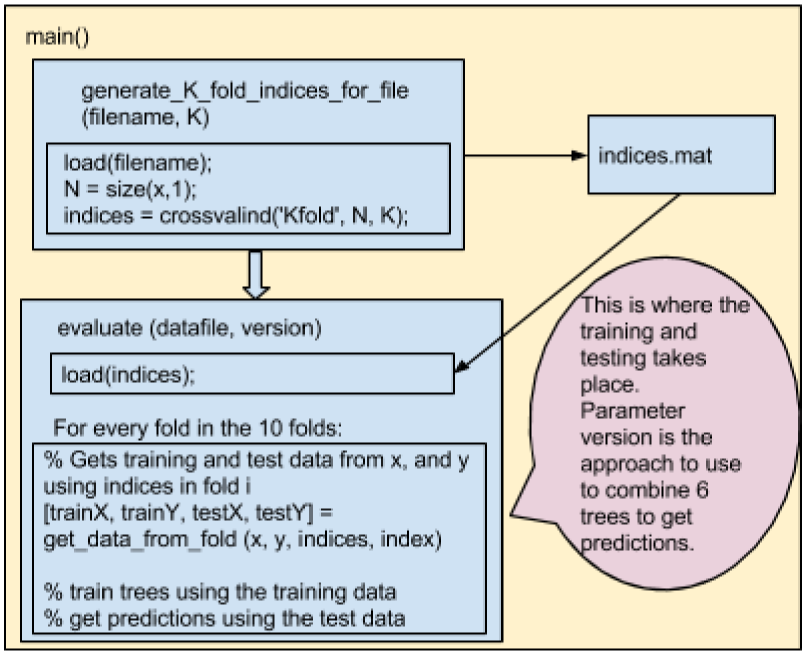
Decision Trees Algorithm Report

Group 20

Summary of Implementation Details

How we performed cross-validation


main()
generate_K_fold_indices_for_file (filename, K)
indices.mat
evaluate (datafile, version)
For every fold in the 10 folds:
This is where the training and testing takes place. Parameter version is the approach to use to combine 6 trees to get predictions.

```
load(filename);
N = size(x,1);
indices = crossvalind('Kfold', N, K);
```

```
load(indices);
```

```
% Gets training and test data from x, and y using indices in fold i
[trainX, trainY, testX, testY] = get_data_from_fold (x, y, indices, index)

% train trees using the training data
% get predictions using the test data
```

The validation set is ignored for now since in the decision trees there are not any parameters involved that have to be tuned.

How we selected the best attribute in each node

The best attribute selected is the attribute with the highest information gain. ID3 algorithm is used to compute the Information gain for each of the attributes. We use the given set of data to compute the information gain for each of the attribute using the following formula as described by the ID3 algorithm:

$$IG(A,S) = E(S) - \sum_{t \in T} p(t) * E(t)$$ [1]

IG: Information Gain

E(S) : calculates the entropy of the given attribute using $E(S) = -\sum_{x \in X} p(x)\, log2(p(x))$

p(x): probability of x in given set

T: The subsets created from splitting set S by attribute A such that $S = \bigcup_{t \in T} t$

To avoid errors in the log function when log of 0 could be computer, we use a wrapper function around the log2 function provided by the Matlab. This wrapper function returns 0 if log2(0) is executed and returns the value using log2 function provided by Matlab for all other values.

How we computed the average results

Average Confusion Matrix & Average Classification Rate

Inside each fold, we calculated the confusion matrix for that fold using confusion_matrix(testY, predictions). At the end of the for loop, We then summed the 10 confusion matrices and divided each element in the confusion matrix by 10 to get the average confusion matrix.

Similarly, we summed up the 10 classification rates calculated from each fold using sum(predictions == testY)/length(testY), and divided this sum by 10 to get the average classification rate.

Average recall and precision rates per class, and derived F1-measures

We calculate the average recall and precision rates for each class from the average confusion matrix using calculate_r_p_rate_fa(avg_c_matrix). Inside each loop for a class in calculate_r_p_rate_fa(), we call f_alpha_measure(1, precision_rate, recall_rate) with the average recall and precision rate for that class and with alpha 1 to get the f1 measure. The results are saved in a structure and exported to a .mat file for future reference.

Commented results of the evaluation - See Appendix B for Results

The obvious observation is for either clean dataset and noisy dataset, the second approach, likelihood & similarity approach, is better than the naive random approach in every indicators. Both approaches performed well on the clean dataset (about 70% and 80% classification rates). However, with the noisy dataset, the performances for both approaches decreased significantly (about 60% and 65%).

Let's us have a deeper look on the performances. For both approaches, by using the F1-measure indicators, we can see we are doing well enough in the clean dataset except class 5 which has a lower value than other classes in both ways. Furthermore, by looking at the F1-measure for our noisy data, we can conclude that we are doing very badly for class 1, 3 and 5 for both approaches with a very low value of F1-measure. On the other hand, we can see that we are doing really well in class 4 for both ways in both dataset.

Noisy-Clean Datasets Question

Is there any difference in the performance when using the clean and noisy datasets?

Yes there is a difference. The noisy data is obtained by an automated action unit (AU) recognition system which is not 100% accurate. Although the label allocated to each example is always correct, the AU's detected may be wrongly detected or missing. The clean dataset on the other hand was obtained from human experts so is 100% correct.

The incorrect AU's detected in the noisy dataset introduces errors into the decision tree. As a result, when classifying an unseen example there may be a difference between the viable choices when compared with the clean dataset tree. This may mean a viable choice may be introduced or missed out when it wouldn't have been in the clean dataset delivering reduced performance compared to the clean dataset.

What we observed

We observed this difference in performance in our results. Using the data in the Average Classification Rate table (see appendix B) we can see there is a clear difference in the performance obtained by using the clean data set and the noisy data set. Using the noisy data set at, best we obtained a successful classification rate of 65.0% however with the clean data set we obtained a success rate of 79.8% - a difference of nearly 15% which is significant difference.

We also observed a visual difference in the trees produced when using the clean and noisy data sets. We noticed from the pruning examples, as we provided more of the noisy data, the optimal tree size was reduced. This would suggest that the noisy data had missing AU's compared to the clean data set resulting in a less complex decision tree once pruned.

Countering performance loss

We have managed to reduce the performance impact of the noisy data by introducing a similarity function and calculating the likelihood of each option. We have implemented the cosine similarity function that can will choose viable options based on the similarity or viability of a choice rather than expecting an exact match. We then pick the choice that is statistically most likely. We have shown this to slightly reduce (but not eliminate) the performance impact of the noisy dataset compared to the clean dataset.

Looking at the Average Classification Rate result table (see appendix B) after training with the noisy dataset and using a random selection approach the successful classification rate was 59.8%. Using the same noisy training set but using the cosine similarity and likelihood approach, we were able to increase the successful classification rate to 65.0%.

What we could have done

In fact, before we build the tree, we should use our similarity function to filter out any training data which cannot pass our average similarity threshold that calculated from the clean data set. Therefore, we will have a better training set to build our tree and we will in turn get better results for our predictions.

We could have used different algorithms for computing the best attribute such as C4.5 algorithm which is improved version of ID3 and could potentially provide us with better results.

Ambiguity Question

For each emotion, we have a corresponding tree for classification. When a new instance **x** come, we send this instance to all 6 trees and gather the results from the trees. After this process, we will have a set of result with length = 6 and each element in the set is either true or false.

The following is an example

1	2	3	4	5	6
False	False	True	False	True	True

As shown in this example, our trees classified the instance to all class 3, 5 and 6.
Therefore, in some cases, we cannot determine a unique class to this instance.
However, we come up with 2 solutions to solve this problem.

1) Naive Random Approach
In this approach, after we get the results from 6 trees.
We will be in one of the following cases.

Case 1. Only one true element in our result set	In this case, we can return the index, which representing the class, for that element.
Case 2. No element in our result set is true	In this case, we pick a class from all classes randomly according to the uniform distribution and return it
Case 3. More than one true elements in our result set	In this case, we randomly pick a class from the classes that we think it belongs to according to the uniform distribution and return it

2) Likelihood & Cosine Similarity Approach
We think we can have a better approach than naive random, and therefore, we came up with this approach which is more statistical and should be more reliable.
In this approach, we also will be one of the following 3 cases.
Case 1. Only one true element in our result set
Similar to approach 1, we can return the index, which representing the class, for that element.
Case 2. No element in our result set is true
In this case, if we have no element in our result set, we compare the similarities of the instance with each classes to see which class is the most similar. And return the most similar class.
Case 3. More than one true elements in our result set
In this case, for all classes we think it belongs to be, we try to find
$P(x|C_i)$, $\forall$ *class i we think it belongs to be*
By the Kolmogorov definition of conditional probability, we can rewrite $P(x|C_i)$

$$P(x|C_i) = \frac{P(x \wedge C_i)}{P(C_i)}$$
$$= \frac{\frac{\text{number of class with feature exactly equal to x and also belong to class i}}{\text{total number of class in the training data}}}{\frac{\text{number of class i in the training data}}{\text{total number of class in the training data}}}$$
$$= \frac{\text{number of class with feature exactly equal to x and also belong to class i}}{\text{number of class i in the training data}}$$

However, we think this calculation is not practical enough since in the dataset, either clean or the noisy one, it is really hard to have one instance that have the exact features with our new test datum.
As a result, instead of exact equal, we are now trying to find out the cosine similarity for the new instance **x** and the instances in class i in our clean dataset.
A cosine similarity for two vectors is defined as follow:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

[http://en.wikipedia.org/wiki/Cosine_similarity]
In this scenario, **A** is the new instance **x** and **B** is one class i instance in our clean dataset.
Now, if the cosine similarity of instance **x** and our compared data is larger than a threshold, we assume they are the “same”.
The question now is how to determine the similarity thresholds. We believe that for different classes we have a different thresholds T_i. For each element in one specific class, we compare it with all other elements in the same class and get a average value of similarities. Then we average all the average values.

$$T_i = \frac{\sum_{k=1}^{n} \sum_{j=1, k \neq j}^{n} \frac{similarity(d_k, d_j)}{n}}{n}$$

,where n is the number instances in class i and d_1 to d_n are the instances in class i.
So now, our equation becomes the following:

$$P(x|C_i) = \frac{\text{number of class with cosine similarity to } x > T_i \text{ and also belong to class i}}{\text{number of class i in the training data}}$$

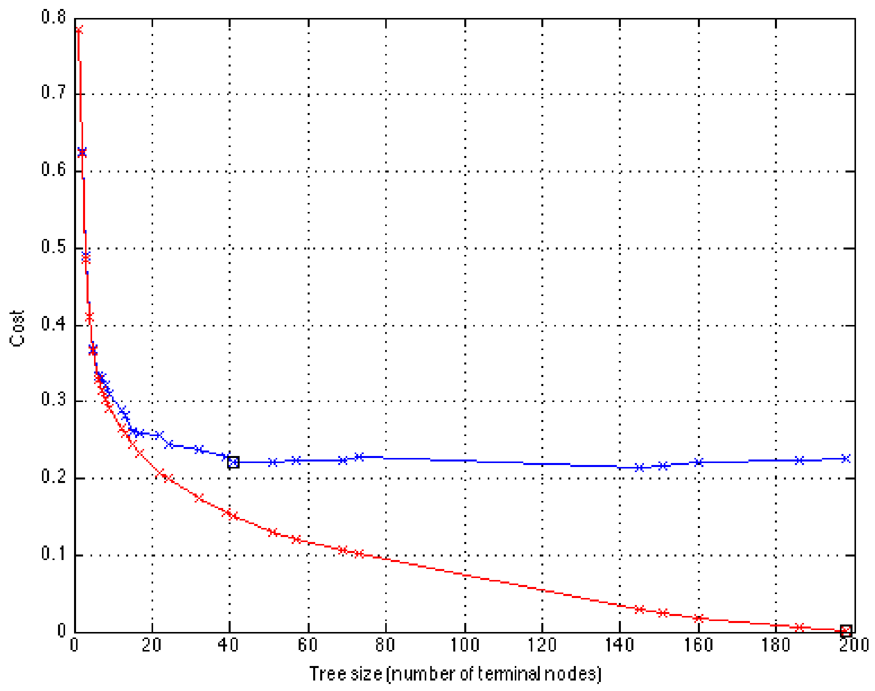
After calculating all $P(x|C_i)$, we return the class with the highest value.

	Naive Random	Likelihood & Cosine Similarity
Advantages	● Easy to understand ● Easy to implement ● Small computation	● More accurate ● Make sense with statistics ● Totally separate with trees
Disadvantages	Not very accurate	● Harder to implement ● Very large computation

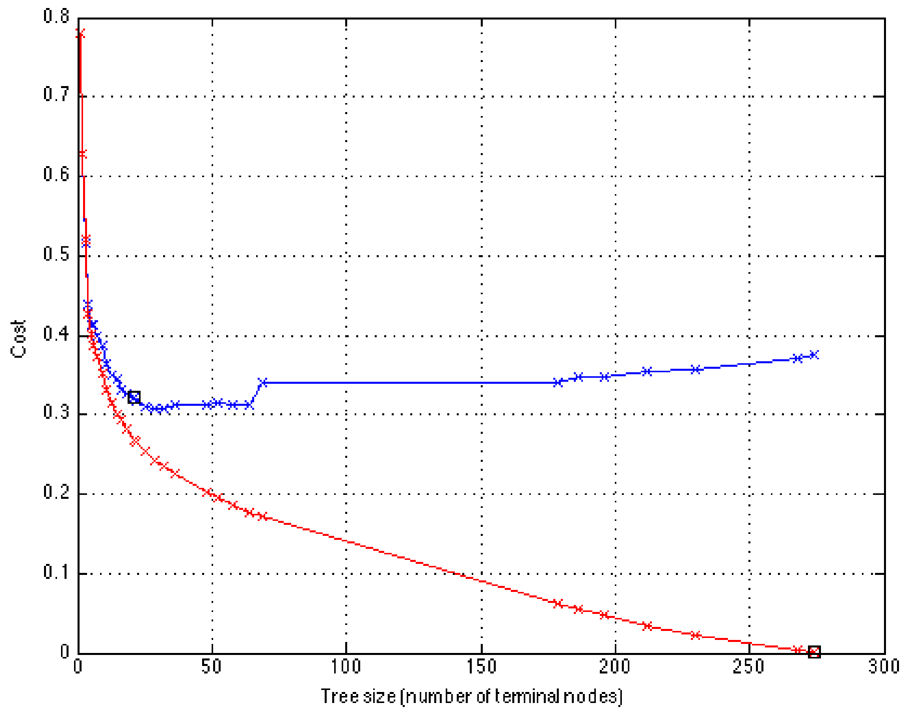
From the results above, we can see the second approach is more accurate and reliable, but it requires longer implementation and computation time.

Pruning Question

The pruning_example function firstly will build a tree, pass it to two validation methods which are 10-fold cross validation and resubstitution. Then, it will plot the graph below which shows the relationship between the tree size and the cost which can be considered to be the error rate for our case for both validation methods, where the blue curve is 10-fold cross validation and the red curve is resubstitution. After plotting the graph, it will run a matlab built-in function called testTree which basically gives us all the points we can prune on and the corresponding costs, of course included the best level to prune and the minimum cost point. Note, it is needed to mention there is a trade-off on the tree size and the cost. As a result, the best level for pruning is always within 1 standard deviation from minimum cost tree according to the matlab's implementation..



The above Graph was obtained using the pruning_example on the clean data. By analysing the graph we can see that the best level value of the tree for the clean data set under 10-fold validation is approximately 42 and under resubstitution is about 198.



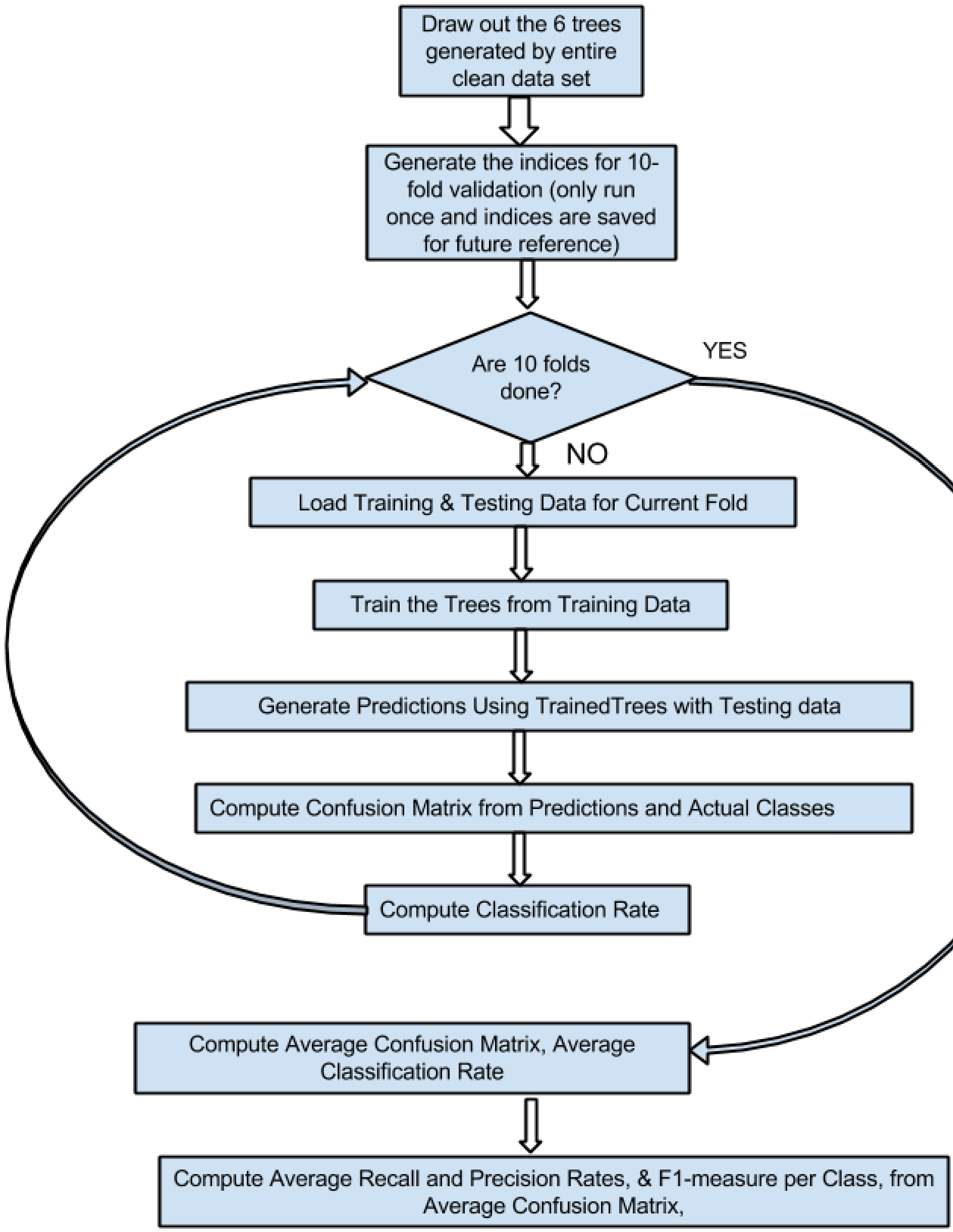
The above graph displays the result obtained when pruning_example was run using the noisy data. We can see that the best level value obtained for the 10-fold cross validation in the noisy data set is approximately 21. In this graph, there is a sudden increase point for the 10-fold validation that appear in trees with between 50 and 100 terminal nodes. We think that may suggest it is a threshold point which indicated overfitting. Therefore, we may stop training after this point. Also, we see that the best point to prune the tree using resubstitution is in maximum tree size.

From both results we can see the problem of overfitting which is very typical while training decision trees. Overfitting happens when the learning algorithm continues to develop hypotheses that reduce training set error at the cost of an increased test set/new data error. We can see that the error rate is affected more for the noisy data, and hence the optimal size of tree under 10-fold cross validation is lower than the optimal size for the clean data which was expected. The resubstitution curve approaches 0(also the best point to prune) because once we have provided all the data for the training, the predictions will mostly be correct, therefore a lower error rate. However, in other words, it means it is completely overfitting.

REFERENCES:

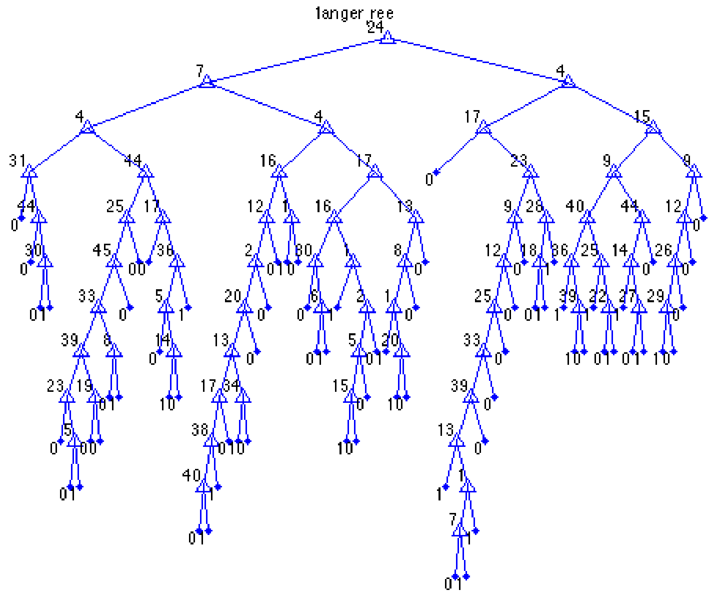
[1]: http://en.wikipedia.org/wiki/ID3_algorithm
[2]: http://www.mathworks.co.uk/help/stats/treetest.html

Flowchart of Code - Starting Point main()

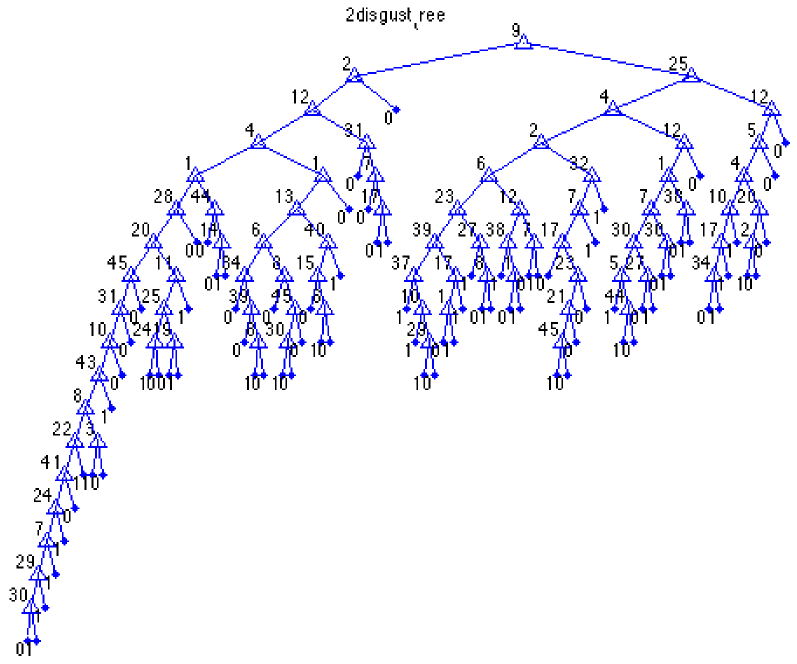

Draw out the 6 trees generated by entire clean data set
Generate the indices for 10-fold validation (only run once and indices are saved for future reference)
Are 10 folds done?
YES
NO
Load Training & Testing Data for Current Fold
Train the Trees from Training Data
Generate Predictions Using TrainedTrees with Testing data
Compute Confusion Matrix from Predictions and Actual Classes
Compute Classification Rate
Compute Average Confusion Matrix, Average Classification Rate
Compute Average Recall and Precision Rates, & F1-measure per Class, from Average Confusion Matrix,

Appendix A - Diagrams of the six trees trained on the entire clean dataset

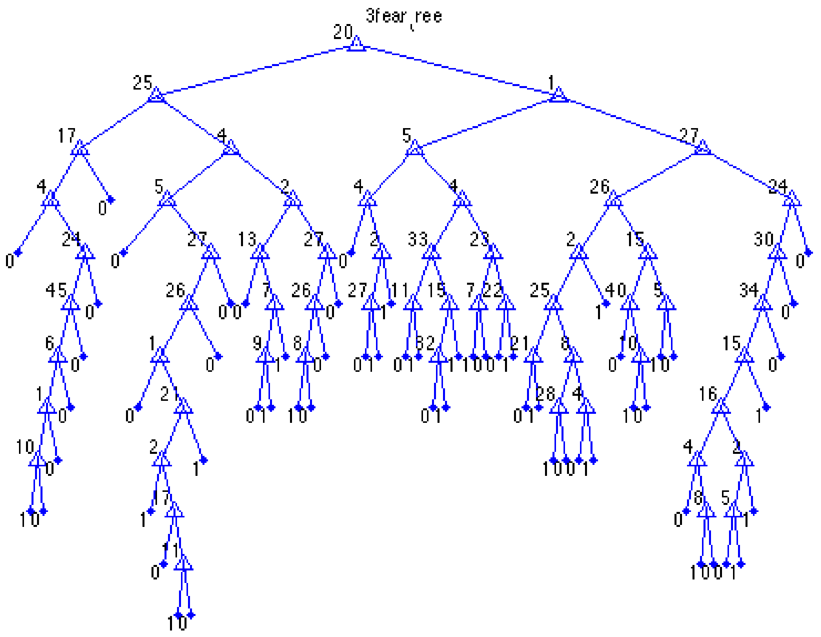
Tree 1 - Anger tree



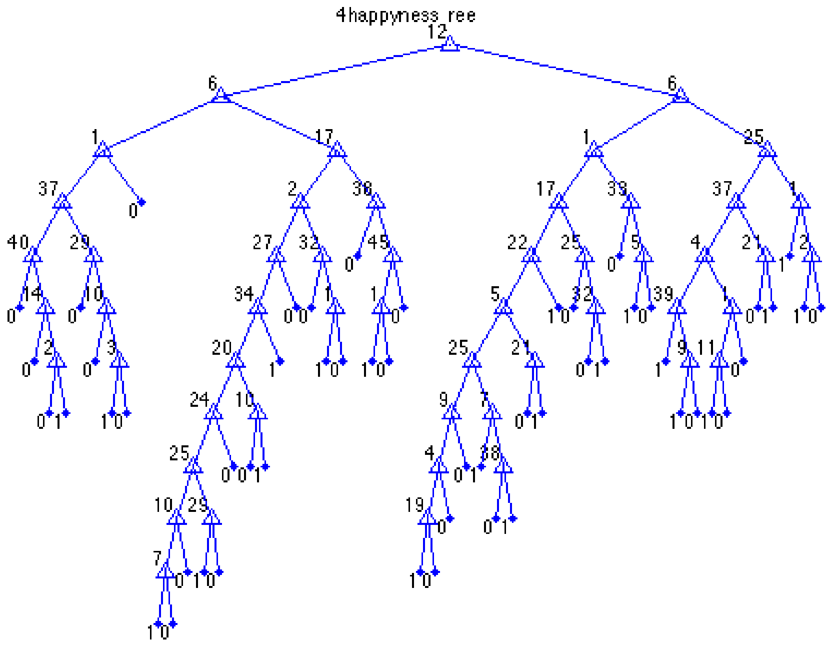
Tree 2 - Disgust tree



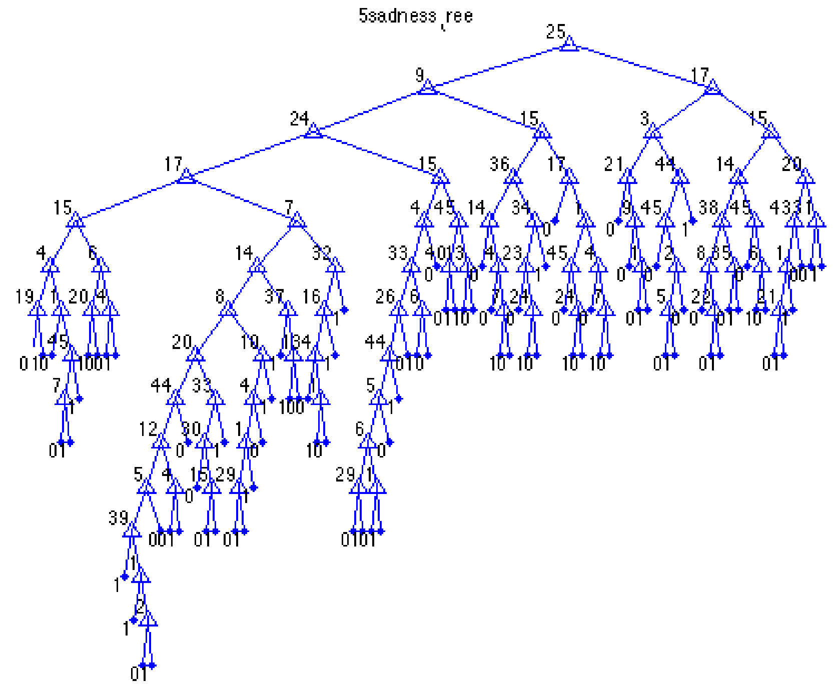
Tree 3 - Fear tree



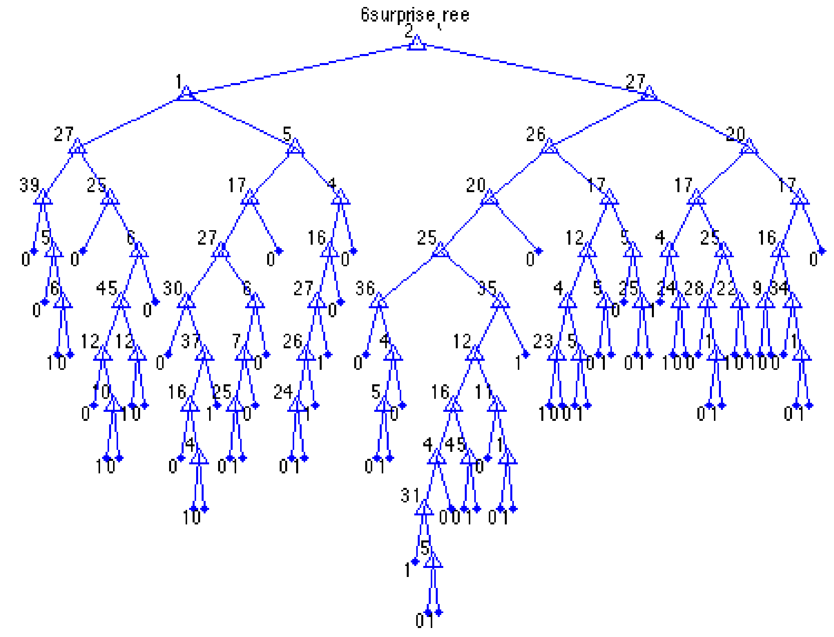
Tree 4 - Happiness tree



Tree 5 - Sadness tree



Tree 6 - Surprise tree



Appendix B - Results Tables

Average Classification Rate

Dataset	Approach	Average Classification Rate
Clean	Naive Random Approach	0.728
Clean	Likelihood & Cosine Similarity Approach	0.798
Noisy	Naive Random Approach	0.598
Noisy	Likelihood & Cosine Similarity Approach	0.650

Clean Dataset - Naive Random Approach

Average Confusion Matrix

	Predicted 1	Predicted 2	Predicted 3	Predicted 4	Predicted 5	Predicted 6
Actual 1	8.6	1.4	0.5	0.5	1.6	0.6
Actual 2	0.6	15.1	0.8	1.2	1.2	0.9
Actual 3	0.8	0.9	8.4	0.6	0.4	0.8
Actual 4	0.5	0.8	0.4	17.9	1.1	0.9
Actual 5	1.7	1.9	1	0.7	7	0.9
Actual 6	0.5	0.8	2.2	0.5	0.6	16.1

Average Recall Rate, Average Precision Rate, F1-measure for each class

Class	Average Recall Rate	Average Precision Rate	F1-Measure
1	65.15	67.72	66.41
2	76.26	72.25	74.20
3	70.59	63.16	66.67
4	82.87	83.64	83.26
5	53.03	58.82	55.78
6	77.78	79.70	78.73

Clean Dataset - Likelihood & Cosine Similarity Approach

Average Confusion Matrix

	Predicted 1	Predicted 2	Predicted 3	Predicted 4	Predicted 5	Predicted 6
Actual 1	9.6	0.8	0.8	0.4	1.3	0.3
Actual 2	0.9	14.9	0.3	1.2	1.2	1.3
Actual 3	0.2	0	8.9	0.1	0.3	2.4
Actual 4	0	0.5	0.1	19.2	0.5	1.3
Actual 5	1.6	1.1	0.6	0.3	8	1.6
Actual 6	0	0.2	0.7	0	0.3	19.5

Average Recall Rate, Average Precision Rate, F1-measure for each class

Class	Average Recall Rate	Average Precision Rate	F1-Measure
1	72.73	78.05	75.29
2	75.25	85.14	79.89
3	74.79	78.07	76.39
4	88.89	90.57	89.72
5	60.61	68.97	64.52
6	94.20	73.86	82.80

Noisy Dataset - Naive Random Approach

Average Confusion Matrix

	Predicted 1	Predicted 2	Predicted 3	Predicted 4	Predicted 5	Predicted 6
Actual 1	2.5	1.4	1.6	1	1.5	0.8
Actual 2	1.3	11.9	1.4	1.5	2	0.6
Actual 3	2.4	1.4	9.8	1.8	1.5	1.8
Actual 4	1.2	0.7	1.4	15.6	0.7	1.3
Actual 5	1.8	1.3	1.6	0.8	4.4	1.1
Actual 6	0.9	0.7	1.4	1.6	1.7	15.7

Average Recall Rate, Average Precision Rate, F1-measure for each class

Class	Average Recall Rate	Average Precision Rate	F1-Measure
1	28.41	24.75	26.46
2	63.64	68.39	65.93
3	52.41	56.98	54.60
4	74.64	69.96	72.22
5	40.00	37.29	38.60
6	71.36	73.71	72.52

Noisy Dataset - Likelihood & Cosine Similarity Approach

Average Confusion Matrix

	Predicted 1	Predicted 2	Predicted 3	Predicted 4	Predicted 5	Predicted 6
Actual 1	3.5	0.3	1.6	0.6	1.7	1.1
Actual 2	1.8	11.4	2	1.4	1.3	0.8
Actual 3	1.2	0.5	11	1	0.6	4.4
Actual 4	0.9	0.1	1.3	15.5	0.4	2.7
Actual 5	1.8	0.6	1.4	0.3	4.9	2
Actual 6	0.4	0.2	0.9	0.9	0.8	18.8

Average Recall Rate, Average Precision Rate, F1-measure for each class

Class	Average Recall Rate	Average Precision Rate	F1-Measure
1	39.77	36.46	38.04
2	60.96	87.02	71.70
3	58.82	60.44	59.62
4	74.16	78.68	76.35
5	44.55	50.52	47.34
6	85.45	63.09	72.59