Decision Trees

Implementation Details

The best attribute is selected by determining which is the one that would split differentiating all of the true positives from true negatives. This is done computing the gain for each of the possible attributes, returning the one with the highest value.
The fold function took the examples and the labels, returning a struct in the following format:

```
folds:
        test:       [1x10]
        train:      [1x10]
        validate:   [1x10]
```

where test, train and validate contained 80, 900, and 20 examples and corresponding labels (there was some slight variation in size due to the data not being an exact multiple of 10). We then passed the training data for each fold into createDecisionTrees to make the decision trees for each fold, and then used testTrees on each fold's trees and each folds test examples. To find our final result, we calculated the error for each of the 10 folds by comparing the expected labels with the predicted ones, then took the arithmetic mean of them. *(Results are shown further on)*

Evaluation

For evaluating the average confusion matrix, we took the average of 100 confusion matrices. The results are as follows for the clean and dirty data set.

Average Confusion Matrices

Clean Data

70	11	5	5	11	4
15	111	3	7	12	9
5	4	70	2	4	15
2	10	3	144	9	4
18	8	7	9	52	6
3	4	11	8	7	134

Dirty Data

18	8	14	10	17	7
9	100	16	15	6	4
12	18	80	20	10	12
7	11	9	124	5	8
15	9	5	8	46	9
11	7	10	10	9	122

Class

Clean	1	2	3	4	5	6
Recall	66.0%	70.7%	70.0%	83.7%	52.0%	80.2%
Precision	62.0%	75.0%	70.7%	82.3%	54.7%	77.9%
F1	63.9%	72.8%	70.4%	83.0%	53.3%	79.1%

Class

Dirty	1	2	3	4	5	6
Recall	24.3%	66.7%	52.6%	75.6%	50.0%	72.2%
Precision	25.0%	65.4%	59.7%	66.3%	49.5%	75.3%
F1	24.7%	66.0%	55.9%	70.7%	49.7%	73.7%

From this we can see that our decision tree perform quite well with clean data, and fairly well with dirty data. All of the classes achieve above 50% recognition with clean data, meaning that the majority of the team we can achieve correct emotion recognition. Emotion 4 (happiness) shows quite a remarkable recognition rate given both dirty and clean data. Emotion 2 however is very hard to recognise given dirty data, and nearly approaches a the expected random recognition rate of ⅙ = 16.7%.

We used 2 different discriminants, one was d, and the other was 1/d (where d is the depth of the classified node in the tree).

We received the following classification rates on the **clean** data:
Discriminant = d :   71.3%
Discriminant = 1/d: 72.4%
Meaning the second discriminant performed better by 1.1%.

We received the following classification rates on the **dirty** data:
Discriminant = d :   59.3%
Discriminant = 1/d: 61.9%
Meaning the second discriminant performed better by 2.6%.

On the first discriminant (d), the clean data performed better by 12%
On the second discriminant (1/d), the clean data performed better by 10.5%

The rates of noisy errors compared to the clean dataset are:
Discriminant = d : 10.5/72.4 = 14,5%
Discriminant = 1/d: 12/71.3 = 16,8%

Noisy-Clean Datasets

There are differences between the clean and noisy datasets. Unsurprisingly, the noisy dataset has a lower performance than the clean one (by 14,5% over the results of clean dataset, using discriminant (1/d)). This was expected, since there are errors in the noisy dataset. Also, the trees were trained on the clean dataset, therefore we would expect to get a maximum performance when we test on the clean dataset (72.4%)
14.5% is an acceptable error rate compared to the clean dataset, with a clean performance of 72.4%, which comforts us in thinking that we used the right discriminant.

Ambiguity

Our reasoning:

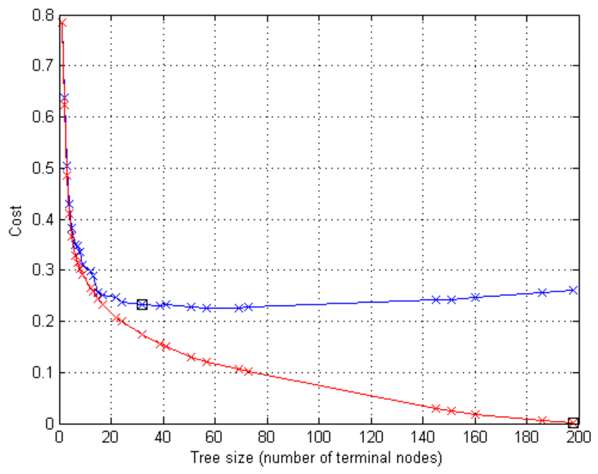
The first discriminant (d) represents the hypothesis that more specific rules would give a more accurate result, while the second discriminant (1/d) represents the hypothesis that more broad rules would give a more accurate result.

This is because a node with low depth represents a more broad rule (as it occurs earlier). Likewise, a node with higher depth represents a more specific rule (as it occurs later).

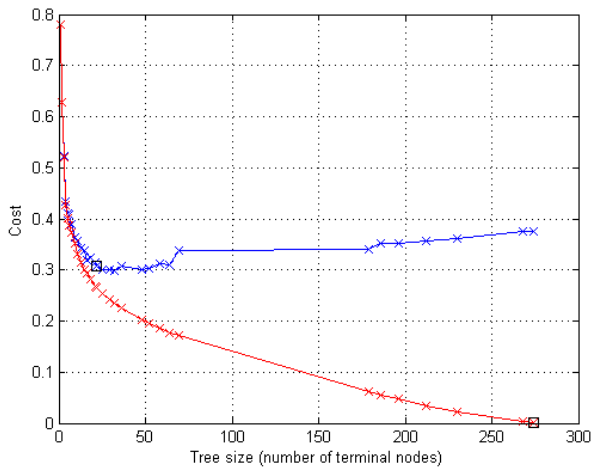
Our results:

In both cases, we found that the second discriminant (1/d) was superior. This suggests that more broad rules should be given more weighting, which makes intuitive sense.

Pruning



clean data

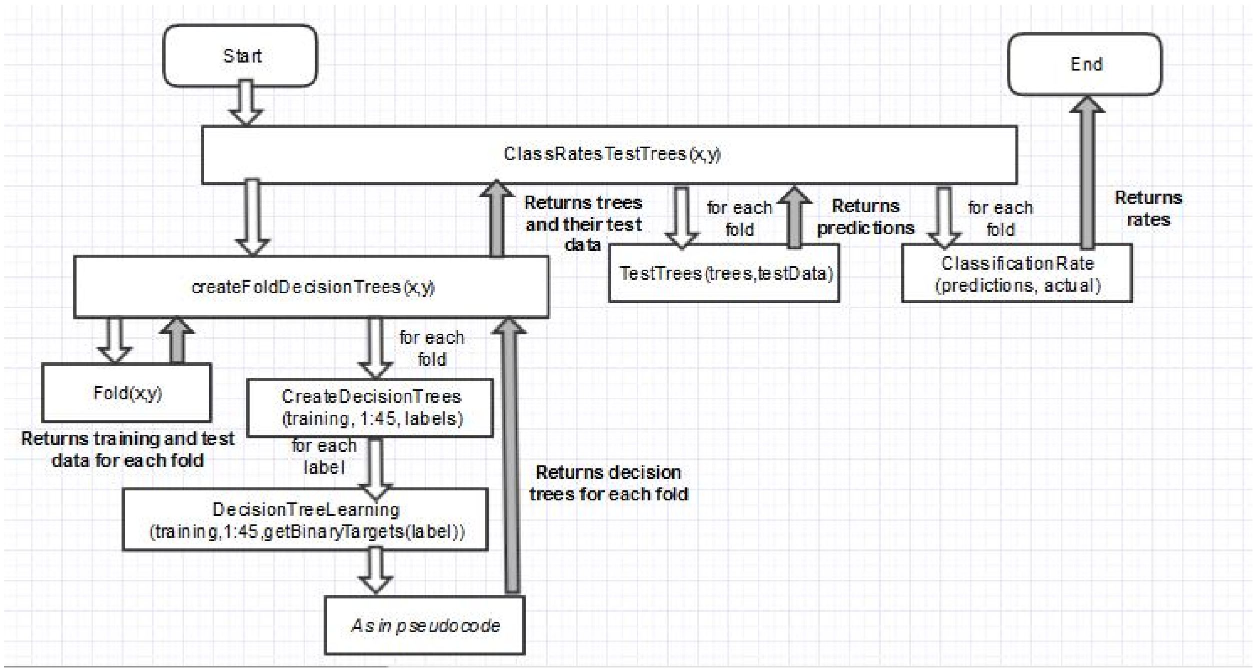


noisy data

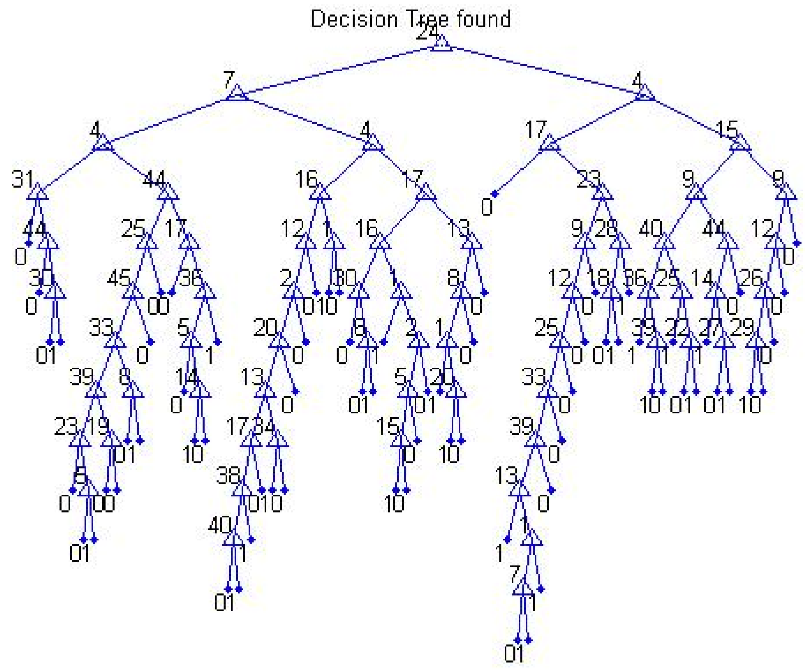
The pruning_example function builds a decision tree from the test data using treefit, and then by using treetest it evaluates both the cost vector and the optimal pruning level (the level which creates the smallest tree that is less than one standard error away from the subtree with minimum cost) for the tree determined by cross validation, and also when using resubstitution.
Pruning works by discarding some data that is negligeable to lower the complexity of the tree.
The blue line indicates the validation error, the red line indicates the training error

As the size of the tree increases, there is a tendency to overfit the training data, which is shown by the curve rising again after a global minima. This happens because as the number of nodes increases, tree becomes more specific to the training set, as we only have a limited number of examples to test on. The minima of the graph shows us the optimum tree size.
Optimal tree size for clean data is about 70, for noisy data it is about 50.

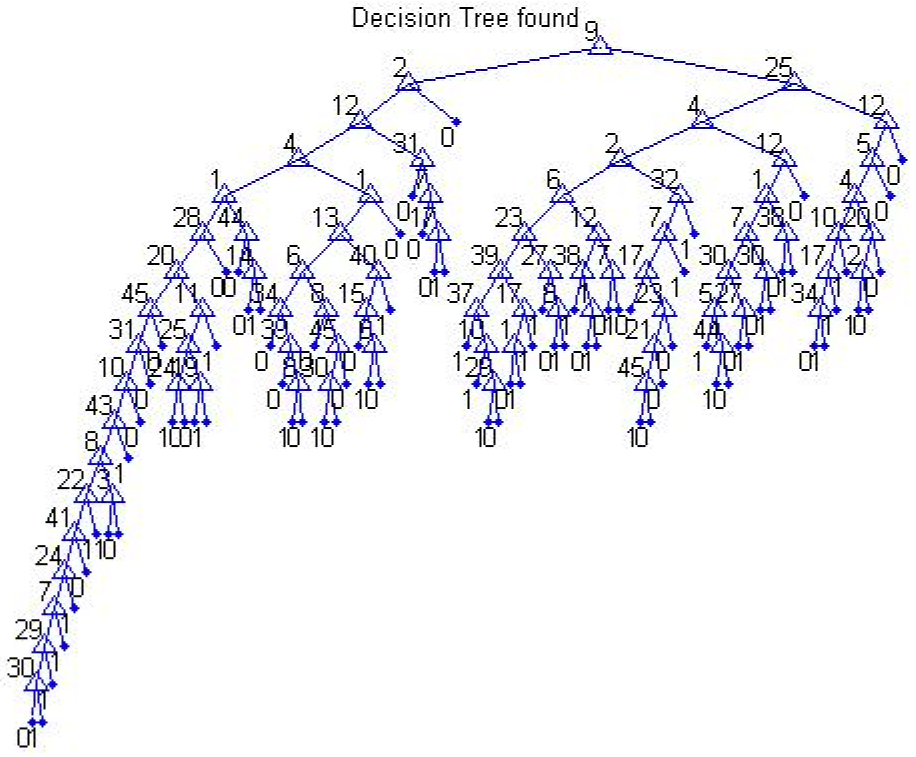
Flowchart illustrating how code works


Start
End
ClassRatesTestTrees(x,y)
Returns trees and their test data
for each fold
Returns predictions
for each fold
Returns rates
createFoldDecisionTrees(x,y)
TestTrees(trees,testData)
ClassificationRate (predictions, actual)
for each fold
Fold(x,y)
CreateDecisionTrees (training, 1:45, labels)
Returns training and test data for each fold
for each label
Returns decision trees for each fold
DecisionTreeLearning (training,1:45,getBinaryTargets(label))
As in pseudocode

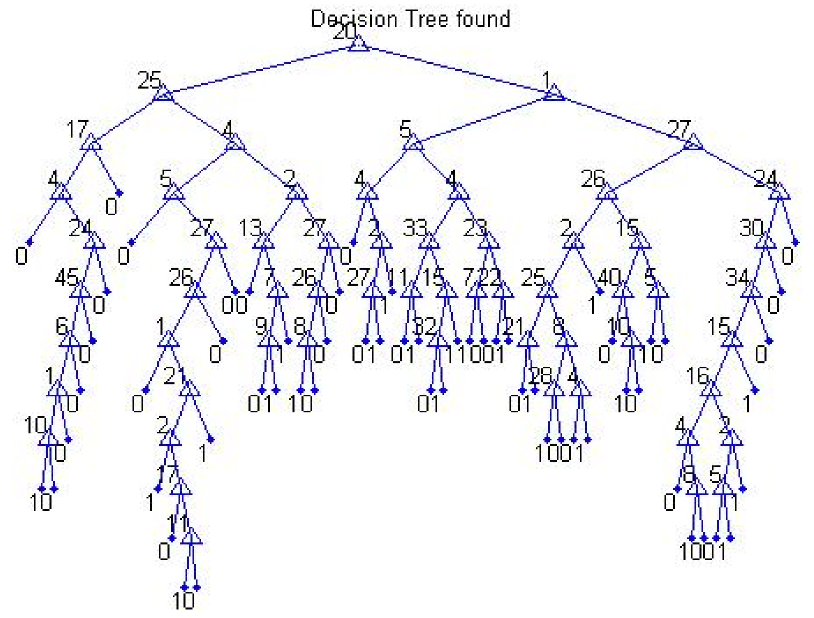
Tree 1



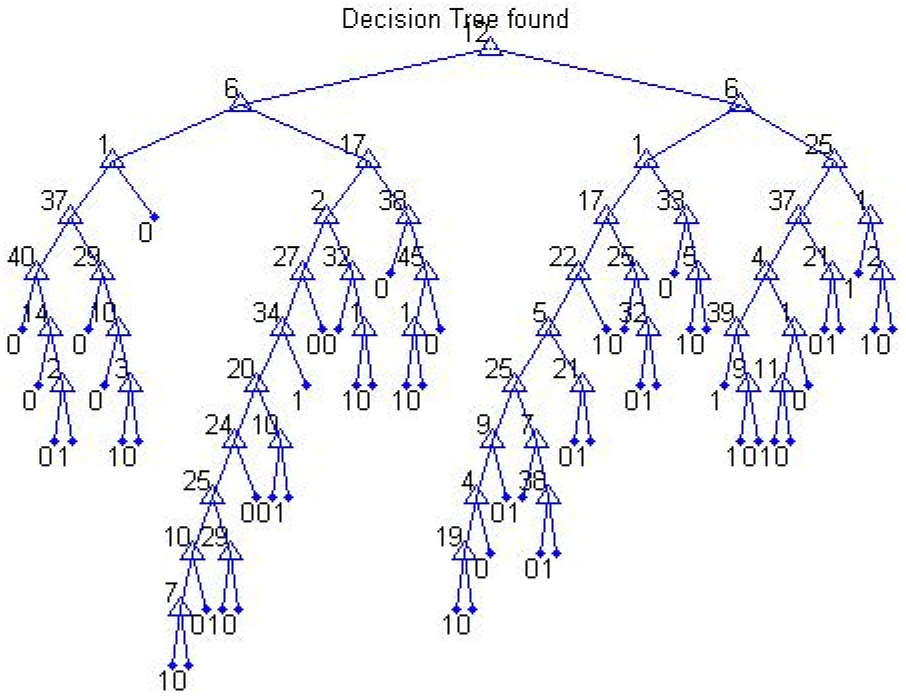
Tree 2



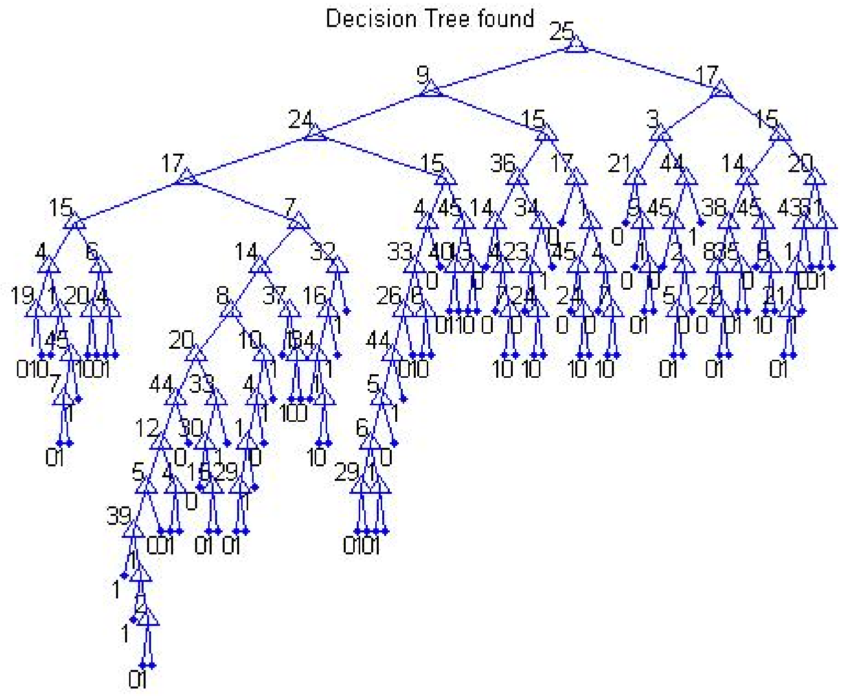
Tree 3



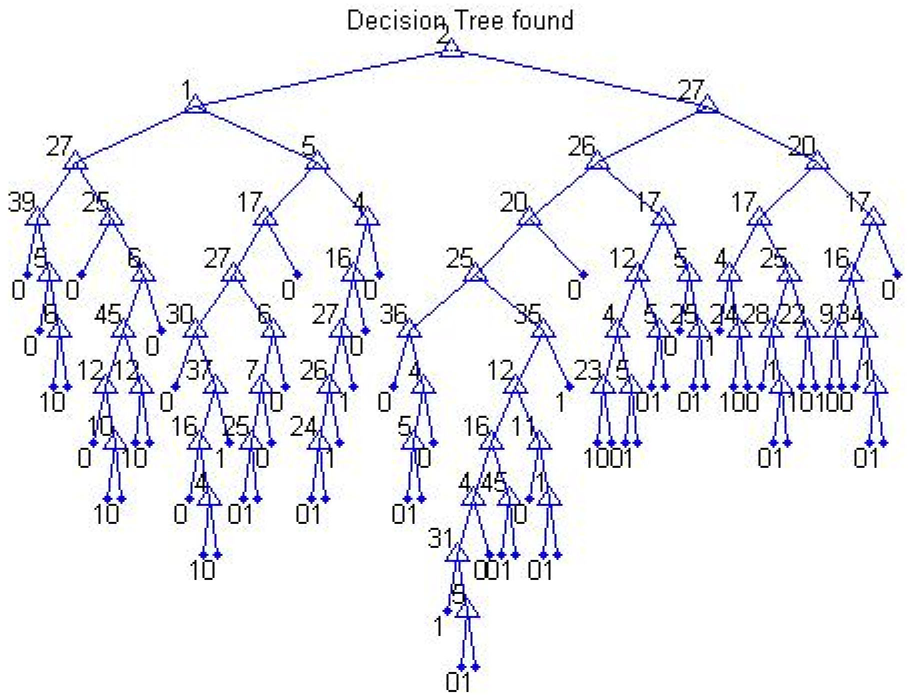
Tree 4



Tree 5



Tree 6


Decision Tree found