotions due to the noise and so the extra attributes introduced are based solely on the noise. This will reduce the performance of the tree, as it will then contain nodes which are irrelevant.

Appendix A: Function Listing

`decisionTreeLearning` was implemented to exactly follow the pseudo-function DECISION-TREE-LEARNING provided in the CBC manual, with appropriate helper functions.

`chooseBestDecisionAttribute` follows the description provided in the manual, going through every valid attribute and returning the one which provides the maximum gain based on the provided formula

`calcI` simply follows the given formula to calculate entropy

`countMember` returns the number of times a particular attribute (`pos`) is of the given value (`elem`).

`getNumExamples` returns the number of times that the given attribute (`index`) occurs in `examples` AND the `binary_target` for that example matches the given target (`exm`)

`nFoldCrossValidation` returns an nx2 matrix (n being the number of folds) with the predictions for each fold in the first column and the actual values in the second column. Calls its local function `partition` and uses the returned indices to partition the data for each fold. The function then calls its other helper `testTrees` to generate predictions for the partitioned data.

`partition` calculates and returns the indices of the partitioning of the data for the current fold. It does so by dividing the size of the data by the number of folds and rounding, taking into account the fractions of examples rounded off so all data is always utilised.

`testTrees` returns the predictions for the 6 trees passed in, assuming that the example matches tree 1 then going through all the other trees in turn, looking for a match. If there are multiple matches then it will return the last tree checked (i.e. the class with the highest number).

`testTrees2` returns the predictions for the 6 trees passed in, where the prediction is the tree that was matched. If there is a conflict, it returns the tree which had the longest branch. Also, if no tree matches then it agian returns the one with the longest branch, as this is the tree which matched the most attributes.

`generateConfusion` generates the confusion matrix for the given set of predictions (`pred`) and actual answers (`y`).

`totalConfusion` takes in the result of `nFoldCrossValidation` and returns the summation of the confusion matrix for each fold.

`calculateStats` works out the various statistics (recall and precision rates, F1 measure and classification rate) for each class, as well as the average statistics for the entire set of classes.