Machine Learning CW1

Amna Askari [asa713, 00835959], Alexander Camuto [ac4132, 00736863],
Max Poynton [map213, 00828141], Arshan Shafiei [as2413,00826335]

13/02/2017

1 Implementation

1.1 Loading Data

Initially we are given a set (x,y), with x being a set of examples indicating whether or not each of 45 facial Action Units (AUs) are present in that example, and y the corresponding emotion for that example. In order to train a tree for a given label, we need to prepare the positive and negative examples for that particular emotion. This is done by the following boolean operation:

$$y\_emotion\_j = (y == j),$$

which takes as input the set of example labels (y) and replaces the labels that match j with 1, and the rest with 0. The result is passed to the decision tree learning function in the next step.

1.2 Creating the Decision Tree

Our tree is essentially a recursive Matlab struct that represents a node and it's subnodes.

$$tree = struct('op', [],' kids', [],' class', []);$$

The members of each node (op,kids and class) are allocated during execution, recursively building the tree until the label for the leaf node (classification result) is hit. The ChooseBestDecisionAttribute function is worthy of particular note; it is important for the operation of the algorithm that the attribute which results in the highest information gain is chosen for each progressive root node. The gain is calculated as shown below.

- In order to find p1, first the indices of those attributes having the value 1 are found. Then, the sum of the binary targets at those indices is calculated. This will essentially be the number of positive examples (with attribute 1), since only those containing 1 will count towards the sum.
- The same is done for p0. n1 and n0 are simply what remains of the indices when subtracting p1 and p0 respectively.
- Using the given equations, we calculate I, the *Remainder* and find the gain for that attribute.
- By replacing the current highest gain with any higher gains each iteration, the attribute with the highest gain is the result.

```
for i=1:length(attributes)

     indices = find(examples(:,attributes{i}) == 1);
     p1 = sum(binary_targets(indices));
     n1 = length(indices) - p1;

     indices2 = find(examples(:,attributes{i}) == 0);
     p0 = sum(binary_targets(indices2));
     n0 = length(indices2)-p0;

     p0n0 = (p0+n0)/(p+n+eps);
     p1n1 = (p1+n1)/(p+n+eps);

    Remainder = p0n0*Inp(p0,n0)+p1n1*Inp(p1,n1);
    Gain = Inp(n,p) - Remainder;
    if ~exist('previous_gain','var')
        previous_gain=Gain;
        best = attributes{i};
        bestIndex = i;
    elseif Gain>previous_gain
        previous_gain = Gain;
        best = attributes{i};
        bestIndex = i;
    end
end
```

Figure 1: Gain function

Trees trained on the entire clean dataset are found in trees.mat. The corresponding index for each tree, trees(1) for instance, matches the index of each emotion, anger in this instance.

1.3 Evaluation

The above method generated the 6 trees when trained with the entire clean dataset as shown in section 1.2. To evaluate the result, the following steps were taken:

1. A 10-fold cross-validation of data was performed, with each fold used to train and test a new set of 6 decision trees (one for each emotion). At each fold, 100 samples of data were used for testing, the remaining samples were used to train the trees. The 100 samples used for testing were determined using the following equation:

$$test\_samples = samples(1 + (i-1) * 100 : i * 100)$$

   where i is the i^{th} fold.

2. Each fold's training data was passed to the *testTrees* function, which generates an Nx*1* matrix consisting of N examples and their corresponding predictions. As part of this function, the trees were combined to generate a single prediction output for that example (more details in section 4.2).

3. Each fold's test data was used to generate a confusion matrix.

4. The sum of confusion matrices, across the 10 folds, was used to generate an average confusion matrix, for clean and noisy test data.

5. The rows and columns of the resultant matrix were used to calculate the average recall rates and precision rates for each emotion.

6. These rates were then used to calculate the F_1 measures, to determine the quality of the resultant classifiers.

The resulting confusion matrices, for noisy and clean data, and corresponding statistics are detailed below.

2 Evaluation Analysis

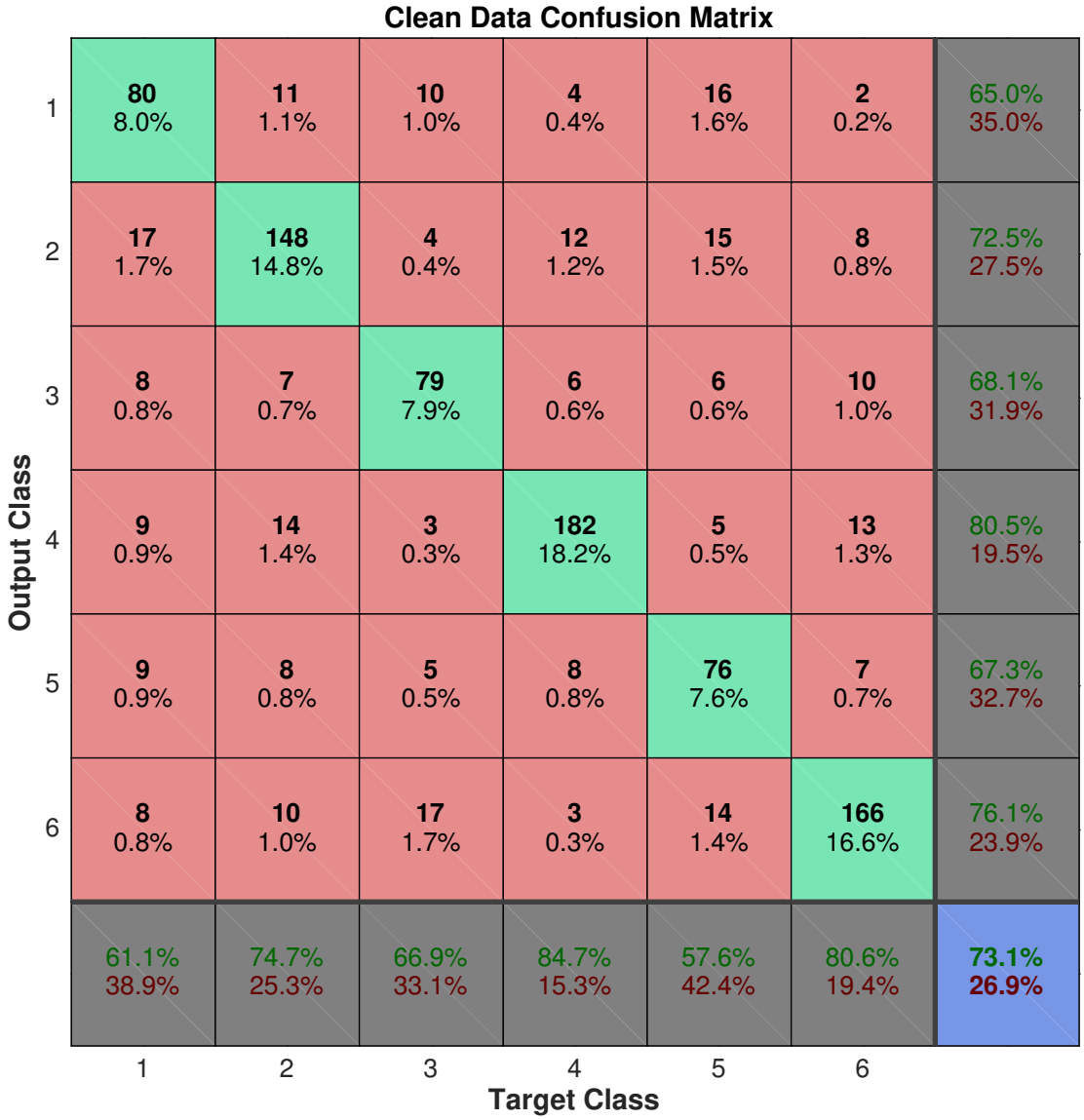

Figure 2: Average confusion matrix for the six decision trees trained on clean data during 10-fold cross-validation. Red squares represent misclassified test data, green squares represent correctly classified data. The final gray column contains the Recall Rates (in green) for each class. The final gray row contains the Precision Rates (in green) for each class. The blue square indicates the overall accuracy of the classifier. Percentages are for the 1000 samples of the 10 fold cross validation.

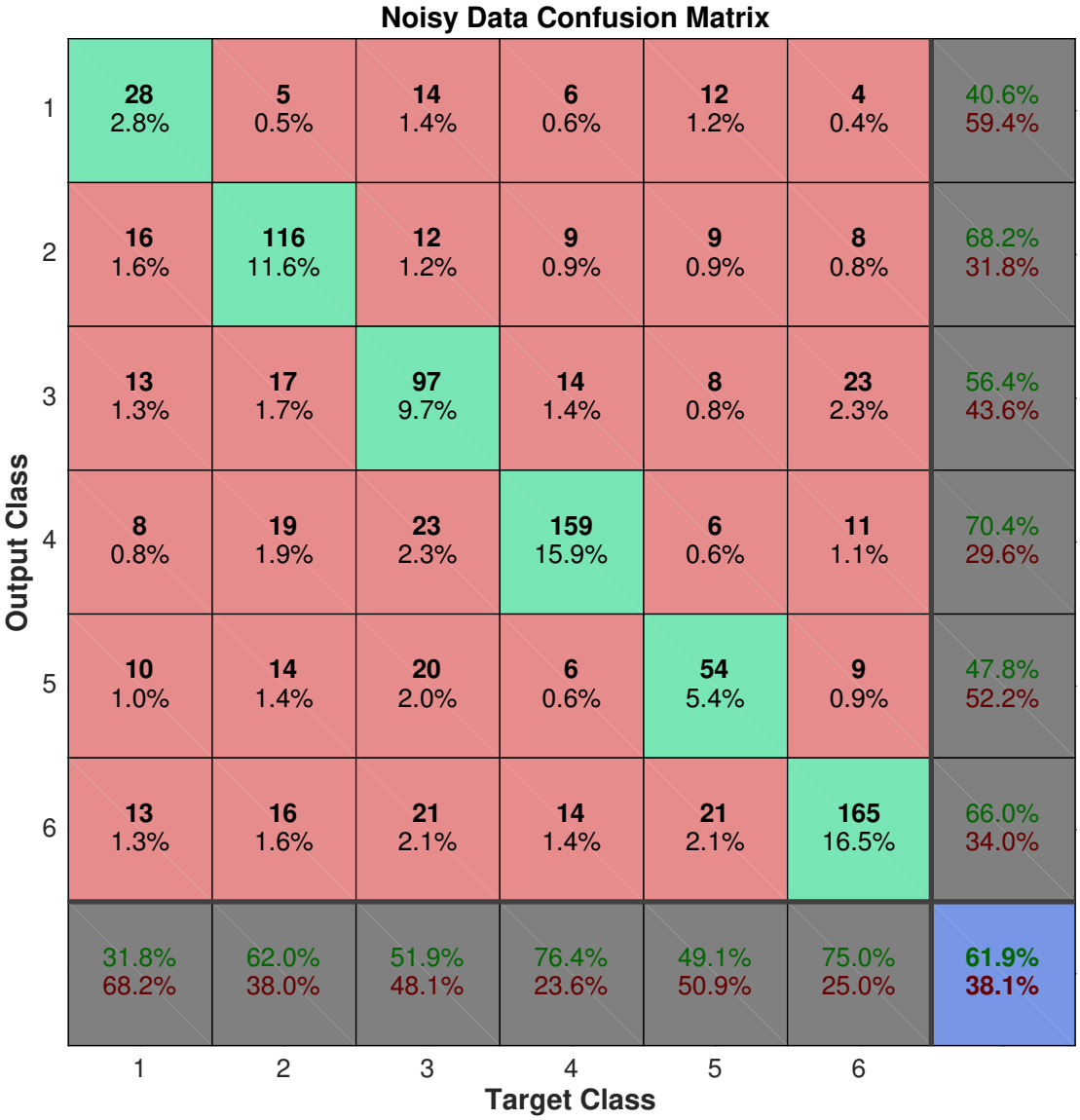

Figure 3: Average confusion matrix for the six decision trees trained on noisy data during 10-fold cross-validation. The representation is identical to that of the matrix described in figure 2.

	Anger	Disgust	Fear	Happiness	Sadness	Surprise
F_1 (clean)	62.9%	73.40%	68.7%	83.4%	63.8%	78.2%
F_1 (noisy)	40.0%	53.2%	55.5%	74.9%	45.8%	74.5%

Table 1: F_1 values, for each emotion, for classifiers trained on clean and noisy data. The mean of the F_1 values for clean data is 71.73. For noisy data the mean is 57.31.

3 Tree Diagrams

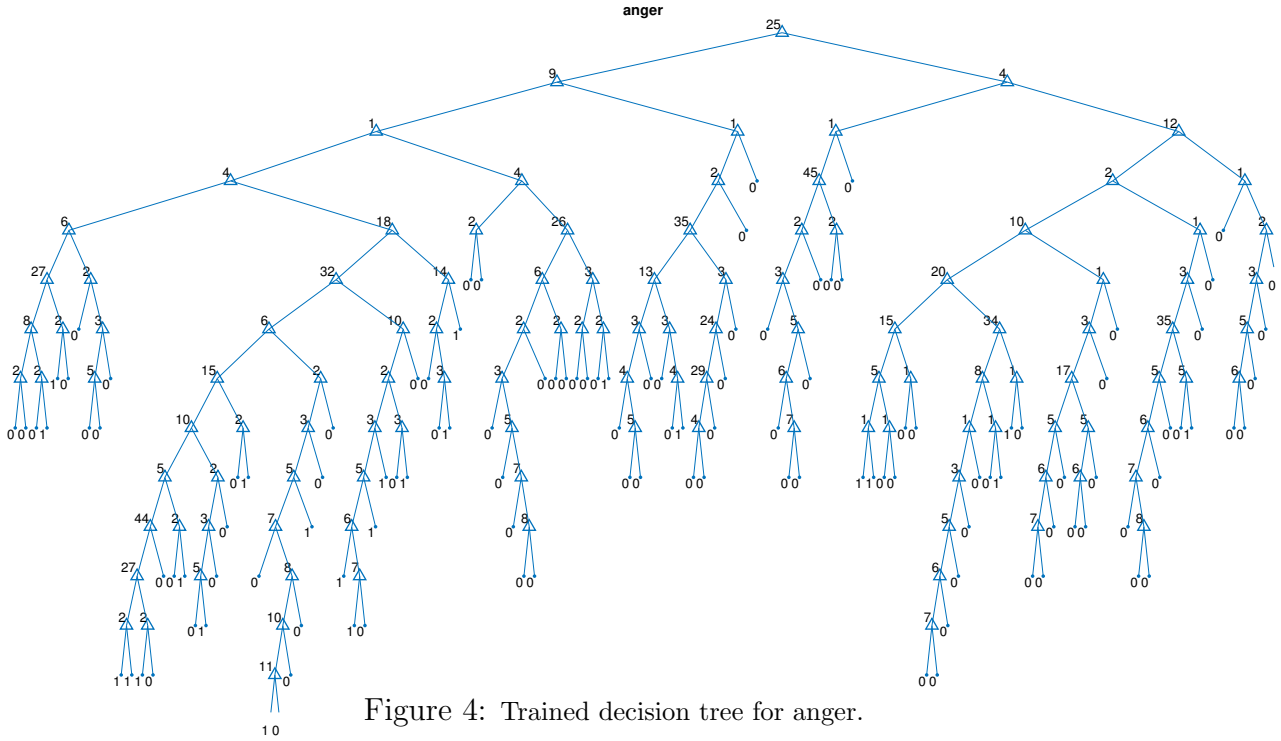

Figure 4: Trained decision tree for anger.

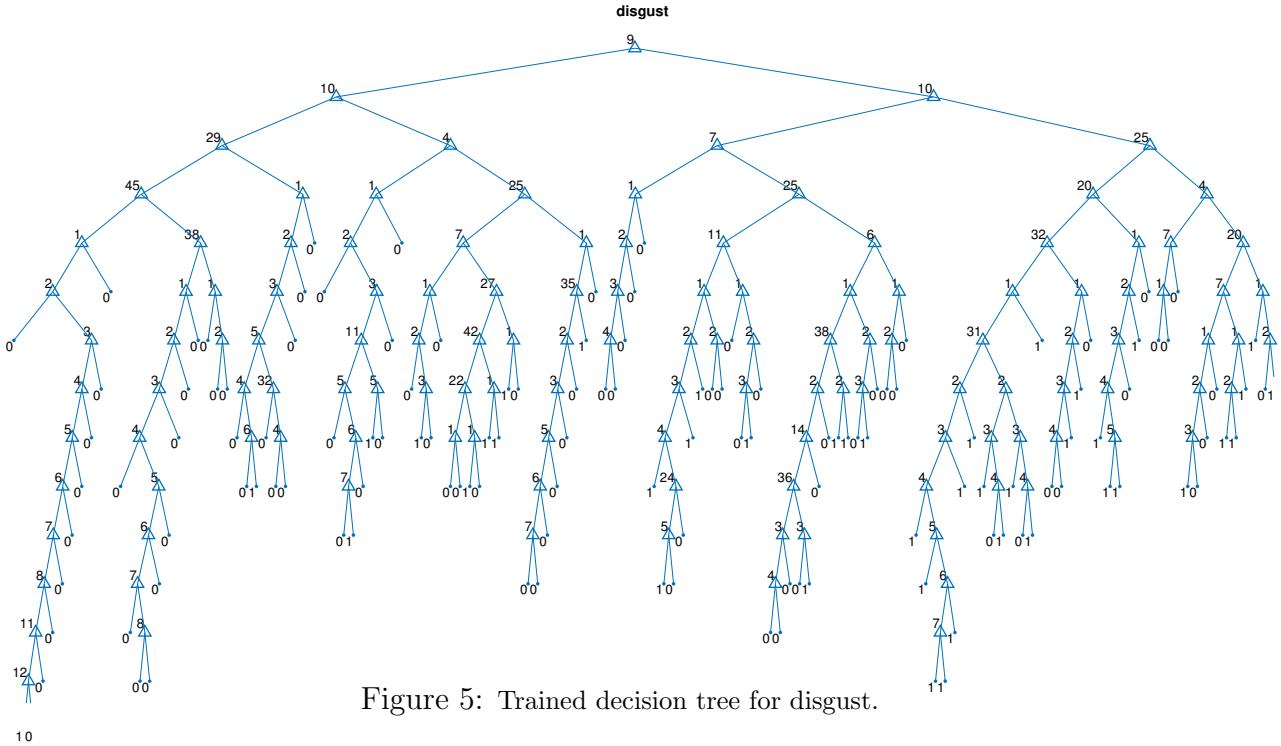

Figure 5: Trained decision tree for disgust.

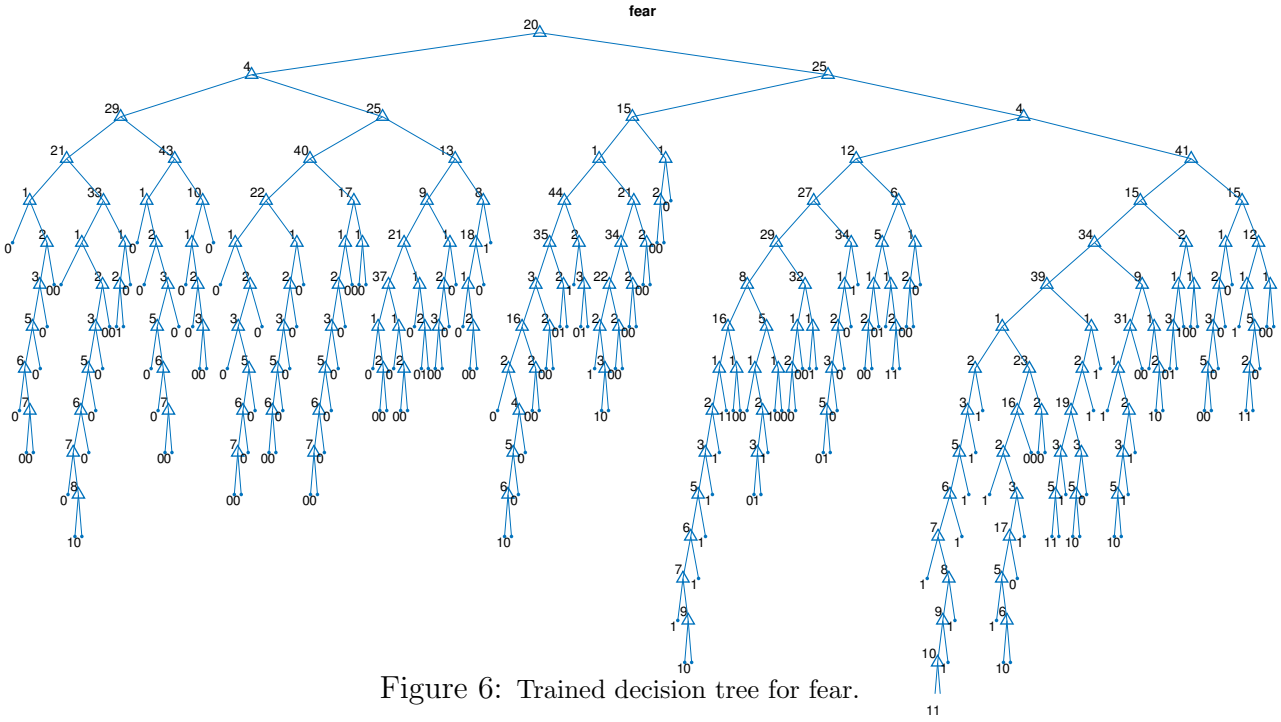

Figure 6: Trained decision tree for fear.

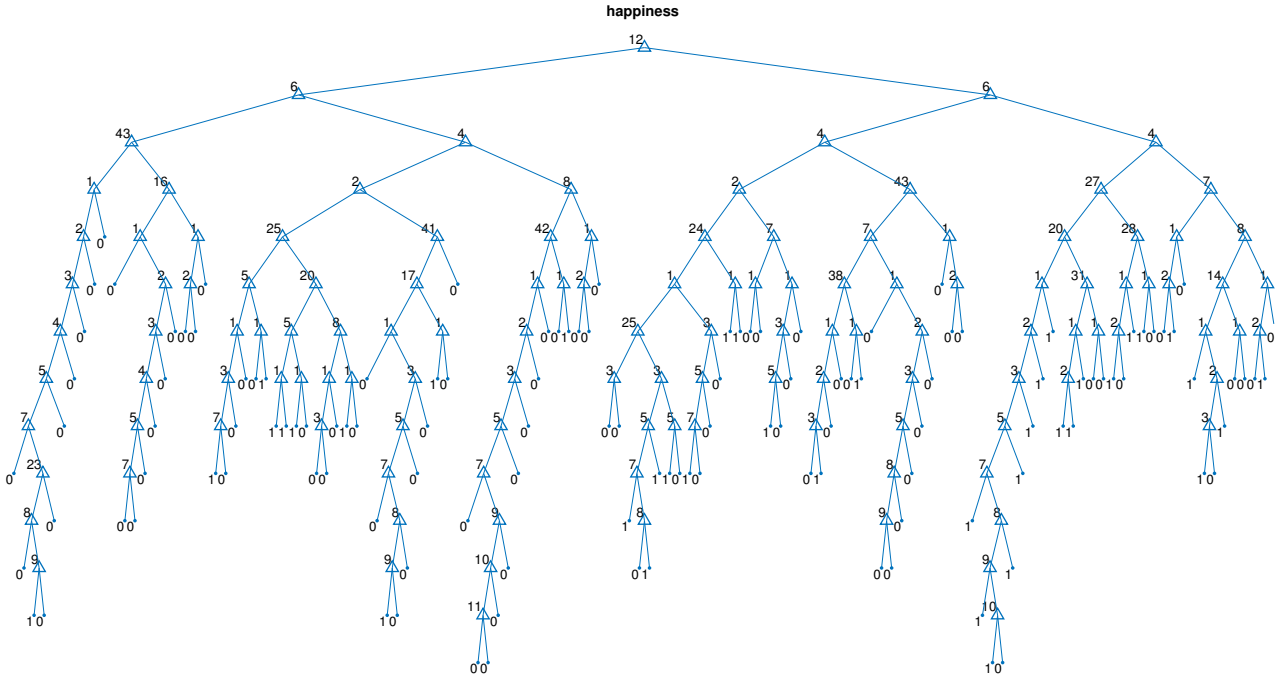

Figure 7: Trained decision tree for happiness.

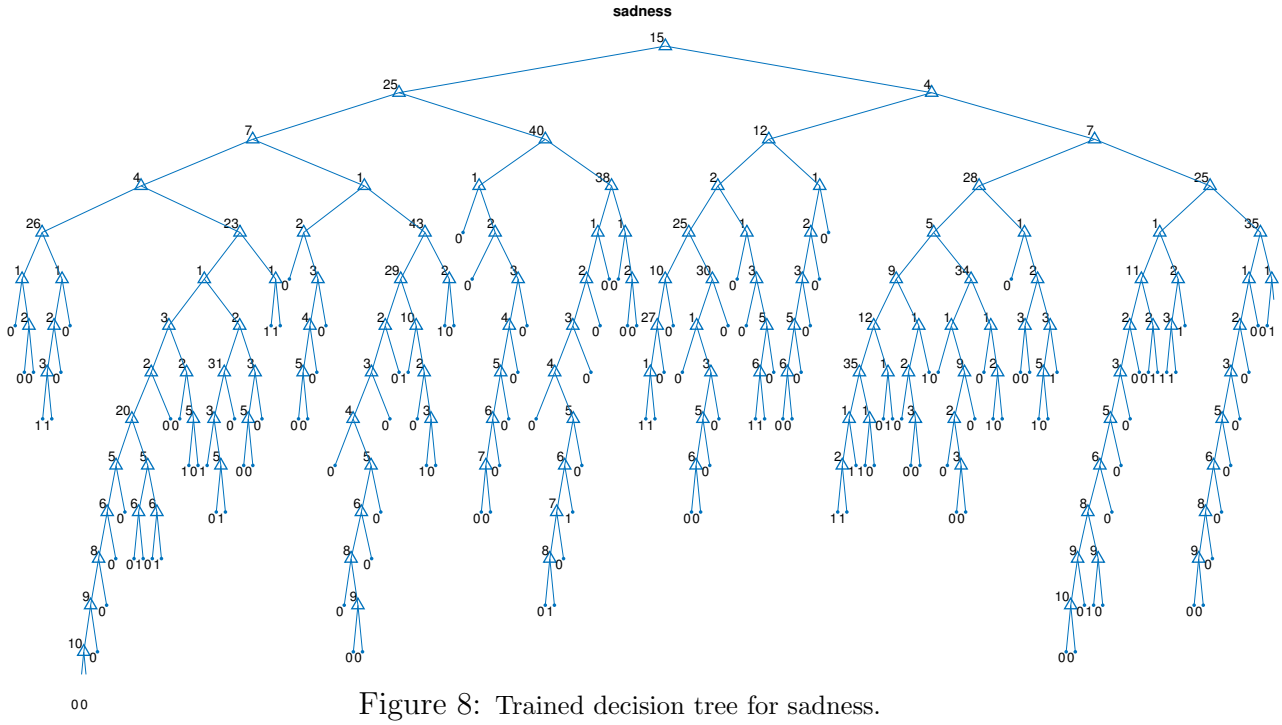

Figure 8: Trained decision tree for sadness.

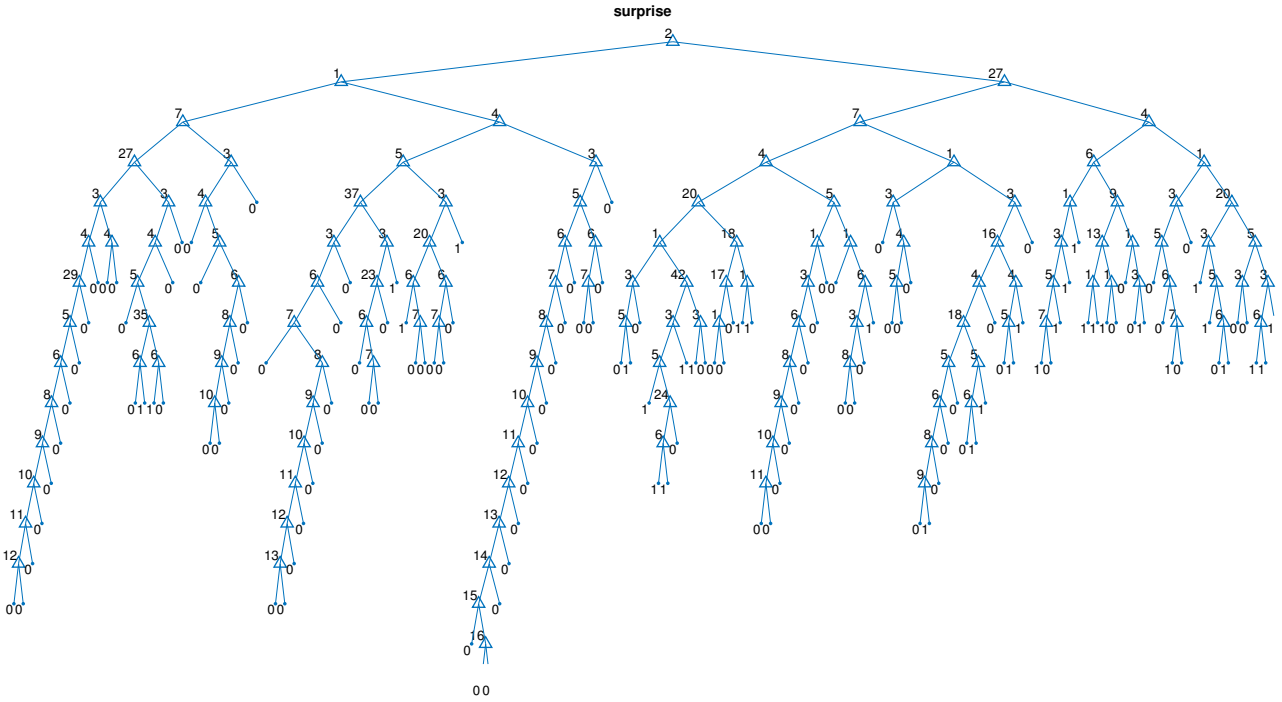

Figure 9: Trained decision tree for surprise.

4 Questions

4.1 Noisy vs Clean Datasets

With reference to the confusion matrices in Fig 1.2 and Fig 1.3, there is a clear observable difference in the performance of the algorithm with the clean and noisy datasets. The overall classification rate is 11.2% higher for the clean dataset (with the same sample size), highlighting the fact that a classifier is only as good as the data it is trained with. This can be further confirmed by comparing the diagonal entries in both confusion matrices: the amount of correctly classified examples for each emotion is lower for the noisy data. The percentage differences are not significantly low and vary across the emotions. To understand why, it is important to look at the nature of noise in classification datasets. Noise in the dataset can either affect class values or attribute values. Noise in class values results in either mislabeled classes or contradictory classes in which attributes with the same combination of values result in two different class labels. Noise in attributes can either be in the form of erroneous values i.e. a 0 when there should be a 1, missing values or 'don't care' values which are those that do not affect the final result but still corrupt the sample. Hence with the same classification algorithm on a noisy dataset, it is more likely to get classifications wrong because either the reference label in the binary vector could be incorrect or the combinations of attributes in the example could be misleading - resulting in an incorrect calculation of information gain. Looking at the individual emotions:

1. Anger - Although both the precision and recall rates are lower for the noisy dataset, the number of mis-classifications in the noisy data set are not all significantly higher than in the clean dataset. The false positive are in some cases actually lower for the noisy dataset. However false positive rates are almost all higher for the noisy dataset.

2. Disgust - The precision and recall rates are both lower for the noisy data set as expected. Again, this doesn't imply that the number of mis-classified examples or incorrect predictions is necessarily higher for each emotion in the noisy dataset. Like in the case of anger, there is a greater difference in the recall rate as compared to the precision. This implies that the type of noise leads to more false negatives than false positives.

3. Fear - Despite the surprisingly low classification rate, the recall and precision rates for Fear are both higher in the clean dataset. False negative and false positives are more prevalent in the noisy dataset in comparison with the clean dataset.

4. Happiness - There are - a few outliers in both false positive and false negative rates in this case, despite the higher number of correct classifications, but two of them are greater by just 1 entry and the other one by 3 entries. Overall, there are more incorrect classifications in the noisy dataset leading to lower recall and precision rates.

5. Sadness - In this case, the red column had a few outliers; the noisy dataset had fewer false negatives in comparison to the clean dataset. However, the red row values in the noisy dataset were higher than that in the clean dataset as expected. This difference was unusually not expressed in the recall and precision rates, which were both lower in the noisy dataset as expected.

6. Surprise - The recall and precision rates are higher in the clean dataset as expected, with most false negative and false positive rates being higher in the noisy dataset. Precision is similar to that of the clean data however, and when compared to the other precision rates for the emotions we can assume that noise has affected 'surprise' data points to a lesser degree. Misclassification of other data points however lead to an increase in false positives for 'surprise', decreasing the recall rate markedly.

There are no particular emotions that are more likely to be misclassified in both the noisy and clean datasets. Noise is uniformly distributed for all classes except 'surprise', for which the classifier retained high precision.

4.2 Ambiguity

Two approaches were taken to solve the question of ambiguity, i.e when several emotions were assigned to a data point, or none were assigned at all. When no emotions were picked, our first approach was to randomly assign a classification, giving each emotion an equal probability of being chosen. The second approach was to assume that the data was stationary, i.e that the probability distribution of classes is constant for each data point. The resultant probability for each emotion was used to pick an emotion at random. If 1/3 of predictions were 'happy' then there was a 1/3 chance a test sample, which originally had no labels, would be attributed the 'happy' label. When several emotions were assigned to a label, similar approaches were used. One of the labels could be picked with equal probability for each. If there were three labels assigned to a data point for instance, one of the three labels would be picked with probability 1/3. Another approach was to weight these probabilities with the number of occurrences of each class in the test data set. The probability to pick a given class was given by the following formula:

$$p_i = \frac{\sum_{n=0}^{N}(pred == class_i)}{\sum_{j \subset M}\sum_{n=0}^{N}(pred == class_j)}$$

where M is the set of classes attributed to a data point, $pred$ is the set of predictions generated by the test set, and N is the number of training samples.

Again this approach assumes stationarity in the data. The equal probability choices will a have low probability, about 16.7%, of attributing the correct emotion to a data point that has not been classified. For most classifiers this drags the accuracy down significantly. The choices based on the probability distributions will increase the accuracy of the classifier on a given dataset but may amplify erroneous classifications. If one classifier is weak, for anger for instance, then its erroneous classifications will distort the probability distribution of emotions. Additionally the classifier will not perform as well on non stationary data sets where the probability distribution is not constant throughout the set. Figures 2 and 3 detail the confusion matrix given by the probability distribution approach. Table 1 describe the F_1 values for the probability distribution approach. For the equal probability approach, overall accuracy is 68.4% for clean data and 57.6% for noisy data. The following table describes the F_1 values for the equal probability approach.

	Anger	Disgust	Fear	Happiness	Sadness	Surprise
F_1 (clean)	69.6%	80.5%	65.9%	87.6%	72.0%	81.6%
F_1 (noisy)	44.6%	66.1%	57.9%	76.1%	49.5%	78.9%

Table 2: F_1 values, for each emotion, for classifiers trained on clean and noisy data. The mean of the F_1 values for clean data is 76.19. For noisy data the mean is 62.16.

Notice that although the accuracy of the classifiers has decreased, the mean F_1 values increase significantly with the equal probability approach. This testifies to this approach's ability to limit overfitting and produce higher quality classifiers.

4.3 Pruning

The pruning_example function demonstrates the effect of reducing the size of the decision tree by removing nodes which add little to the tree's ability to classify examples. Furthermore, the act of pruning helps reduce the problem of overfitting; by removing nodes which are not essential to classification, the process minimises the possibility that the tree contains so many nodes that it is so well trained to a certain set of data that it is unable to generalise to further testing data.

The function first generates a classification tree from the provided x and y datasets. Subsequently, it performs performance tests on the tree. These tests involve iteratively pruning the original tree to different levels and calculating a "cost" for each of these trees. The cost function used depends on the type of test; in this case two types "cross-validation" and "resubstitution" are used.

The test function returns the number of leaf nodes for each pruned version of the original tree (nodes) and a vector of cost values for each subtree (cost). It also returns the level up to which the original tree was pruned which gives the best result, where "best" is given by a statistical measure (bestLevel). The tree is then pruned using these bestLevel estimates.

The pruning_example function plots the number of terminal nodes of each subtree against the cost. Finally, it plots the number of leaf nodes of the smallest tree within one standard error of the subtree with the minimum cost as a square as shown in the following diagrams:

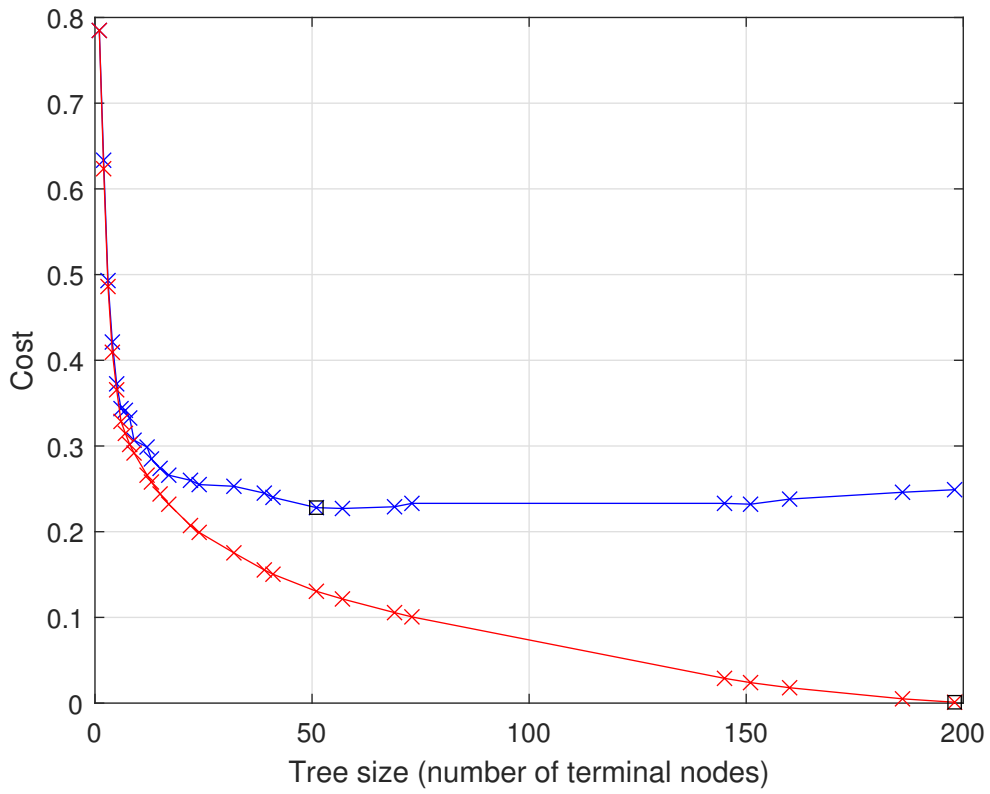

Figure 10: Pruning costs for clean data. With crossvalidation method in blue, resubstitution method in red.

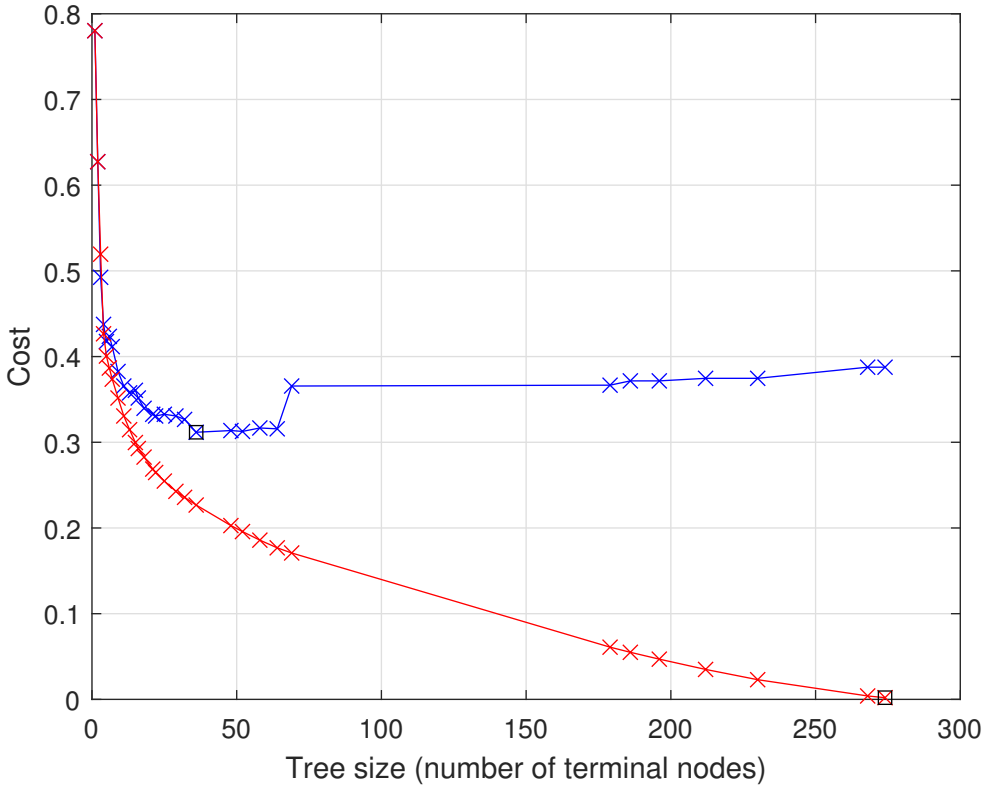

Figure 11: Pruning costs for noisy data. With crossvalidation method in blue, resubstitution method in red.

For both datasets, the curves produced when using resubstitution are roughly inverse exponential, as are those when using cross-validation up to a point where the cost begins to increase again. There is likely overfitting when using resubstitution because the same data is being recycled. This is not shown on the graph as it is not tested with new data.

All four curves follow the general trend that the cost of the subtree decreases as the number of terminal nodes increases. This is likely because the larger trees are better fit to the data and are therefore less likely to give incorrect classifications, and in turn incur lower misclassification costs.

Since the cost for a tree is related to the sum of the misclassification costs of the nodes in that tree, this leads to the graphs showing generally lower costs for the clean dataset than for the noisy dataset as the noise leads to more misclassifications. The cost when using cross-validation evens out at around 0.25 for the clean data, as opposed to around 0.4 for the noisy data. Similarly for resubstitution, at a subtree size of 50 terminal nodes, the clean dataset shows a cost of around 0.15 as opposed to the noisy dataset's 0.2.

For the clean dataset, the optimal tree size is $\sim$50 terminal nodes when testing using cross-validation, and $\sim$200 when using resubstitution. For the noisy dataset, the optimal tree size is $\sim$40 terminal nodes when testing using cross-validation, and $\sim$275 when using resubstitution.