CO395 Machine Learning
CBC #1
Decision Trees

Group 1

Yong Wen Chua, ywc110

Thomas Morrison, tm1810

Marcin Baginski, mgb10

Marcin Kadziela, mk4910

Contents

Implementation Details

The decision tree learning algorithm, along with the subsequent classification and statistics generating functionality are implemented as a set of functions. A flowchart of the code flow can be found in the Code Flowchart section.

At its core, the decision tree is defined by a series of nodes, which are produced by the factory function `make_node`. The structure of the nodes are defined by the coursework specifications.

One tree is built from each of the six possible emotions. Before the trees can be built, the targets for the training data has to be transformed into binary targets by a function `transform_targets` (i.e. `1` or `0` for that particular emotion). The decision tree is then built. Because of the recursive nature of the decision tree algorithm, there needs to be a way to keep track of the attributes that has been used higher up in the tree. This is done with a binary row vector that keeps track of whether a particular attribute has been used. Initially, this vector is a 1×45 vector of ones (`ones(1,45)`). The "best attribute" chosen for building the tree is first computed by calculating the entropy, and subsequently the information gain of the attributes. The attribute with the highest information gain is then chosen to be used as a node. The value corresponding to the attribute in the tracking vector is then set to `0` to indicate that we are no longer concerned with this particular argument and the the two kids of the node are then recursively built.

Using the trees that are built, additional examples can then be classified. Classification takes place in two steps. Firstly, the examples are classified binarily to see if they are positive or negative examples of each tree. At the same time, the depth of the tree at which a decision was made is also tracked, for tie-breaking purposes. Then, the examples are assigned the respective emotions based on the binary classifications. In the case of ties when more than one emotions are deduced positively, or when all emotions are deduced negatively, three strategies of breaking ties are implemented.

The first strategy involves a random selection from all the emotions that were deduced positively, or any random emotion if none of them were deduced positively. Due to the random nature, the prediction will vary when run consecutively for the same set of examples. The second strategy involves choosing the emotion that was decided at the lowest depth of the decision tree. Finally, the third strategy will choose the emotion that was decided at the highest depth of the decision tree. The second and third strategies uses the min and max functions of Matlab. If more than one of the values are equal, min and max will always choose the first instance. Thus, the second and third strategies will always predict the same results for the same set of examples.

Ten-fold-cross validation is implemented as a generic n-fold function that can take any arbitrary number of folds. The predicted results from the examples that were not used in a fold are concatenated at the end and returned from the function.

Throughout the implementation, checks are implemented in place in order to ensure that the code functions properly. For example, there are assertions in place to check that the size of the input is equal to the size of the output. This is done in place of more comprehensive unit and integrations tests.

Tree Figures

Emotion 1 - Anger

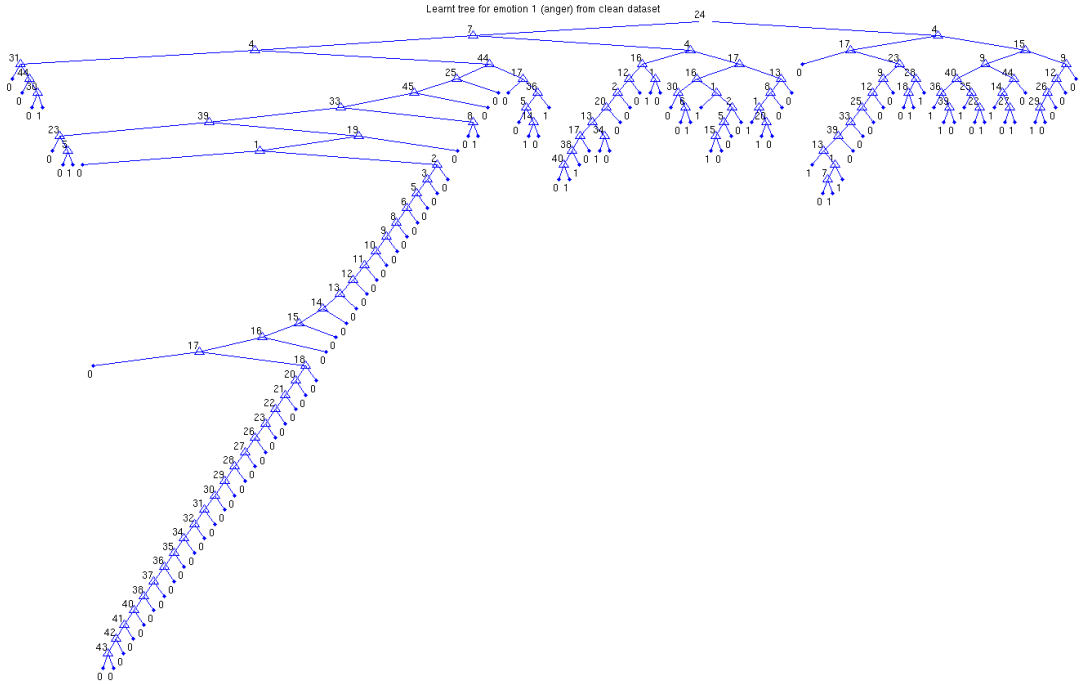

Figure 1: Trained decsion tree on the clean dataset for emotion 1 (anger)

Emotion 2 - Disgust

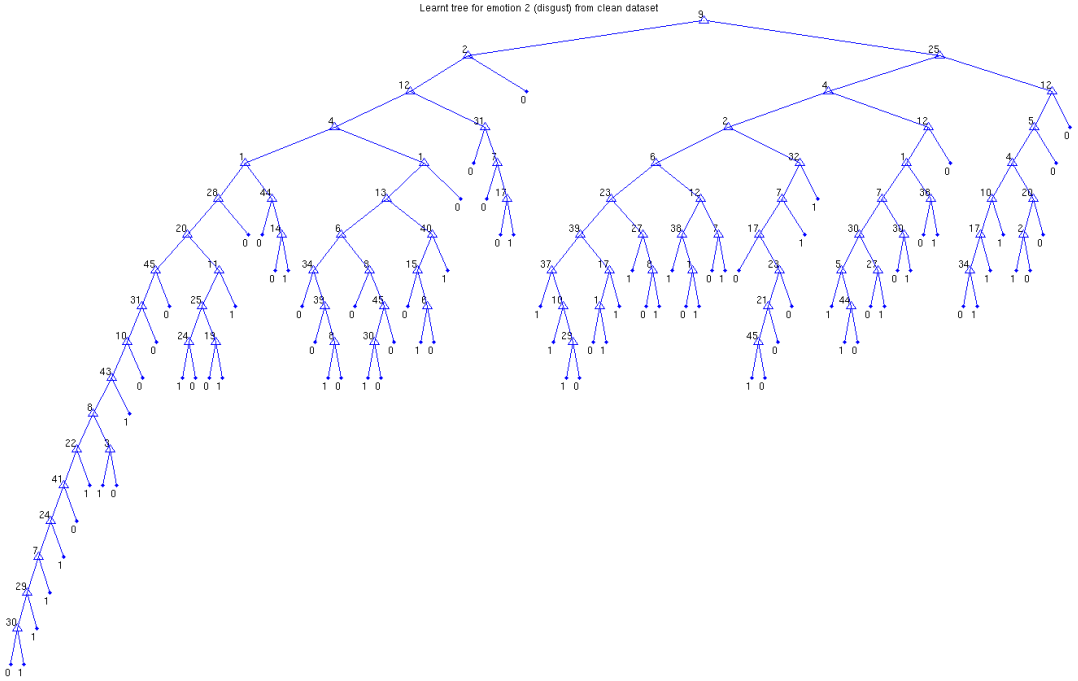

Figure 2: Trained decsion tree on the clean dataset for emotion 2 (disgust)

Emotion 3 - Fear

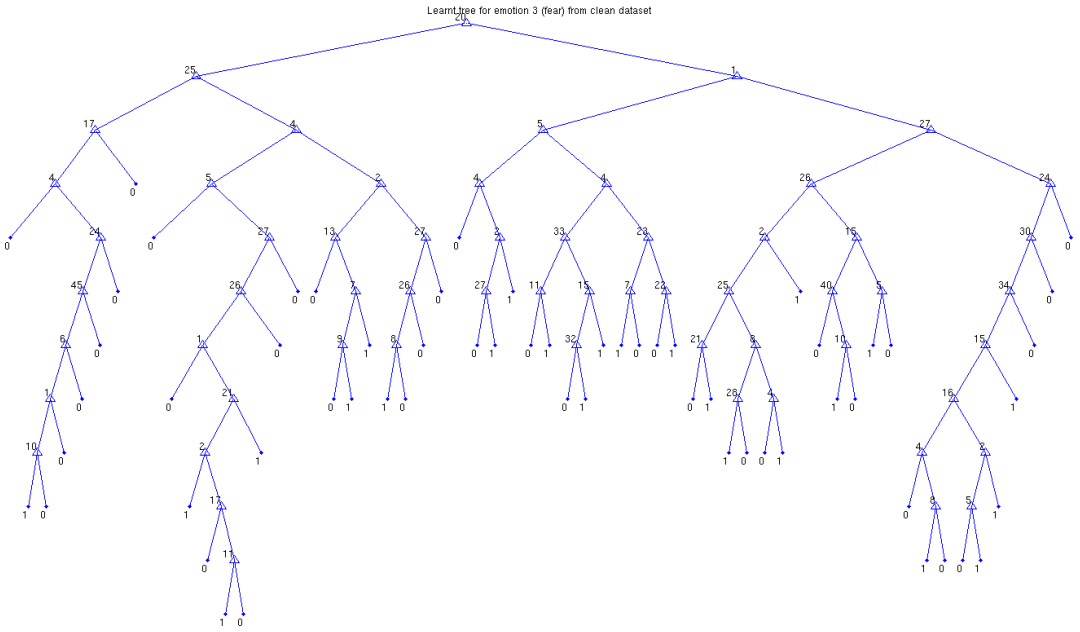

Figure 3: Trained decsion tree on the clean dataset for emotion 3 (fear)

Emotion 4 - Happiness

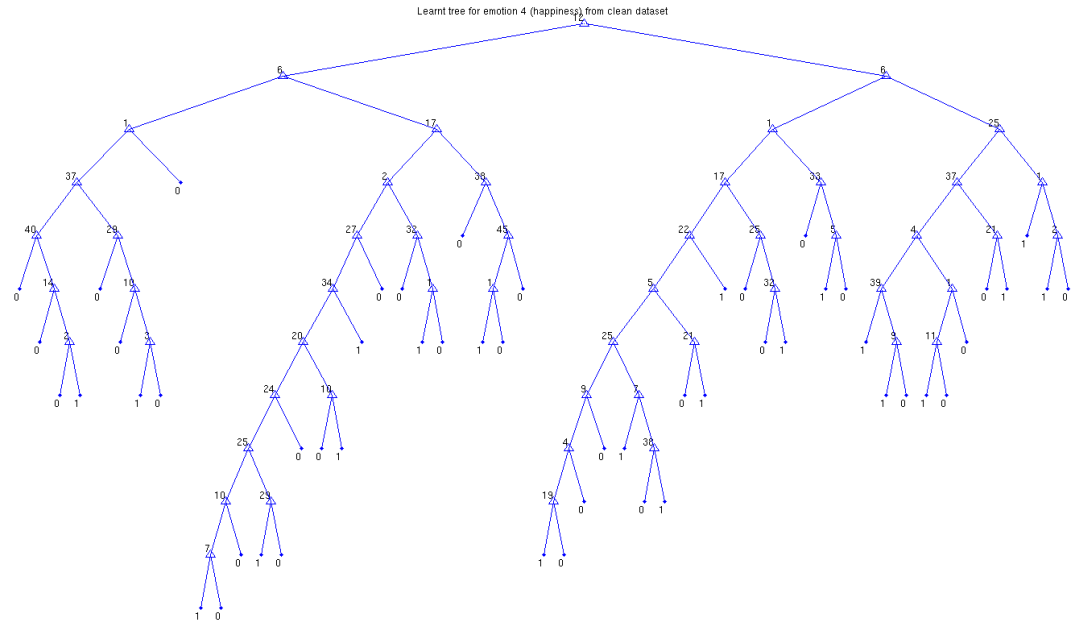

Figure 4: Trained decsion tree on the clean dataset for emotion 4 (happiness)

Emotion 5 - Sadness

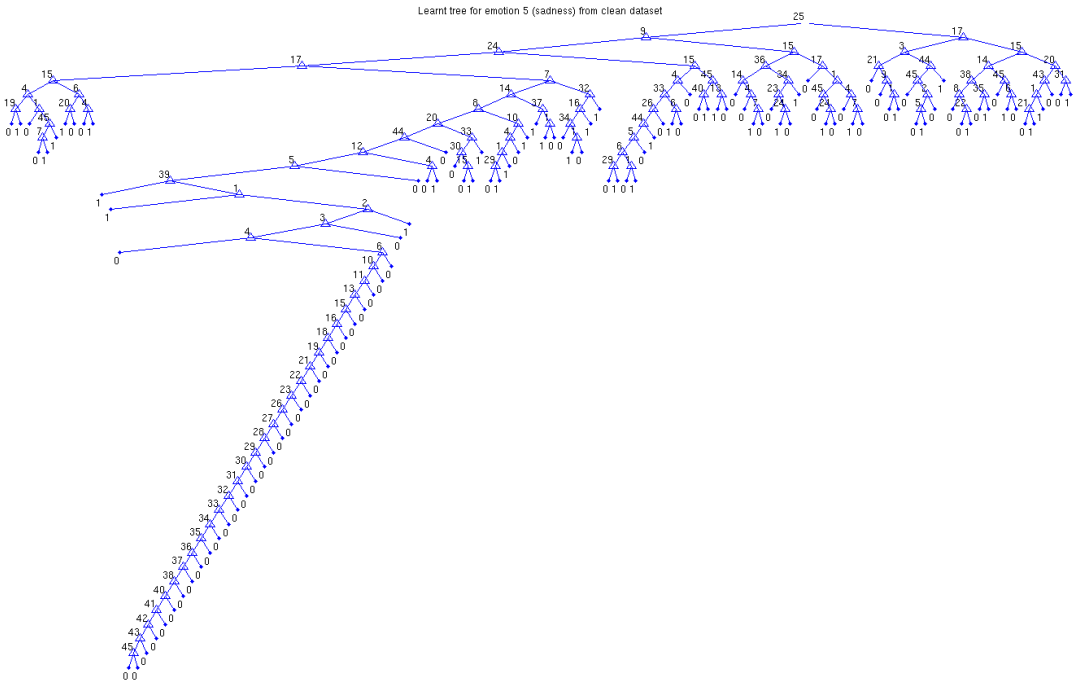

Figure 5: Trained decsion tree on the clean dataset for emotion 5 (sadness)

Emotion 6 - Surprise

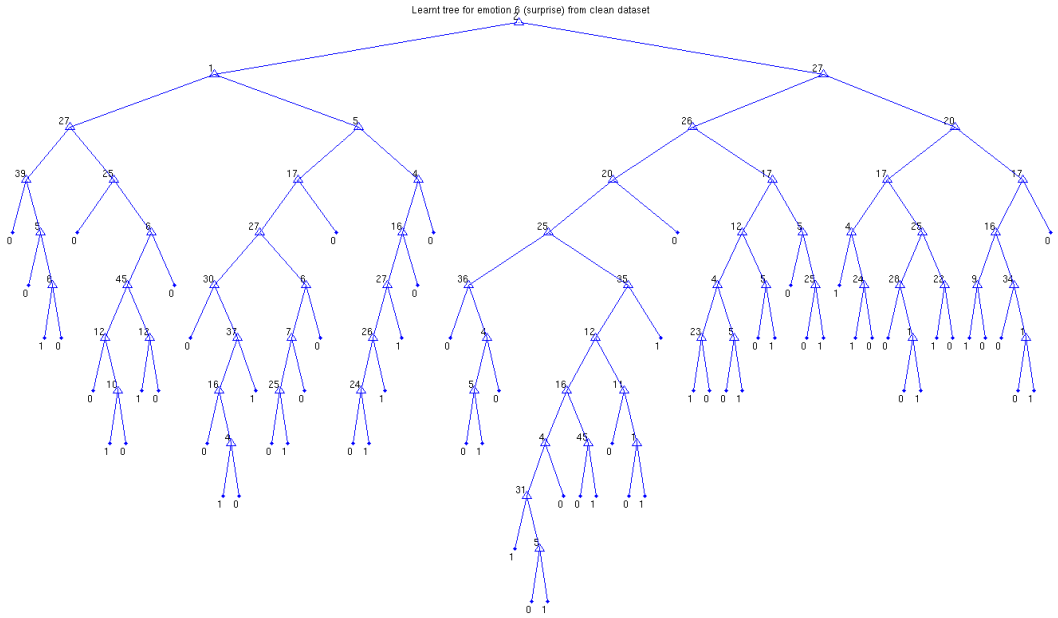

Figure 6: Trained decsion tree on the clean dataset for emotion 6 (surprise)

Evaluation

The results presented below are for a random assignment of conflicted classifications. Evaluation of other strategies is presented in the next section.

Clean dataset

Confusion matrix

		Predicted class					
		1	2	3	4	5	6
Actual class	1	**85**	15	4	9	17	2
	2	17	**137**	5	12	15	12
	3	8	7	**80**	0	10	14
	4	4	12	8	**177**	9	6
	5	15	10	11	11	**75**	10
	6	3	10	11	6	11	**166**

Table 1: Confusion Matrix for the *clean* dataset (Strategy 1 - see next section)

Recall, precision and F_1 measure

		Recall	Precision	F_1
Actual class	1	64%	64%	64%
	2	69%	72%	70%
	3	67%	67%	67%
	4	82%	82%	82%
	5	57%	55%	56%
	6	80%	79%	80%

Table 2: Recall, precision and F_1 measure for the *clean* dataset (Strategy 1 - see next section)

Average classification rate

$$C = \frac{720}{1004} = 71.7\%$$

Figure 7: Classification rate for the *clean* dataset (Strategy 1 - see next section)

Noisy dataset

Confusion matrix

		Predicted class					
		1	2	3	4	5	6
Actual class	1	**25**	11	16	5	22	9
	2	12	**127**	13	19	8	8
	3	17	13	**104**	19	14	20
	4	11	11	13	**154**	6	14
	5	17	7	7	12	**56**	11
	6	10	14	21	9	11	**155**

Table 3: Confusion Matrix for the *noisy* dataset (Strategy 1 - see next section)

Recall, precision and F_1 measure

		Recall	Precision	F_1
Actual class	1	28%	27%	28%
	2	68%	69%	69%
	3	56%	60%	58%
	4	74%	71%	72%
	5	51%	48%	49%
	6	70%	71%	71%

Table 4: Recall, precision and F_1 measure for the *noisy* dataset (Strategy 1 - see next section)

Average classification rate

$$C = \frac{621}{1004} = 61.9\%$$

Figure 8: Classification rate for the *noisy* dataset (Strategy 1 - see next section)

Discussion of results

The discussion of results can be found in the next section Questions/Noisy-Clean Datasets.

Questions

Noisy-Clean Datasets

Unsuprisingly, the trees perform better on the *clean* dataset, for which the proportion of misclassified examples (the error rate) is $E_C = \frac{284}{1004} = 28.3\%$. On the other hand, the error rate for the *noisy* dataset is $E_N = \frac{383}{1004} = 38.1\%$. This is expected, since in case of the *noisy* dataset the trees have been trained on data which itself contains classification errors. Since we trained the trees on an unreliable dataset, we cannot expect them to perform better than the trees trained and tested on an error-free data.

Another observation is that the trees perform exceptionally well on some emotions in both datasets (e.g. 4 and 6) while they fall short of expected performance for other emotions (e.g. 5 in the *clean* dataset or 1, 3, 5 in the *noisy* dataset). There is a particularly interesting loss of performance in the *noisy* dataset for emotion 1 (anger). This might be caused by the fact, that there has been much noise in the training set for this emotion, and the trees might have been incorrectly trained.

Ambiguity

Description of methods

We tried three methods for assignment of a single value to examples which have been positively classified by more than one tree:

1. Assign a *random* class from the set of predicted classes
2. Assign a class which has been returned at a *smallest depth*
3. Assign a class which has been returned at a *greatest depth*

Advantages of the first method include the fact that it is the simplest and easiest to implement. Additionally, it serves as a nice benchmark for the other methods. Comparison of the precision, recall, F_1 measure and the classification rate of other methods to the one which just picks a random class from the conflicted ones makes it possible to determine whether their performance is satisfactory. Obviously, the main disadvantage is that the method does not use any clever heuristic in order to classify the examples. Evaluation of Strategy 1 has been presented in the previous section.

The intuition behind the second method is that the smaller number of steps a tree needs to take in order to classify an emotion, the more "sure" it should be of the emotion being accurately classified. On the other hand, should that intuition prove to be incorrect during evaluation, we also wanted to see what would be the performance of an exact opposite algorithm, which is the third method outlined above. Both methods require us to store the depth of the returned solution, in addition to the classification results of the 6 trees (in a $N \times 6$ matrix). This slightly increases the memory requirements in comparison to Strategy 1, however the time and space complexity stays the same.

Evaluation of Strategy 2 on clean dataset

		Predicted class					
		1	2	3	4	5	6
Actual class	1	**70**	12	21	12	8	9
	2	30	**122**	26	8	5	7
	3	6	16	**71**	9	7	10
	4	17	20	5	**164**	5	5
	5	15	15	15	20	**62**	15
	6	3	28	9	8	2	**157**

Table 5: Confusion Matrix for the *clean* dataset (Strategy 2)

		Recall	Precision	F_1
Actual class	1	53%	50%	51%
	2	62%	57%	59%
	3	60%	48%	53%
	4	76%	74%	75%
	5	47%	70%	56%
	6	76%	81%	79%

Table 6: Recall, precision and F_1 measure for the *clean* dataset (Strategy 2)

$$C = \frac{646}{1004} = 64.3\%$$

Figure 9: Classification rate for the *clean* dataset (Strategy 2)

Evaluation of Strategy 2 on noisy dataset

		Predicted class					
		1	2	3	4	5	6
Actual class	1	**23**	11	21	15	11	7
	2	24	**108**	16	28	5	6
	3	25	14	**83**	37	8	20
	4	28	14	10	**135**	6	16
	5	12	10	12	20	**44**	12
	6	39	8	13	14	8	**138**

Table 7: Confusion Matrix for the *noisy* dataset (Strategy 2)

		Recall	Precision	F_1
Actual class	1	26%	15%	19%
	2	58%	65%	61%
	3	44%	54%	49%
	4	65%	54%	59%
	5	40%	54%	46%
	6	63%	69%	66%

Table 8: Recall, precision and F_1 measure for the *noisy* dataset (Strategy 2)

$$C = \frac{531}{1004} = 52.9\%$$

Figure 10: Classification rate for the *noisy* dataset (Strategy 2)

Evaluation of Strategy 3 on clean dataset

		Predicted class					
		1	2	3	4	5	6
Actual class	1	**78**	29	3	6	15	1
	2	12	**163**	1	9	9	4
	3	14	9	**80**	3	3	10
	4	3	10	3	**179**	13	8
	5	21	20	4	7	**76**	4
	6	4	9	14	3	10	**167**

Table 9: Confusion Matrix for the *clean* dataset (Strategy 3)

		Recall	Precision	F_1
Actual class	1	59%	59%	59%
	2	82%	68%	74%
	3	67%	76%	71%
	4	83%	86%	85%
	5	58%	60%	59%
	6	81%	86%	83%

Table 10: Recall, precision and F_1 measure for the *clean* dataset (Strategy 3)

$$C = \frac{743}{1004} = 74.0\%$$

Figure 11: Classification rate for the *clean* dataset (Strategy 3)

Evaluation of Strategy 3 on noisy dataset

		Predicted class					
		1	2	3	4	5	6
Actual class	1	**36**	10	15	2	18	7
	2	21	**123**	20	11	8	4
	3	24	12	**108**	16	9	18
	4	15	18	18	**141**	5	12
	5	25	6	9	6	**55**	9
	6	10	6	32	7	15	**150**

Table 11: Confusion Matrix for the *noisy* dataset (Strategy 3)

		Recall	Precision	F_1
Actual class	1	41%	27%	33%
	2	66%	70%	68%
	3	58%	53%	56%
	4	67%	77%	72%
	5	50%	50%	50%
	6	58%	75%	71%

Table 12: Recall, precision and F_1 measure for the *noisy* dataset (Strategy 3)

$$C = \frac{613}{1004} = 61.1\%$$

Figure 12: Classification rate for the *noisy* dataset (Strategy 3)

Comparison of methods

Table 13 presents classification rates for all three strategies. The first thing to notice is that Strategy 2, which we expected to work reasonably well, has actually performed the worst of all investigated strategies. On the other hand, Strategy 3 performed best in case of the *clean* dataset and very closely to Strategy 1 in the *noisy* dataset. However, since the classification rates for Strategy 1 and 3 in the *noisy* dataset are very close to each other, and Strategy 1 includes some random assignment, it might just have been the case that in this particular evaluation, Strategy 1 was just "lucky" and assigned the correct example in most cases where there was a conflict. If we ran the simulation again, Strategy 1 might have performed a bit worse. Taking this into consideration, the conclusion of this evaluation is that Strategy 3 is our best choice.

	Classification rate *clean* dataset	Classification rate *noisy* dataset
Strategy 1	71.7%	**61.9%**
Strategy 2	64.3%	52.9%
Strategy 3	**74.0%**	61.1%

Table 13: Aggregated classification rates for all strategies. Highlighted is the best result for a given dataset.

Pruning

The function `pruning_example` first generates a decision tree using MATLAB's built-in `treefit` function, based on the input examples `x` and classes `y`. It then runs the `treetest` function twice on the generated tree, each time using different arguments attributes:

1. `'cross'` - which performs a 10-fold cross-validation on the generated tree in order to determine, what is the optimal tree size, which is defined as the one for which the cost is within one standard error of the minimum-cost subtree
2. `'resubstitution'` - which tries to determine the optimal tree size, however this time using the resubstitution method. Resubstitution means that the test set is the same as training set

The function also prunes the entire tree based on the results of the `treetest` function (however these trees are never used anywhere in the function) and finally plots the results of the `treetest` function. The plots for the *clean* and *noisy* datasets are presented in Figures 13 and 14.

For the 10-fold cross-validation data, the blue line shows nicely the concept of overfitting. As we can see, the cost (or error) for the tree starts to go up above a certain tree size. In this case, the optimal tree size (as defined by MATLAB) for the *clean* dataset is 39, however the smallest cost is achieved at a tree size of 51 (tree size defined as a number of terminal nodes). For the *noisy* dataset, the optimal tree size is 29, however the smallest cost is achieved at a tree size of 51.

For the resubstitution method, the cost (red line) always goes down with the tree size. It is expected, since we are using the same data for training as then for testing. Therefore, the resubstitition method is not a reliable indicator of the tree performance on an unseen data at all, and should only be used to see whether the tree has been correctly trained on the training data.

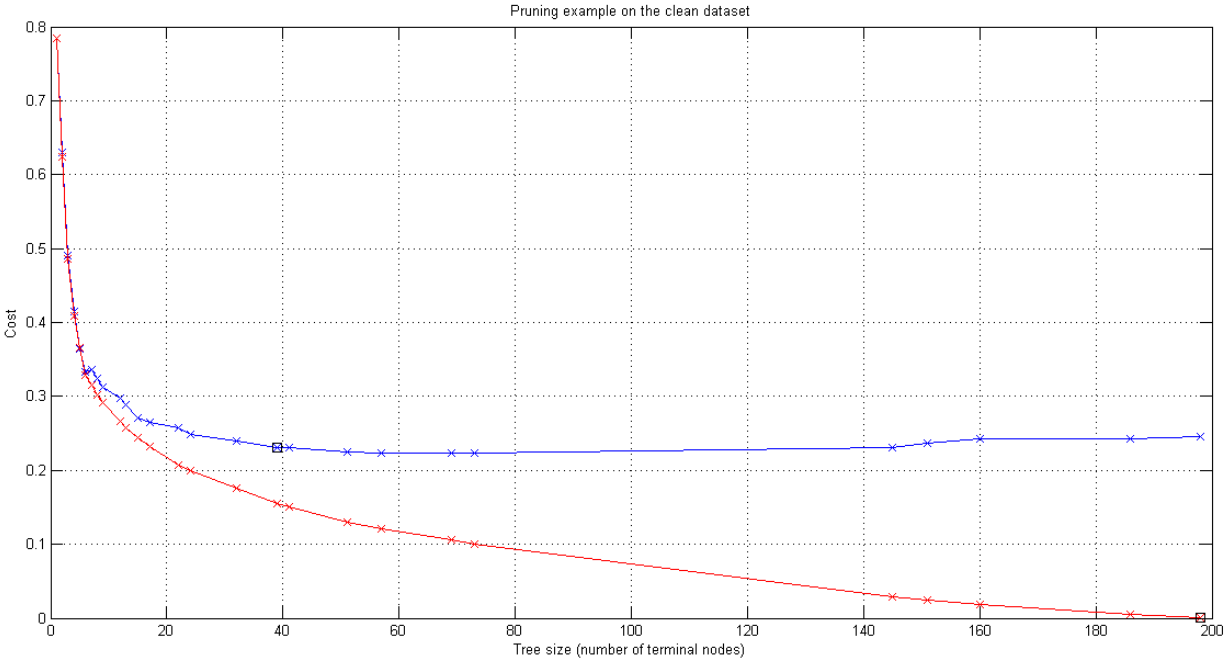

Figure 13: Pruning example on the *clean* dataset

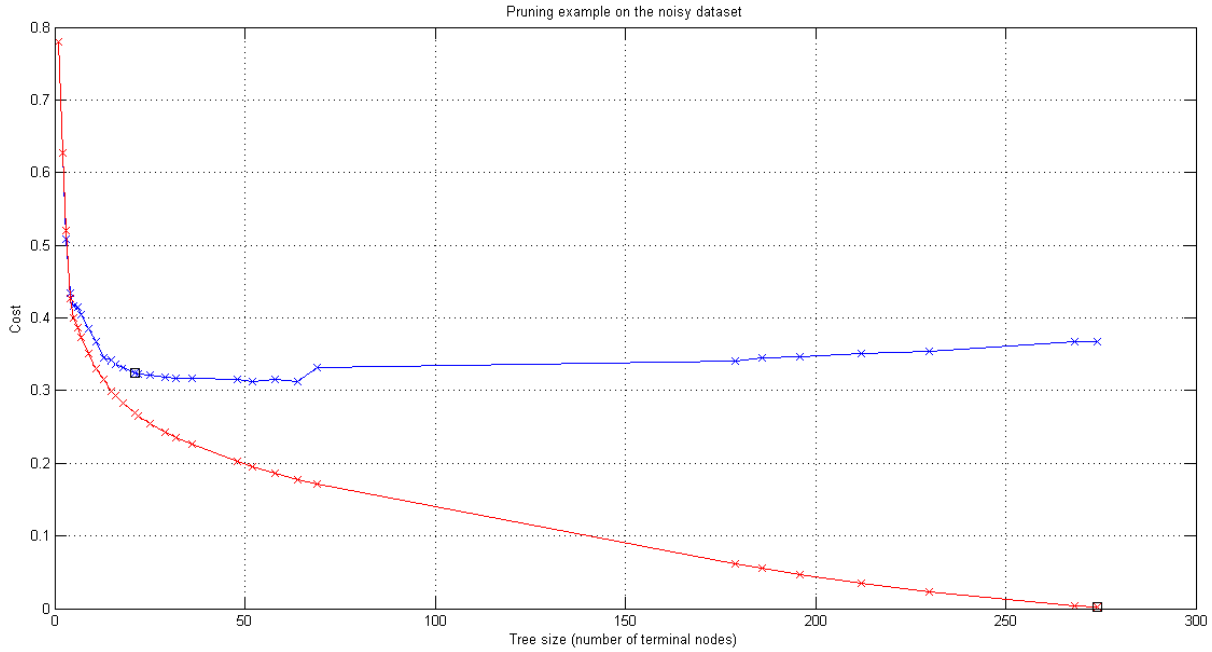

Figure 14: Pruning example on the *noisy* dataset

Code Flowchart

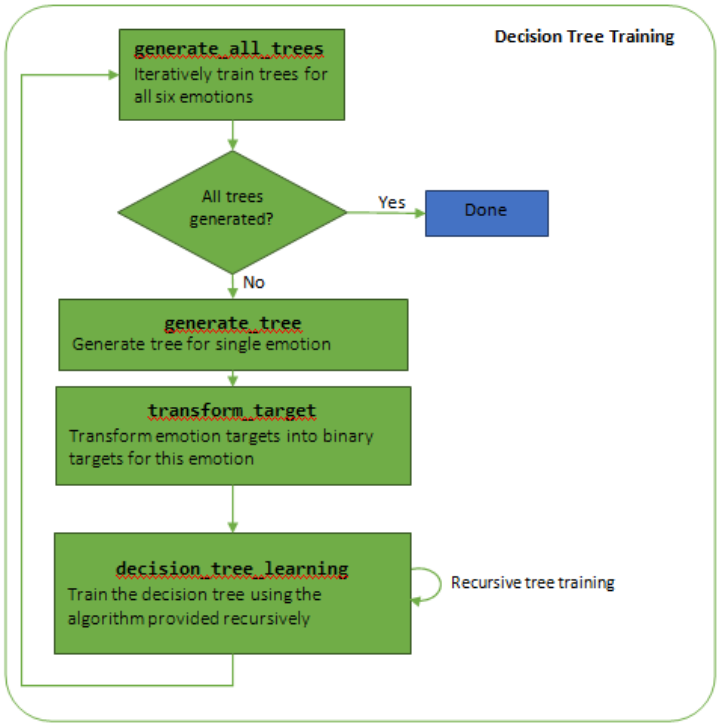

Figure 15: Flowchart for the functions which train the trees

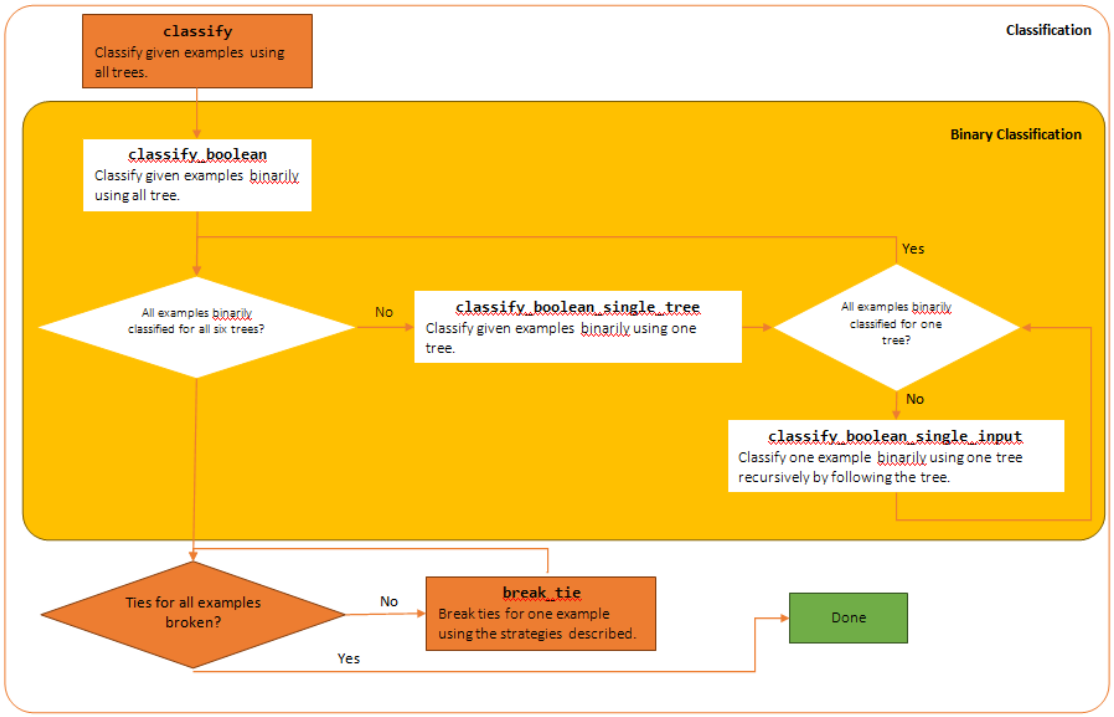

Figure 16: Flowchart for the functions which classify the examples