Implementation Details

a) Best attribute

Our CHOOSE_BEST_DECISION_ATTRIBUTE function chooses one attribute based on the information gain one gets from a set of examples, using all remaining attributes and their corresponding binary targets for a given emotion. We calculate the information gain of all the attributes present in our set using calc_entropy and substract the result from calc_remainder from it. The attribute returning the highest information gain is then chosen. It is removed from the set of considered attributes in DECISION_TREE_LEARNING.

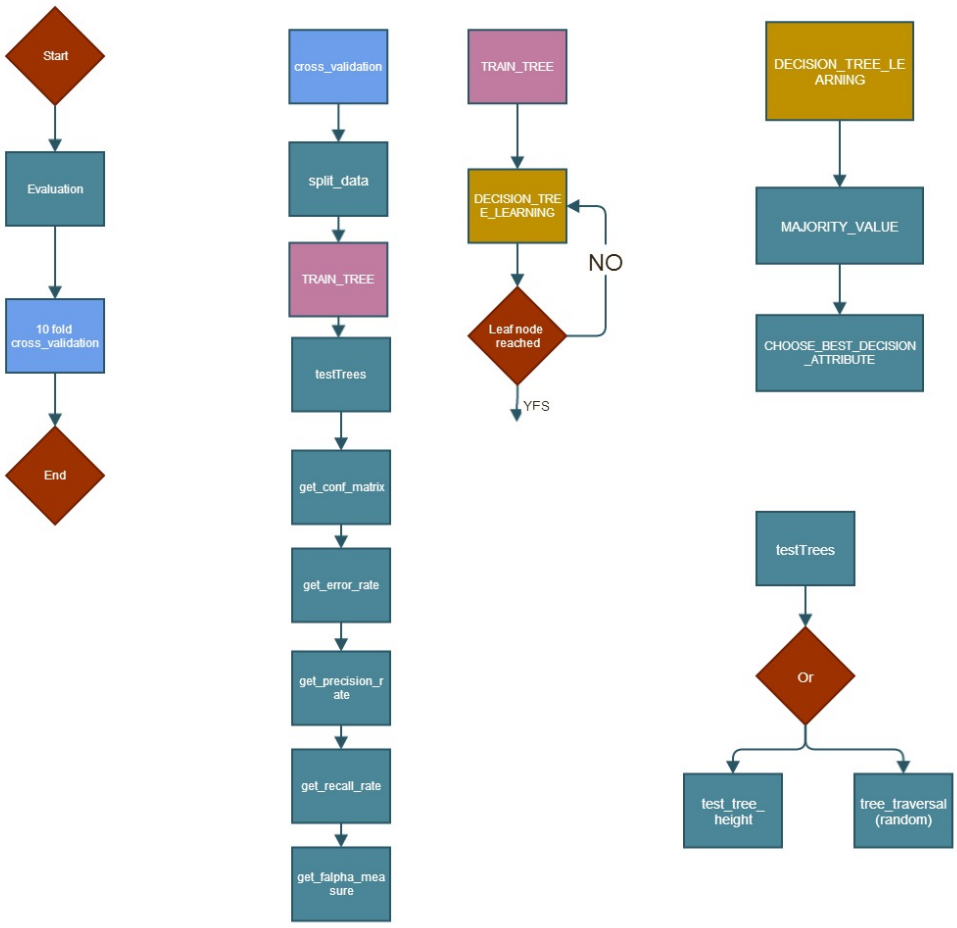
b) Decision Tree Learning

DECISION_TREE_LEARNING has been implemented based on the pseudo code given in the manual. We first check for special cases: for example, if all of the chosen examples have the same value of binary_targets, all we need to do is to create a leaf with that value. To do this, we look at the sum of the binary targets and compare their value to 0 or the length of the set. If the special cases are bypassed, then we need to look for the current best attribute. We therefore call CHOOSE_BEST_DECISION_ATTRIBUTE and start building the tree, using the first best_attribute as the root. We repeat this process, deleting the chosen best_attribute from the set of considered attributes, until we can decide whether an emotion is suitable for a given example no matter which branch we are on (we reach a leaf on every branch). We then use this whole function to construct our 6 emotion trees, trained over the all of our examples using 10-fold cross validation.

c) Cross Validation

cross_validation starts by splitting the example set in 10 folds, using split_data. It then selects examples based on the fold_num argument and creates a tree for each of the 6 emotions, using TRAIN_TREE on this particular set of examples. TRAIN_TREE takes one emotion and a set of examples to create a tree using DECISION_TREE_LEARNING and feeds it the the binary target vector of the given examples according to the emotion chosen. cross_validation then uses testTrees to compare our own predictions against the real results. Using these results, we calculate the confusion matrix, the classification rate, the precision and recall rates, as well as F1 measures.

d) Implementation flow chart



e) Combining the 6 trees

In order to get one-to-one predictions for our examples, we need to combine our 6 trees. We used a Random classification method and a Shortest Height classification method to achieve this. These are discussed in the Ambiguity section.

Questions

Noisy-Clean datasets

We assume that, when using clean data, the mapping from an example to a face as well as the set of attributes are always correct. When using noisy data, the mapping from each example to a face is always correct, however, the set of attributes corresponding to the example may not be. This causes problems with the performance of our predictions, as we base our algorithms on the content of these attribute sets: if the attribute sets have errors in them (missing or falsely added attributes), then we are likely to fail some attribute tests, and therefore either return a false positive or a false negative. This is reflected in our error rate as seen below: our error rate for Clean data is significantly lower to that for Noisy data.

Confusion Matrix:(Clean data)

8	1	1	0	2	0
1	17	0	0	1	0
3	2	9	1	0	2
1	3	0	13	1	0
0	3	1	0	5	0
1	0	2	0	0	23

Confusion Matrix: (Noisy data)

3	0	1	1	0	0
1	18	2	2	0	2
4	1	10	2	0	2
1	0	1	17	0	0
1	2	4	2	2	2
2	0	2	1	1	14

When looking at these two confusion matrices, we can see that the diagonal elements are highest (predictions corresponding to reality): the probability of getting the right predicted emotion is higher than getting any other emotion for all of our trees, given an example, and vice-versa.

Clean Dataset

	Anger	Disgust	Fear	Happiness	Sadness	Surprise
Average Recall Rate	66.66	89.4737	52.9412	72.2222	55.5556	88.4615
Average precision rate	57.1429	65.3846	69.2308	92.8571	55.5556	92
average F1 measure	61.5385	75.5556	60	81.25	55.5556	90.1961

Error Rate: 0.2574

The above table shows the average classification results for each class over our 10 folds. Looking at Disgust, we can see that the Average Recall Rate(ARR) is much higher than the Average Precision Rate(APR). This means that we are very likely to recognise that someone is expressing Disgust, but we may not know when they are not: we have about 45% chances of returning a presence when the emotion is not actually that of Disgust. We see the very opposite phenomenon with Happiness, where ARR is much lower than APR. For the other emotions, ARR and APR are generally similar.
We can see that our F1 for Happiness and Surprise are particularly high: our trees are quite reliable when it comes to recognising these two emotions.

Noisy Dataset

	Anger	Disgust	Fear	Happiness	Sadness	Surprise
Average Recall Rate	60	72	52.6316	89.4737	15.3846	70
Average precision rate	25	85.7143	50	68	66.6667	70
average F1 measure	35.2941	78.2609	51.2821	77.2727	25.0000	70

Error Rate: 0.3663

In the table above, we can see our statistical analysis for noisy data. To the exception of Sadness, the recall rates drop, but stay rather high: we still have a fair chance of recognising an emotion when it is shown to us. However, our APR change quite dramatically. For Anger, it drops by half, which is clearly due to noise obstructing the obtention of data (see the assumptions in *Ambiguity*). However, for Disgust, it experiences an increase of about 33%, which is considerable. This is most likely due to pure chance.

Ambiguity

In our problem, we want each example to be mapped to only one emotion amongst the 6 available. However, each tree is specific to one emotion and one only. We therefore need to combine our trained trees and our results in order to get a proper mapping function. Yet, even with the trees combined, we could get the problem of not being able to choose between two trees, and therefore, two emotions. Indeed, if more than one tree return a positive classification for an example, we need to decide which one is more suitable for an example. In order to solve this problem, we adopted two methods: a random method, and a height-based method.

a) Random

We put all of the classes that return a positive classification into a separate set, and choose randomly amongst them. Assuming we have n classes in total, amongst which k returned a positive classification. Then we choose one class amongst k, each holding probability 1/k of being chosen. The advantage of the random algorithm is that it is easy to implement and it is efficient in terms of computing time. Indeed, there is no need to get any additional data from the examples, as all we need is the final set of classes that returned a positive result. Furthermore, in the case where we cannot get any more information, we have a fair algorithm that provides a one-to-one result, which is what we want. However, this method has downsides. Indeed, the fact that we choose randomly means that we have low accuracy due to the introduction of possible false positives; we have only 1/k chances of getting the right result. Furthermore, the hitting rate is proportional to the number of classes returning a positive value. In the case all of our 6 classes return a positive result, we have about 16% of getting the right classification, which is very low.

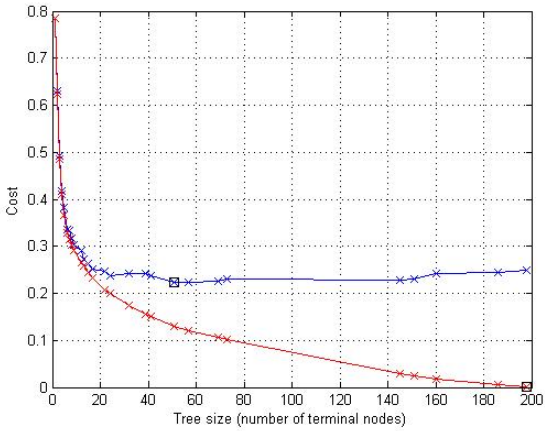
b) Shortest Height

Another algorithm we decided to implement uses the height of the tree branches that lead to a positive result. We take each of these branches and calculate the height from the root to that particular positive leaf, and then choose the tree that returns the smallest length. This can be used to decide on the predicted emotion, because a shorter tree means a smaller set of deciding attributes, and therefore induces that the said attributes are very specific to a given emotion - a short hypothesis gives a more probable result.

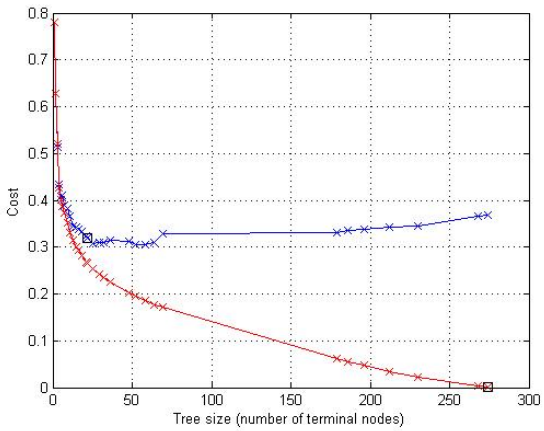
Furthermore, if we have a small set of attributes, then we can reduce the problem of overfitting: we will have just enough characteristics to decide on a result, but not so many that we get false negatives. However, we have a problem when multiple trees return the same branch height. In that case, we basically return to the random algorithm, as we cannot decide on a tree using only this much information.

Pruning

Clean Data



Noisy Data

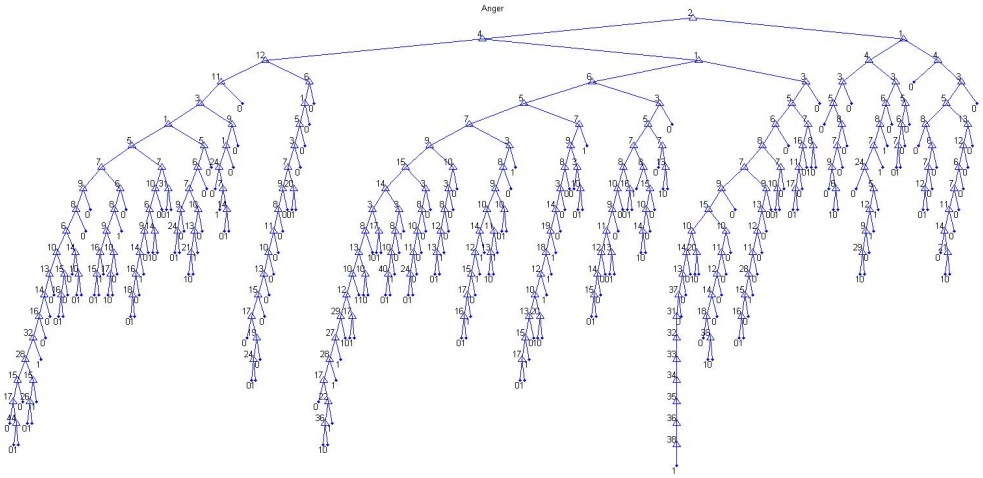


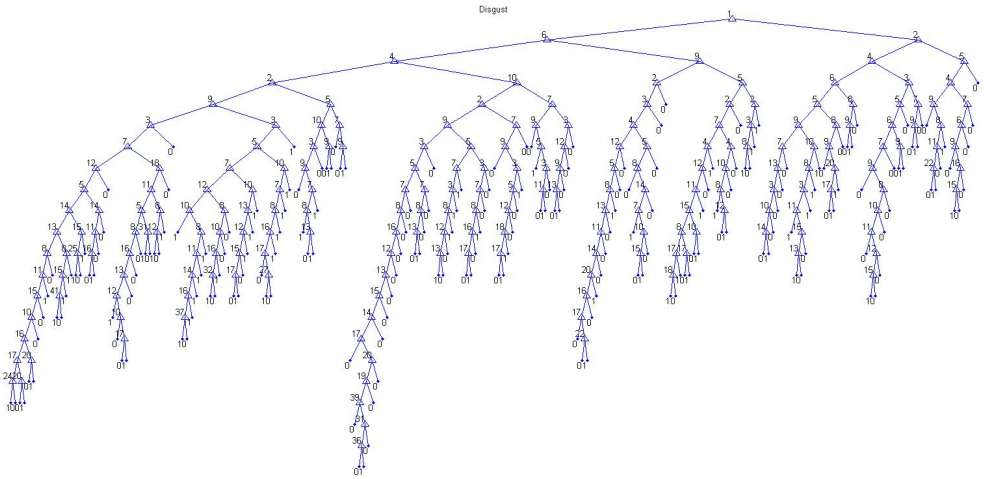
pruning_example creates a tree using the two different data sets we have: the clean data set and the noisy data set. It then measures their respective cost for all of our classifiers and plots it under two curves, each corresponding to a different type of learning method: cross-validation, in blue, and resubstitution, in red.

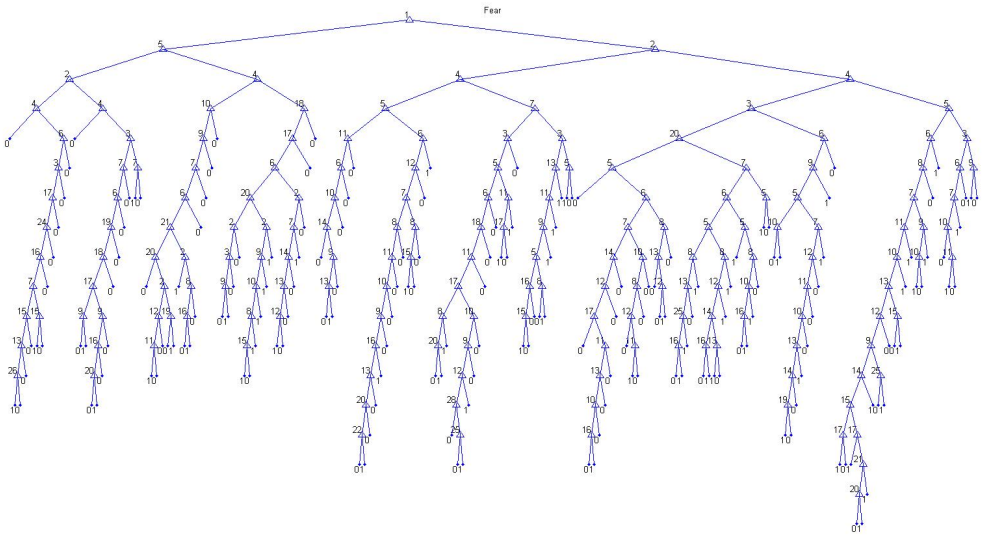
When we look at the clean data set, we can see that the resubstitution method's cost decreases until it reaches its minimum 0 at 200 leaf nodes. In comparison, when we look at the noisy data, the optimal tree size is reached around 275 leaf nodes, which is about 37% more than for clean data. This means that noise highly hinders the recognition process in the case of resubstitution: if we have more leaves, then we must have a larger and therefore longer tree (since we can only have 0 or 1 as an outcome to each attribute), hence more attributes to test. If we have more attributes to test, then these attributes must be not very specific to a particular emotion, so our hypotheses are not very good since they cannot differentiate one emotion from another easily. It also means we may have a problem of overfitting: a too long hypothesis would mean many attributes must be satisfied in order to reach a conclusion, but the more attributes there are, the more the noisy testing data is likely to fail one (or more of these tests).

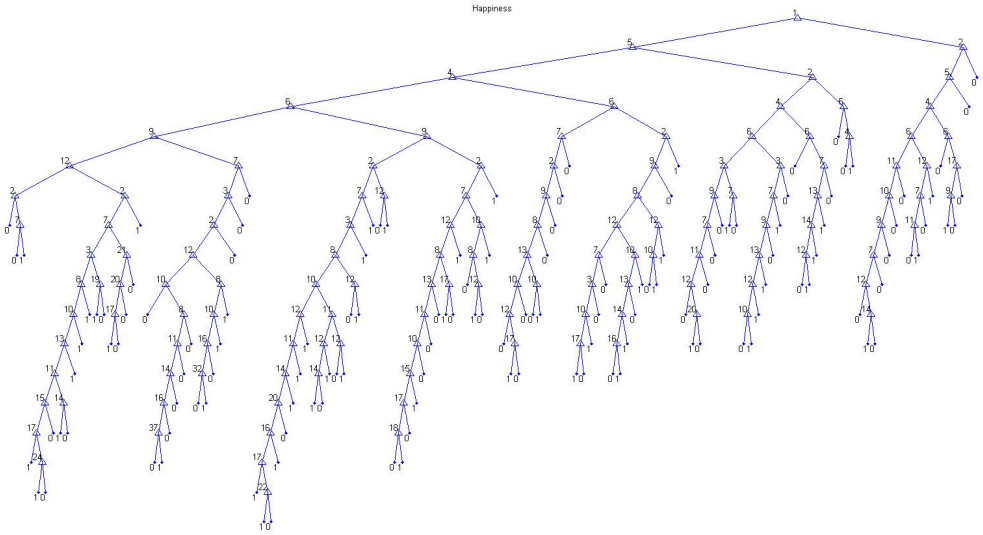
In the case of cross-validation, we can see that the cost never reaches 0. However, with clean data, it stabilises at a minimum point situated around 0.22 when approaching 50 leaf nodes. With noisy data, it reaches a minimum of about 0.3 around 20 leaf nodes, but slightly increases after passing 60 leaf nodes. We therefore have a difference of about 20%, which is much less than for resubstitution, but still quite high. But the advantage lies in the fact that the optimal tree size is much lower than for resubstitution; about ¼ of the resubstitution optimal tree size! Ideally, our trees should not be too big, and we therefore would have a good tradeoff between cost and robustness of our recognition system, as clean and noisy data costs are similar for cross-validation.

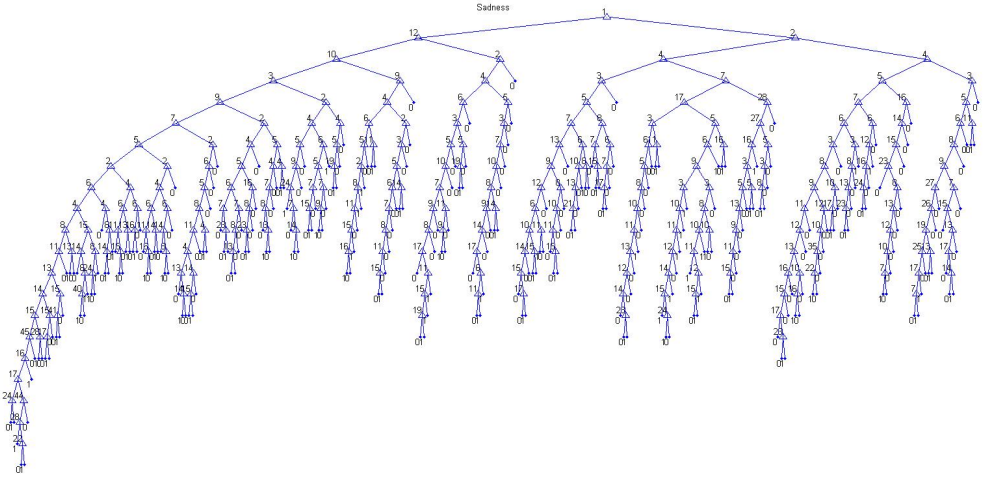
We can deduce that cross-validation is actually better than resubstitution in terms of performance for small trees, as the difference in cost between noisy data and clean data is quite low.

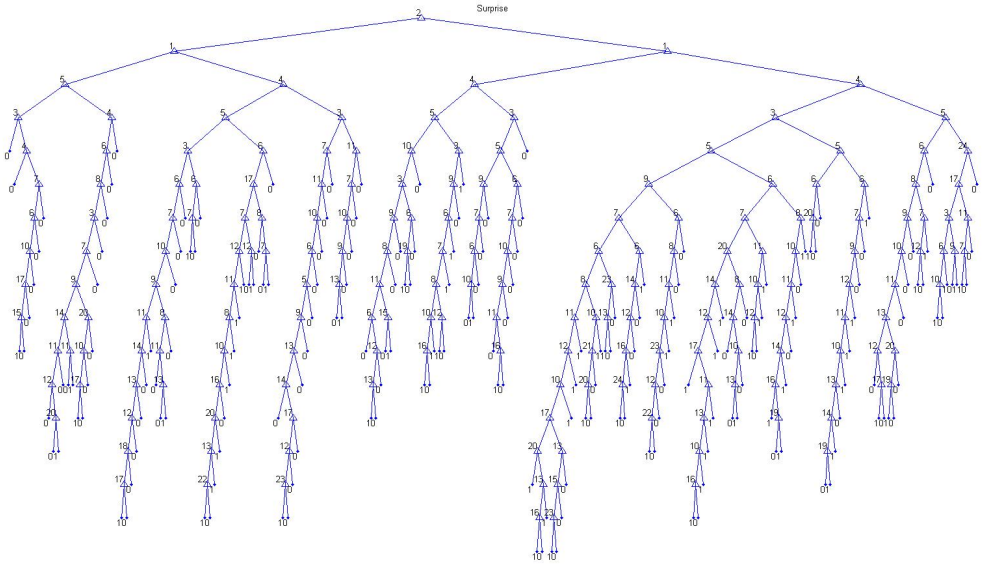

Anger


Disgust


Fear


Happiness


Sadness


Surprise