Implementation Details

Selecting the best attribute

When building the decision tree for a given training set, if all the binary target values are not same, then we have to split by the right attribute, such that it will split the current training set into two sets which splits binary targets into two uneven sets. We used information gain to calculate to determine the best attribute. The information gain for an attribute a, is defined to:
Gain(*a*) = entropy of current set (*I*) - entropy of the subsets that we get from splitting using *a* (Remainder).
We give preference to higher information gain, since it implies that there is less uncertatinity in the resulting subsets. Once we choose the best attribute, we remove it from the list of attributes from which we can choose the next best attributes.

Cross Validation

When performing cross-validation we had to ensure that the test set and training set were approximately equal and all data was used in testing and training, this is not a problem when the number of folds is a factor of the data size. However if the number of folds was not a factor of the size of the data set, then we had to split it evenly. To do this we first calculated the modulus(m) of folds (f) and data size (d), and for m folds we used floor(d/f) + 1 as our testing set size, and for the remaining f-m folds we used floor(d/f) as our testing set size. For each of fold we generate the confusion matrix, that we got by training the decision tree using the training set and testing it with the test set.

Computing Average Results

Once we perform cross validation, to get the average result, we sum the confusion matrices and divide each element by the number of folds. Once we obtain the average confusion matrix, we can easily calculate the necessary performace measure for each class. We then compared performance by using classification rate and F1 values that we obtained for each class.

System Implementation

The implementation of our decision tree algorithm, requires us to at times, choose the most common value in binary targets. When the binary target has multiple most common values, matlab's mode function returns the smallest value, but to ensure fairness, we altered the majority-value function, so that if there are multiple most common value then we randomly choose one the values.

Average Results

Having analysed the performance for both clean and noisy data it seems that classifying data based on tree lengths is slightly better than classifying data by giving preference to balanced trees. Below is the performance result for classifying data based on length.

Average Confusion Matrix

Clean data-Length based classification

90.9	11.5	3.5	2.1	8.4	2.4
54.6	104.5	2.8	4.1	9.6	2.6
26.9	10.2	56.3	2.2	3.1	8.4
30.6	23.2	4.5	131.6	2.6	1.9
48.7	16.1	4.5	5.1	40.4	4
35.4	11.3	16.3	7.3	7.8	108.2

Class	Recall rate (%)	Precision rate (%)	F1 (%)
1	76.5152	31.6614	44.7894
2	58.642	59.1063	58.8732
3	52.5677	64.0501	57.7436
4	67.6955	86.3517	75.8939
5	34.0067	56.1892	42.3702
6	58.0784	84.8627	68.9611

Classification rate = 58.86%

Clean Data-balanced tree based classification

71.4	15.9	5.3	6	14.8	5.4
29.5	107	6.5	8.5	18.6	8.1
16.6	10.9	56.5	4.4	6.1	12.6
18.9	23.6	5.1	135.3	6.8	4.7
28.1	20.4	5.8	9.5	47.1	7.9
17.9	15	17.3	7.9	10	118.2

Class	Recall rate (%)	Precision rate (%)	F1 (%)
1	60.1010	39.1447	47.4104
2	60.0449	55.4979	57.6819
3	52.7544	58.5492	55.5010
4	69.5988	78.8462	73.9344
5	39.6465	45.5513	42.3942
6	63.4461	75.3346	68.8811

Classification rate = 59.26

Noisy Data-Length based classification

46.1	9.6	9.4	5	6.7	2.4
43	109.9	5.9	3.8	3.4	2.3
54.9	18.1	67.4	9.8	6	12.1
43.9	18.3	14.3	100.3	3.4	7.9
44	11.8	11	4.3	23.9	4
43	10.4	17.7	12.5	13.1	101.3

Class	Recall rate (%)	Precision rate (%)	F1 (%)
1	58.2071	16.7697	26.0378
2	65.3001	61.7069	63.4527
3	40.0475	53.6197	45.8503
4	53.3227	73.913	61.9518
5	24.1414	42.3009	30.7395
6	51.1616	77.9231	61.7683

Classification rate = 49.83%

Noisy Data-balanced tree based classification

20.9	10.5	15.5	11.2	13.5	7.6
17.9	100.9	14.2	12.6	12	10.7
19.5	19.1	71.3	16.7	14.4	27.3
12.5	17.2	15.5	111	11.3	20.6
18.3	12.6	14.2	7.1	34.2	12.6
11.4	13.4	17.9	16.6	16	122.7

Class	Recall rate (%)	Precision rate (%)	F1 (%)
1	26.3889	20.7960	23.261
2	59.9525	58.0887	59.0058
3	42.3648	47.9812	44.9984
4	59.0112	63.3562	61.1065
5	34.5455	33.7278	34.1317
6	61.9697	60.8933	61.4268

Classification rate = 51.17%

We can see that when using both strategies to determine the class, Class 4 (happiness) seems to have higher recognition, and Class 5 (sadness) is what gets recognised least and is mostly confused with anger. This may also be due to the fact that we have plenty of data for class 4 and not so much for class 5.
From the performance analysis of noisy data, we can see that again class 4 is recognised with high precision when using balanced tree classification and class 2 (disgust) is recognised with more precision when using length based classification. Both strategies have difficulties in identifying class 1 and class 5.

Noisy-Clean Datasets Question

There is a small performance difference between clean and noisy dataset, in particular with noisy dataset, the decision tree seems to have more difficulty in identifying anger This may be due to the fact that noisy data are missing some attributes, and these may play a key role in identifying the specific emotion. Classification rate is dependant on the data quality and the algorithm used, and since the algorithm is fixed, we know that the noisy dataset has not truly captured the actual attributes needed to distinguish between the emotions.

We can compare the performance for each emotion using clean and noisy dataset using F1 value, since it takes a weighted average of both recall rate and precision value. Anger in the clean set has an F1 value of 44.79%, where as in the noisy dataset this value reduces significantly to 26.04%, which implies that the accuracy almost halves when using the noisy data. However with disgust, the F1 value increases from 58.87% to 63.45%, this is quite surprising, knowing that the noisy data is not truly representative of the actual data since it was collected by an automated system, which is prone to have errors. Fear's F1 value has reduced from 57.74% to 45.85%. Happiness's F1 value has also reduced from 75.89% to 61.9518%. Sadness from 42.37% to 30.74%. Surprise from 68.97% to 61.7683%. These values are expected since the reduction is not too great and given the fact that noisy data is not completely accurate.

Ambiguity

testTrees takes a matrix of 6 trees, and another matrix of data to be classified. It then classifies each of the data, to exactly one emotion. For each row in data (*rd*) we check if each of the trees classifies *rd* as positive or negative, implying whether *rd* can be classed as belonging to the emotion. If there is only one emotion classified as being true for *rd*, then we simply output that result, however the true problem arises when there is multiple emotions that identify *rd* as being positive, or none of the emotions classify *rd* as being positive. If none of them classify *rd* to be positive, we create a list of all possible emotions and pass it to a function *classify* which chooses one emotion, else we pass the classified emotions to *classify.*

Classify takes a list of possible emotions for a specific data and the list of trees, that classified it. It then uses a strategy to choose exactly one class. We chose two strategies and determined the best one. Our first strategy was to give preference to balanced trees, this is because a balanced tree handles all possible values of the attributes equally. The advantage of using this method is that if the dataset used to train the tree was vast and balanced, then the tree would adapt to every situation, and would therefore positively identify future cases with high probabilities. However checking if a tree is balanced, is pretty intensive as we need to traverse through every node (and leaf) and check if that is balanced too. Also, determining the maximum depth difference between trees for them to be balanced is also, quite tricky to pick, since too low of a depth, would lead most trees being identified as being not balanced, whereas too high of a depth would incorrectly identify an unbalanced tree as being balanced. We chose 2 as our maximum depth limit, since this accounts for both situations.

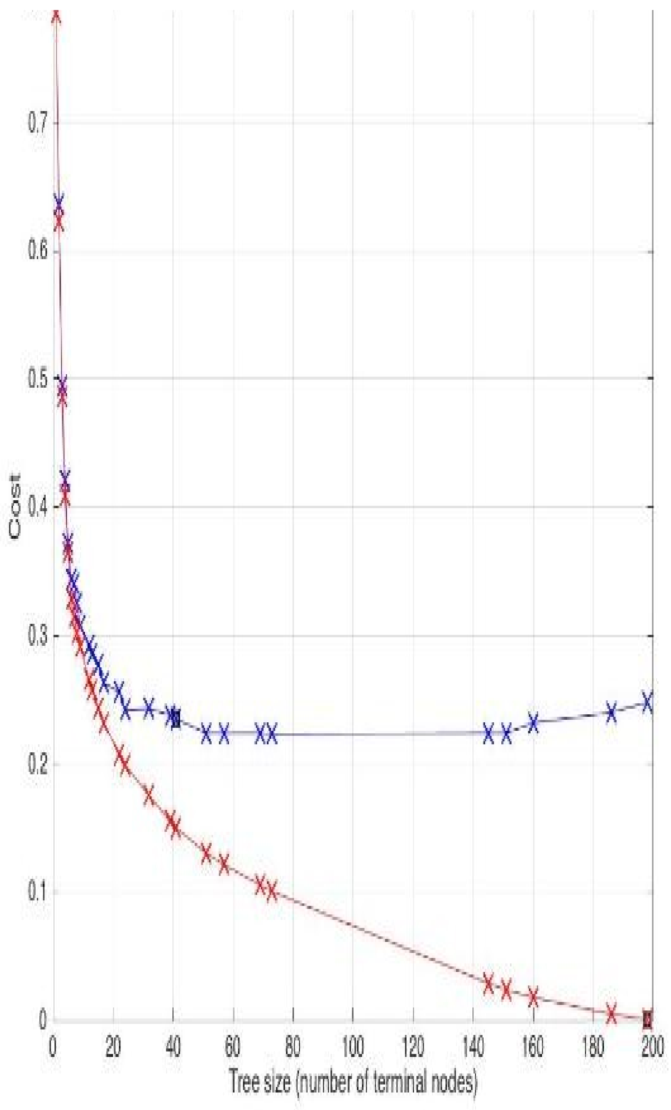
Our alternative strategy was to give preference to shorter trees. This follows Occam's razor that shorter hypothesis, is less likely to be a coincidence, and therefore if a shorter tree classified a row vector to be positive it is likely to be more accurate, than a longer tree, due to the fact that the longer tree is likely to have been overfitting to the data. The disadvantage with this approach is that if the training data contained inconsistent data, then the tree would quickly become long depending on the number of attributes present. We compared the performance of both of these strategies and their results are presented above. We compared the average F1 value for both strategies with clean and noisy data. The average F1 value for clean data when using length strategy was 58.1, whereas balanced strategy had a F1 value of 57.6, also for noisy data length strategy; F1 value was 48.3 and for balanced strategy it was 47.3. Although the difference is relatively small, length based strategy performs better, than balanced based strategy. This was expected, since a shorter tree that does indeed classify the data as being true is likely to be more reliable than a long tree classifying the same tree as being true.

Pruning

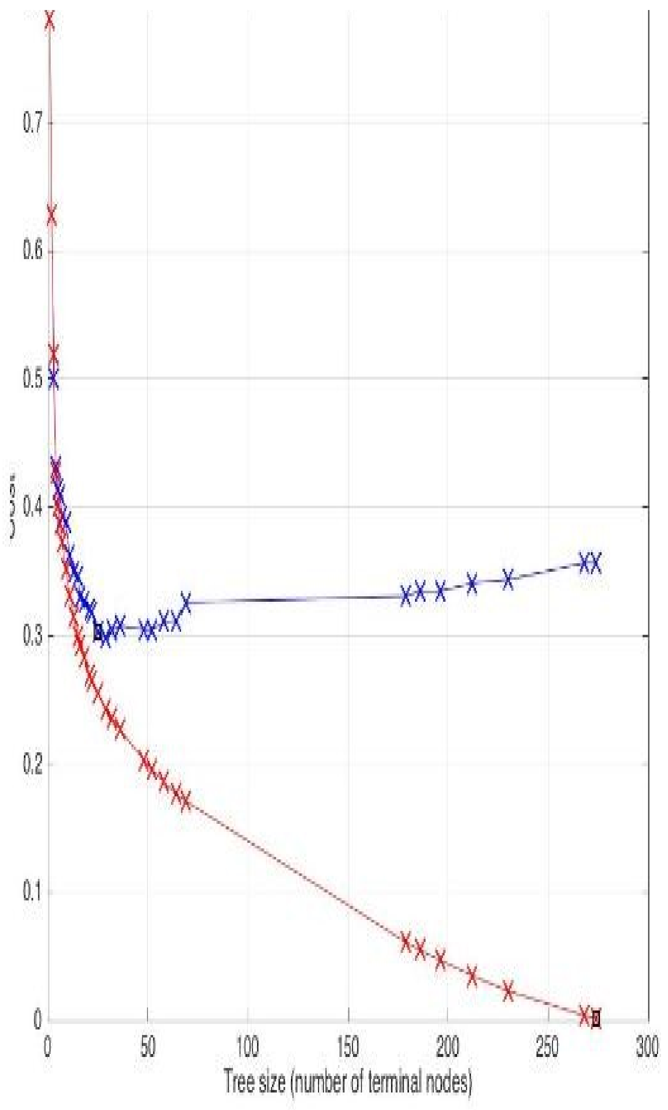
The goal of the pruning_example function is to apply different techniques to reduce the size of each of the classification trees and to avoid over-fitting. The function first creates a classification tree using the classregtree function. It then calls the test function to return a vector of costs related to two of the testing methods (cross-validation and resubstitution). The tree is pruned to create two different pruned trees to the best level (number of nodes) for both cross-validation and the resubstitution methods. These best levels were also returned by the test function. Finally a graph is plotted for the data (as shown above) showing the costs associated with the tree size for the two testing methods with the best levels marked by black squares.

From the graph it is clear to see that there is a difference in the costs associated with pruning based on the two methods (cross-validation and resubstitution). For clean data, the best level based on the cross-validation method is a tree size of approximately 40 terminal nodes, whereas for resubstitution the best level is around 200 terminal nodes. For the noisy data, the best level to prune to is approximately 25 nodes based on cross-validation and around 275 nodes based on the resubstitution method.

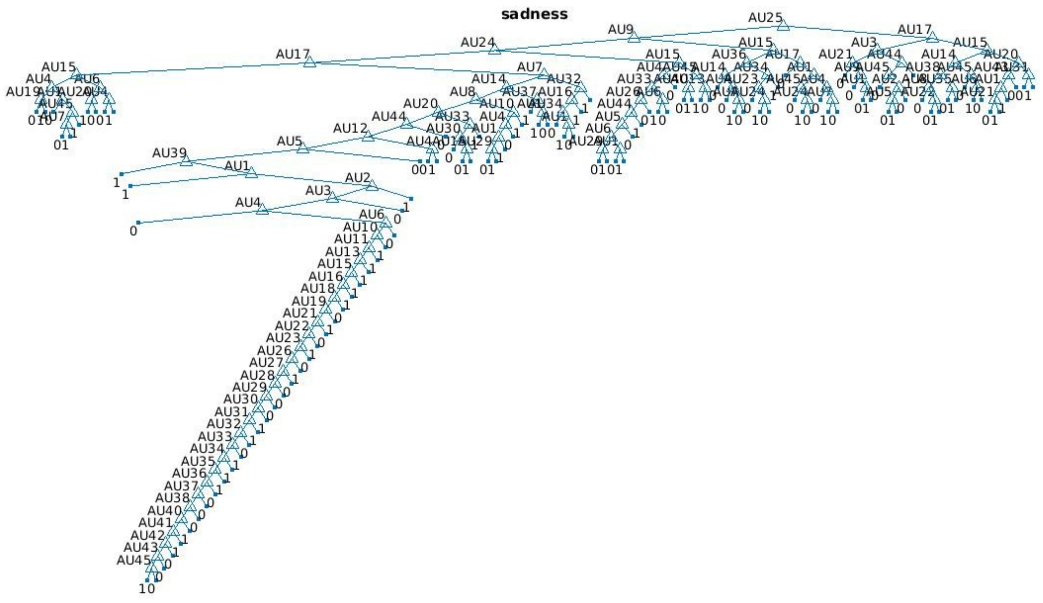
Clean

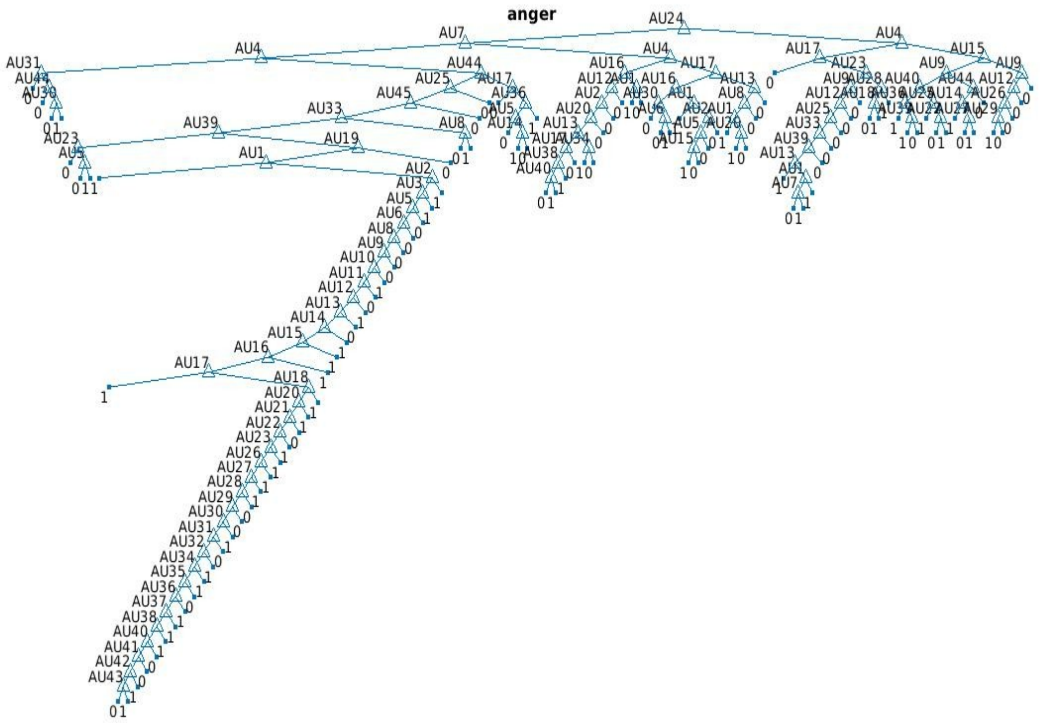


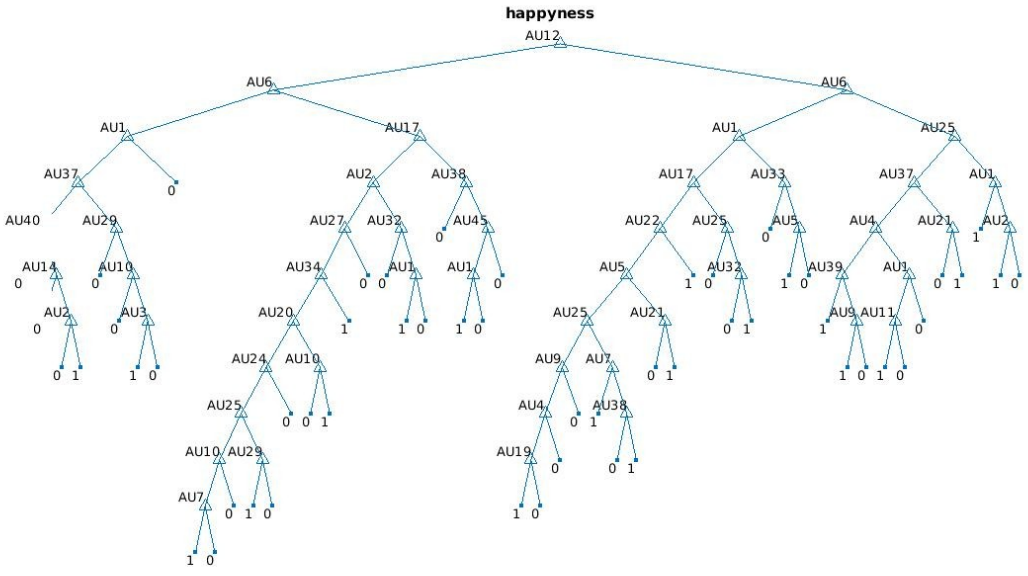
Noisy

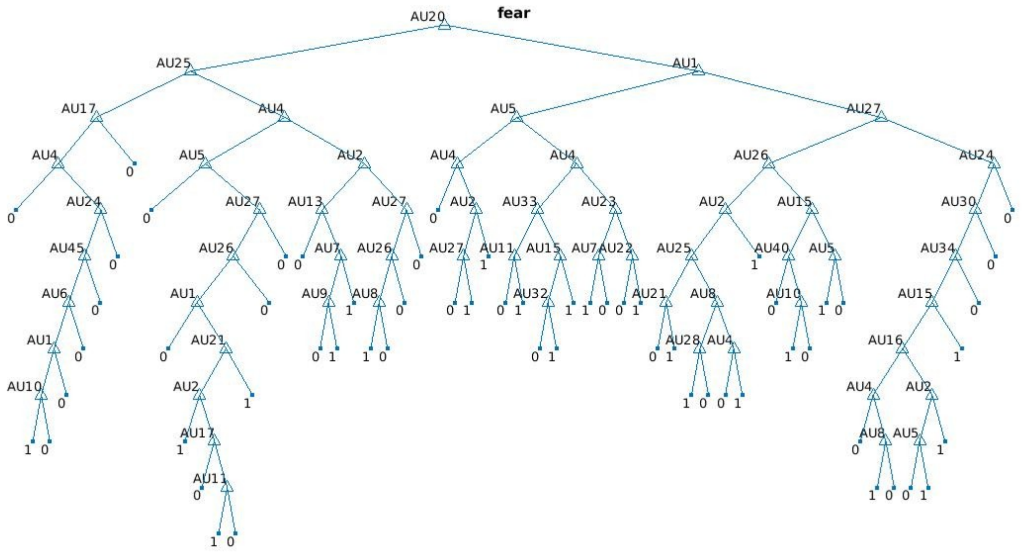


Decision Trees

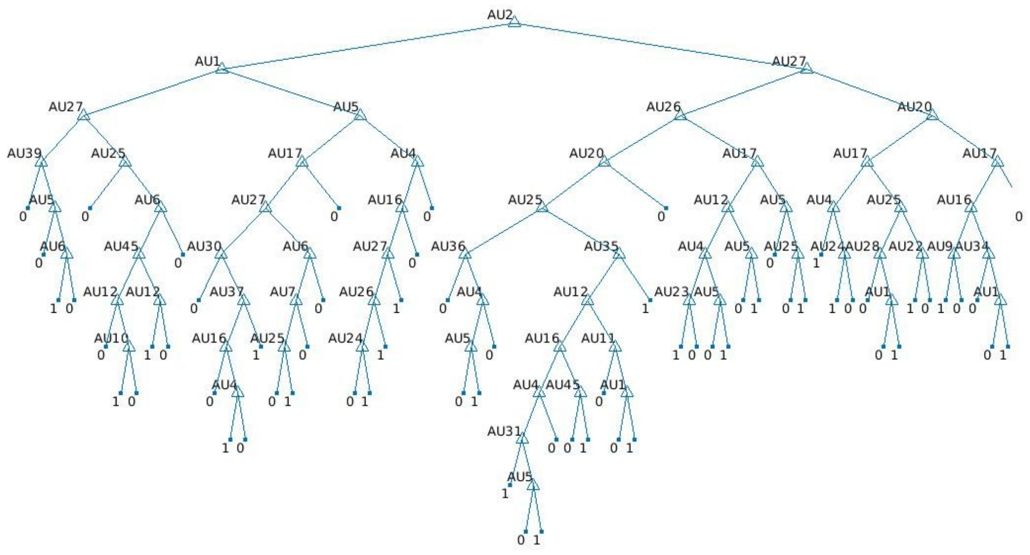




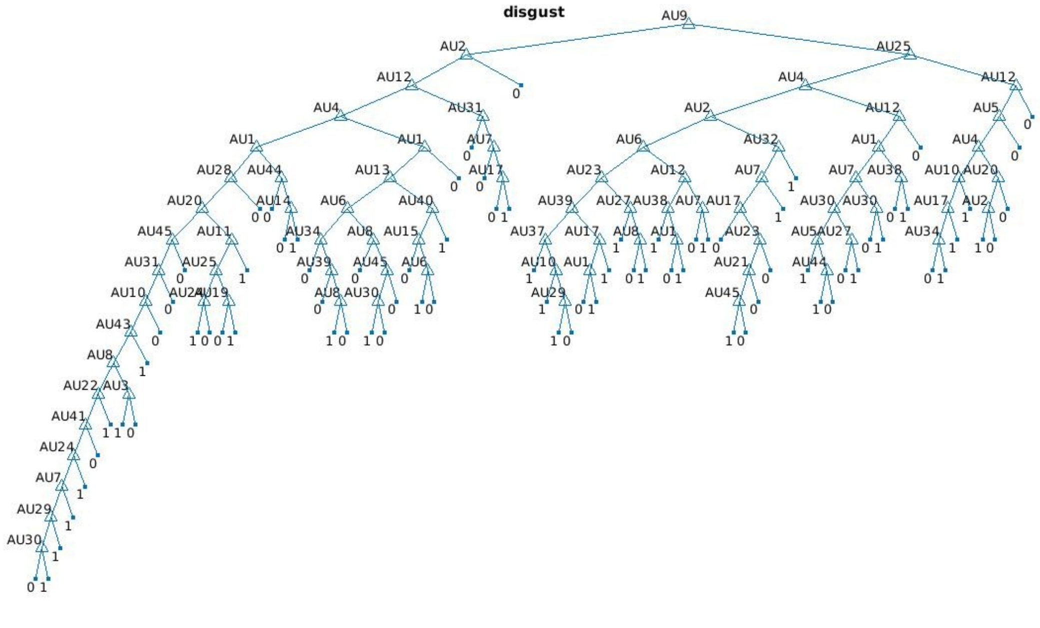

happyness
AU12
AU6
AU6
AU1
AU17
AU1
AU25


fear
AU20
AU25
AU1
AU17
AU4
AU5
AU27

surprise


AU2
AU1 AU27
AU27 AU5 AU26 AU20
AU39 AU25 AU17 AU4 AU20 AU17 AU17 AU17
AU5 AU6 AU27 AU16 AU25 AU12 AU5 AU4 AU25 AU16
AU6 AU45 AU30 AU6 AU27 AU36 AU35 AU4 AU5 AU25 AU24 AU28 AU22 AU9 AU34
AU12 AU12 AU37 AU7 AU26 AU4 AU12 AU23 AU5 AU1 AU1
AU10 AU16 AU25 AU24 AU5 AU16 AU11
AU4 AU4 AU45 AU1
AU31
AU5

disgust


AU9
AU2 AU25
AU12 AU4 AU12
AU4 AU31 AU2 AU12 AU5
AU1 AU1 AU7 AU6 AU32 AU1 AU4
AU28 AU44 AU13 AU17 AU23 AU12 AU7 AU7 AU38 AU10 AU20
AU20 AU14 AU6 AU40 AU39 AU27 AU38 AU7 AU17 AU30 AU30 AU17 AU2
AU45 AU11 AU34 AU8 AU15 AU37 AU17 AU8 AU1 AU23 AU5 AU27 AU34
AU31 AU25 AU39 AU45 AU6 AU10 AU1 AU21 AU44
AU10 AU24 AU19 AU8 AU30 AU29 AU45
AU43
AU8
AU22 AU3
AU41
AU24
AU7
AU29
AU30