Coursework 1

Imperial College London

Department of Computing

Machine Learning (Course 395)

Authors:
Mario Zusag (CID: 00834438)
Anna Mitenkova (CID: 00927771)
Jack Collins (CID: 00829530)
Aditya Chaturvedi (CID: 00751251)

Date: February 13, 2018

1 Brief Summary of implementation details

To implement the coursework, we have created three python files using Python3.

1.1 File decision_tree.py

This file contains the definition of all required functions and contains

- a section concerned with the decision tree itself, where we have defined
  - the datastructure of the decision tree (class **Tree**)
  - function **subset** for creating binary vectors for a specified emotion
  - function **entropy**, which computes the entropy based on the number of positive and negative samples
  - function **majority_value**, which sums over all binary targets and divides over the number of samples and rounds the result (everything below 0.5 represents less votes than everything above 0.5)
  - function **choose_best_decision_attribute** for evaluating the best attribute to split the data. It computes the information gain (as described in the lecture) for every attribute and returns the attribute that has the largest information gain
  - function **decision_tree_learning**, which builds a decision tree given a binary target value as described in the manual. The function takes samples, an array of attributes (just a range between 1 and 45) and the binary target vector and builds a (binary) decision tree recursively by evaluating the attribute with the largest information gain and splitting the data according to this attribute
- a section concerned with the prediction, where we have defined
  - a **prediction** function, which takes a trained decision tree and test data and traverses down the decision tree, depending on the action units that are saved in the test data until it reaches a leaf node and returns either 0 or 1
  - function **test_accuracy**, which calculates which percentage of the predicted labels and the actual labels are the same
  - function **testTrees**, which takes a list of 6 pre-trained trees and returns the first emotion one of the 6 trees predicts.
  - function **testTrees2**, which also takes a list of 6 pre-trained trees and in case more than one emotion is predicted for the same datapoint, the emotion is randomly chosen.
- a section concerned with accuracy measures, where we have defined

  - function **confusion_matrix**, which returns a 6×6 matrix, where each row represents the actual class and each column represents the predicted emotion. For each datapoint the entry (i, j) is incremented, where i is the actual emotion and j is the predicted emotion.
  - function **recall**, which returns the ratio of $\frac{TP}{TP+FN}$ for each class based on the confusion matrix.
  - function **precision**, which returns the ratio of $\frac{TP}{TP+FP}$ for each class based on the confusion matrix.
  - function **f1_measure**, which computes $F_1 = 2\frac{precision*recall}{precision+recall}$ for each class
  - function **classification_rate**, which sums over the diagonal of the confusion matrix (the correctly classified emotions) and divides it by the sum of all elements of the confusion matrix.
  - function **k_fold_cross_validation**, which prints the confusion matrix, the precision, recall and F1-measure rate to screen. It trains 6 decision trees on k-1 parts of the data and predicts the emotions with the pre-defined function **testTrees** with the k-th part of the data. The predicted emotions for the whole dataset are passed on to functions, which build up the different accuracy measures. Each of the measures is incremented k times by the accuracy of the current predicted emotions and is averaged before printed to screen by a simple division over k.

- a section concerned with visualization, which is described in detail at 2
- a section concerned with saving and loading the trees as pickle (.p) files

1.2 File main.py

This file loads the datasets, calls the functions defined in decision_tree.py and prints the results to screen. It contains

- a section concerned with initialisation, which loads the clean and noisy datasets into arrays.
- a section concerned with defining the binary emotion targets for the clean and noisy dataset
- a section concerned with the training of 6 trees by calling the function **decision_tree_learning** on the binary targets, the clean and noisy dataset and a simple range between 1 and 45, which represents the attributes (45 action untis)
- a section concerned with testing the perfect decision trees and printing the results to screen
- a section concerned with testing all emotions for the perfect decision trees
- a section for displaying the results from **k_fold_cross_validation** for $k = 10$

- a section saving the visualisation of the 6 trees that were trained on the clean dataset as .png files
- a section saving .p files for all perfectly trained trees
- a section showing how to load a trained tree that has been saved as a pickle file

1.3 File test.py

This file is a layout for how to evaluate our coursework. This code:

- loads the data into numpy arrays
- imports all decision trees from pickle files
- gets predictions using the testTrees3 function
- prints the evaluation metrics

2 Decision Tree Diagrams

To visualize the decision trees we used the graphviz *(https://www.graphviz.org/)* and pydot *(https://github.com/erocarrera/pydot/blob/master/pydot.py)* packages. These can be installed on linux using:

```
sudo apt install python-pydot python-pydot-ng graphviz
```

or other operating systems using:

```
pip3 install graphviz
pip3 install pydot
```

In the visualization we included:

- Path to each node in terms of left (L) and right (R) movements from the root node (0).
- The number of the attributes on which the node split (for non-leaf nodes).
- The prediction assigned by the node (for leaf nodes).

Leaf nodes are colored red if assigned a prediction of 0 (emotion not present) or green if assigned a prediction of 1 (emotion present). The diagrams for each of the six trees trained on the entire clean data set can be seen in the following figures 1 to 6.

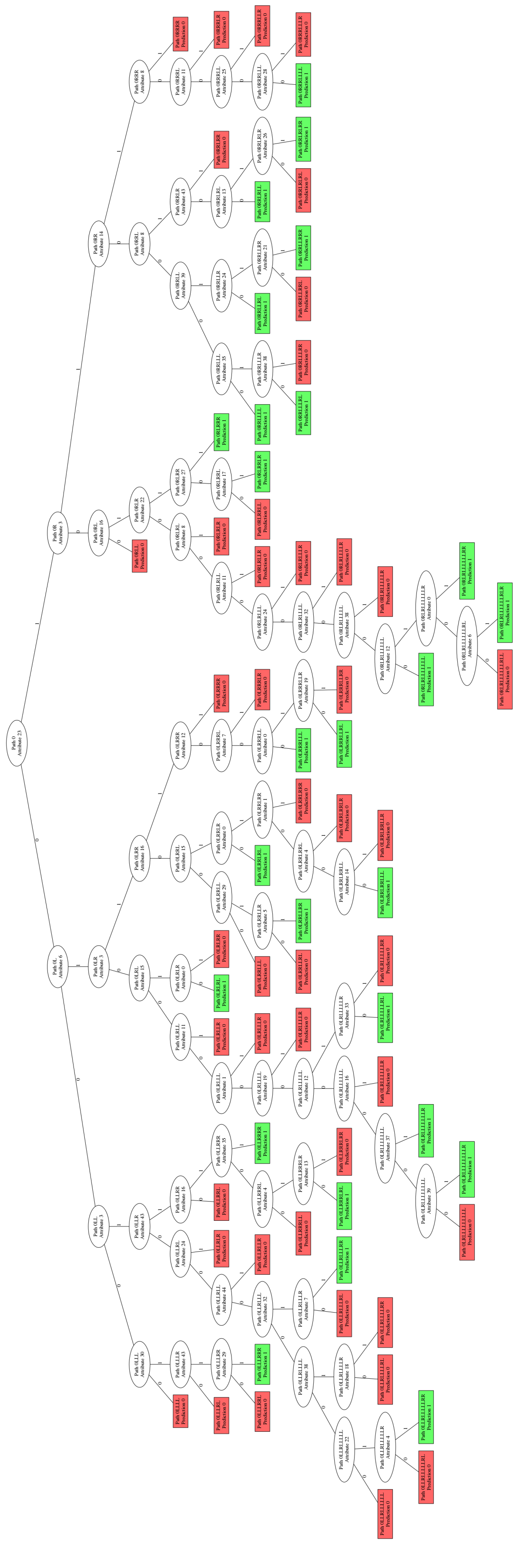

Figure 1: Anger decision tree

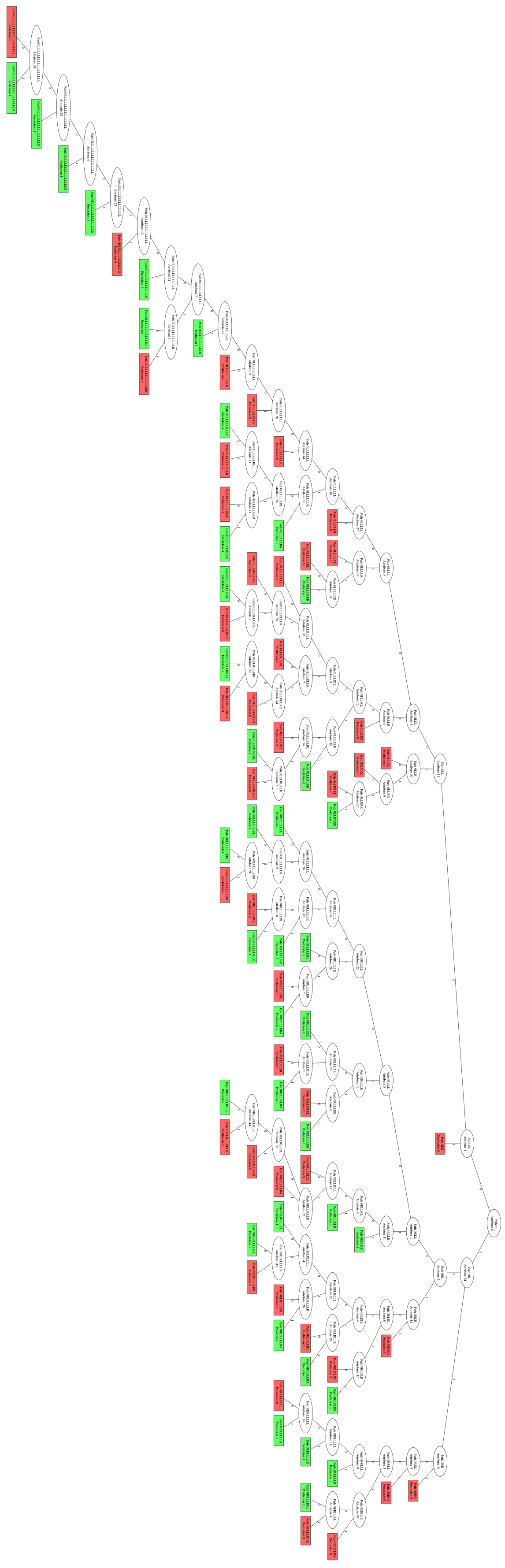

Figure 2: Disgust decision tree

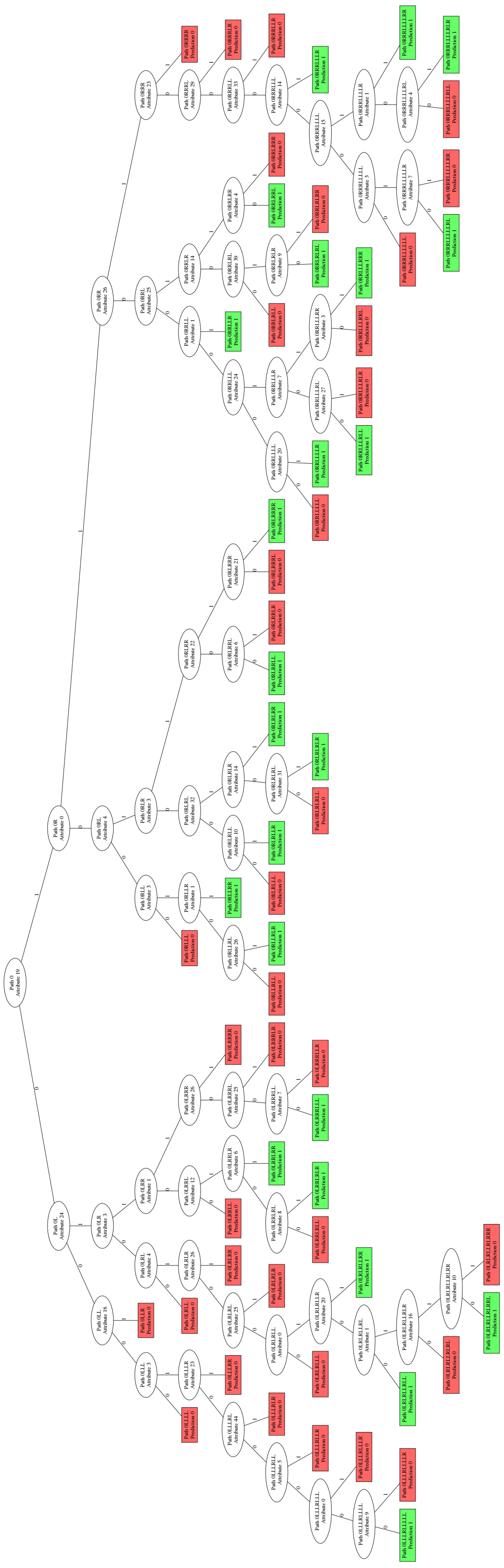

Figure 3: Fear decision tree

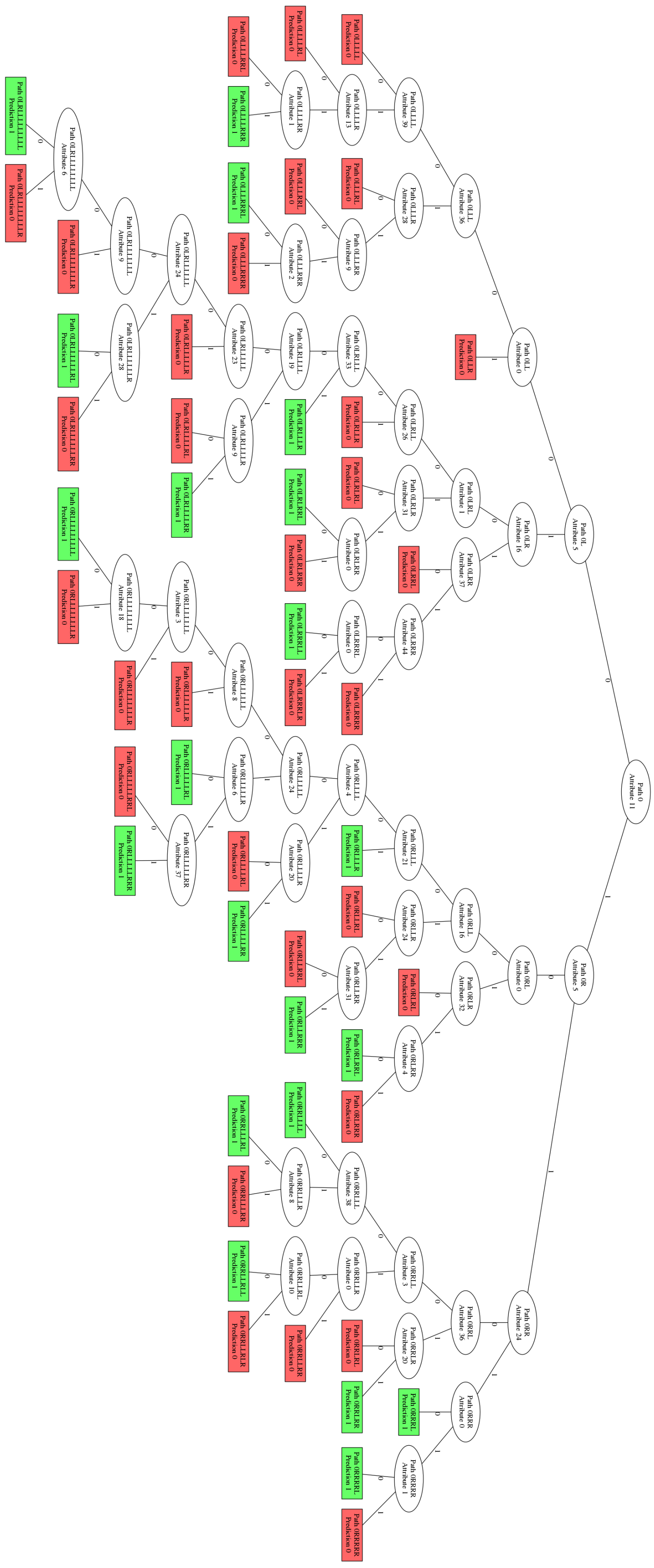

Figure 4: Happiness decision tree

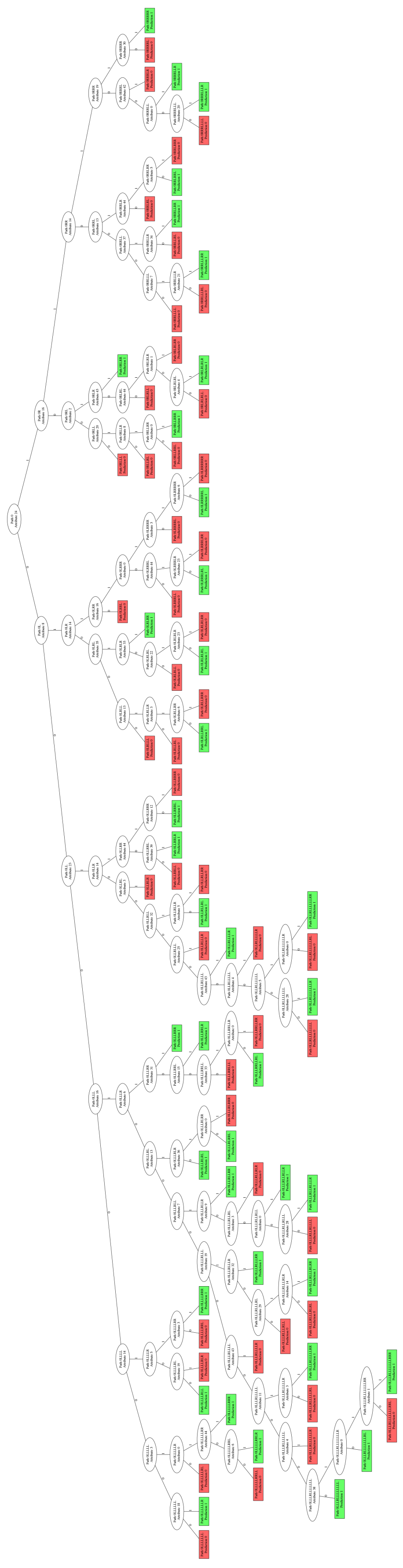

Figure 5: Sadness decision tree

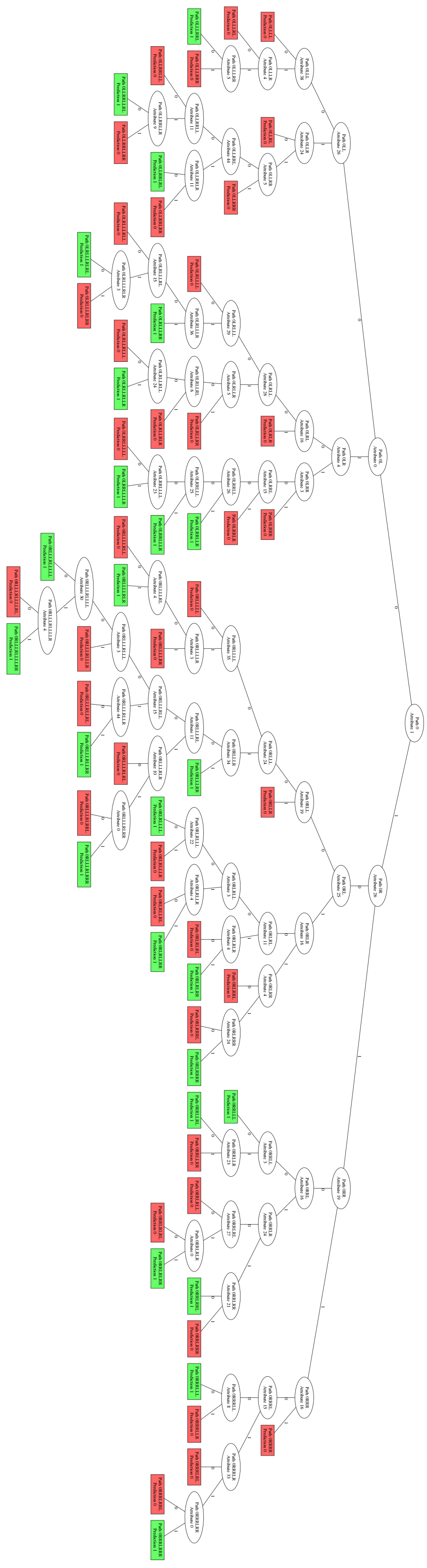

Figure 6: Surprise decision tree

3 Results of the evaluation

Comment on the results of both datasets, e.g. which emotions are recognised with high/low accuracy, which emotions are confused, i.e. analysis of cross-validation experiments.

Please find below the tables that contain information about the average confusion matrix, average precision, average recall(per class), average F_1-measure and the classification rate. The reason for using average for all the measures is due to the fact that we are evaluating our decision trees using the cross validation method.

Looking at the tables below we can see that the emotions that are recognised with the highest accuracy for the clean data are happiness and surprise. Next, in the decreasing order of accuracy for the clean data, the emotions are as follows: disgust, anger and fear, sometimes being confused with sadness, disgust and surprise respectively. The emotion that is recognised with the lowest accuracy for the clean data is sadness, often being confused with anger or disgust.

Looking at the table below we can see that the emotions that are recognised with the highest accuracy for the noisy data are happiness and surprise. Next, in the decreasing order of accuracy for the noisy data, the emotions are as follows: disgust, fear and sadness, sometimes being confused with fear, hapiness and anger respectively. The emotion that is recognised with the lowest accuracy for the noisy data is anger, often being confused with fear.

For a more detailed explanation on the results of the evaluation, see **Question 4: clean-noisy question.**

3.1 Clean data set

3.1.1 Average Confusion Matrix

The table below provides the average confusion matrix - one of the classification measures, that can allow to easily identify the confusion between different classes, i.e. if for example one class is commonly mislabelled as the other. Each entry represents the average number of times of how the label was predicted given the actual label value.

		Actual					
		anger	disgust	fear	happiness	sadness	surprise
Predicted	anger	9.5	1.3	0.5	0.4	1.1	0.3
	disgust	0.9	16.8	0.0	0.8	1.0	0.3
	fear	0.5	0.9	8.1	0.3	0.4	1.6
	happiness	0.1	0.9	0.0	19.5	0.6	0.4
	sadness	1.6	1.3	0.6	0.8	8.	0.9
	surprise	0.1	1.0	1.4	0.8	0.0	17.3

3.1.2 Average Precision per class

The table below provides the average precision per class - proportion of correctly classified positive examples out of the total number of predicted positive examples. It was computed using the formula: $\frac{TP}{TP+FP}$.

Emotion	anger	disgust	fear	happiness	sadness	surprise
Average Precision Rate (%)	77.25	76.39	76.62	86.32	73.77	83.62

3.1.3 Average Recall per class

The table below provides the average recall per class - proportion of positives that are correctly identified. It was computed using the formula: $\frac{TP}{TP+FN}$.

Emotion	anger	disgust	fear	happiness	sadness	surprise
Average Recall Rate (%)	72.67	84.36	68.39	91.40	59.17	84.40

3.1.4 Average F_1-measure for each of the 6 classes

The table below provides the average F1-measure for each of the 6 classes. It is convenient to use as it combined both recall and precision together. It was computed using the formula: $2 * \frac{precision*recall}{precision+recall}$.

Emotion	anger	disgust	fear	happiness	sadness	surprise
Average F_1-measure (%)	73.12	79.75	71.78	88.32	63.78	83.75

3.1.5 Classification Rate

This section provides information about the average classification rate - number of correctly classified examples divided by the total number of examples. It was computed using the formula: $\frac{TP+TN}{TP+TN+FP+FN}$.

Average Classification Rate: 80%

3.2 Noisy data set

3.2.1 Average Confusion Matrix

The table below provides the average confusion matrix - one of the classification measures, that can allow to easily identify the confusion between different classes, i.e. if for example one class is commonly mislabelled as the other. Each entry represents the average number of times of how the label was predicted given the actual label value.

		Actual					
		anger	disgust	fear	happiness	sadness	surprise
Predicted	anger	3.0	1.0	1.9	0.9	1.3	0.7
	disgust	0.8	14.5	1.4	0.9	0.9	0.2
	fear	1.1	0.8	13.1	1.7	1.1	0.9
	happiness	0.8	0.8	1.1	17.1	0.2	0.8
	sadness	1.5	0.8	1.0	0.5	6.2	1.0
	surprise	0.9	0.6	1.5	0.5	0.6	17.9

3.2.2 Average Precision per class

The table below provides the average precision per class - proportion of correctly classified positive examples out of the total number of predicted positive examples. It was computed using the formula: $\frac{TP}{TP+FP}$.

Emotion	anger	disgust	fear	happiness	sadness	surprise
Average Precision Rate (%)	39.81	78.14	65.59	78.59	61.69	83.23

3.2.3 Average Recall per class

The table below provides the average recall per class - proportion of positives that are correctly identified. It was computed using the formula: $\frac{TP}{TP+FN}$.

Emotion	anger	disgust	fear	happiness	sadness	surprise
Average Recall Rate (%)	33.14	76.56	70.20	82.62	56.75	81.36

3.2.4 Average F_1-measure for each of the 6 classes

The table below provides the average F1-measure for each of the 6 classes. It is convenient to use as it combined both recall and precision together. It was computed using the formula: $2 * \frac{precision*recall}{precision+recall}$.

Emotion	anger	disgust	fear	happiness	sadness	surprise
Average F_1-measure (%)	33.75	77.21	67.47	80.15	57.71	82.03

3.2.5 Classification Rate

This section provides information about the average classification rate - number of correctly classified examples divided by the total number of examples. It was computed using the formula: $\frac{TP+TN}{TP+TN+FP+FN}$.

Average Classification Rate: 72%

4 Questions

4.1 Noisy-Clean Datasets Question

Is there any difference in the performance when using the clean and noisy datasets? Yes, as can be seen from the performance tables in section 3, there is a significant difference between the clean and noisy datasets. The overall classification rate between the two sets drops from 80% to 72%.

If yes/no explain why. The difference in the decreasing rate of the overall classification rate between the two datasets occurs due to the nature of the data. When the data is noisy, some examples might have same attributes but different classifications, some attributes might be irrelevant (due to the way the data was acquired), or might be labeled incorrectly. All these result in high probability of misclassification, and hence lower classification rate. Additionally, emotions such as happiness are distinctly recognised, while emotions such as anger, fear and sadness (being more identical in nature) trigger similar action units, leading to more confusion during classification.

Discuss the differences in the overall performance and per emotion. As mentioned above, the difference in the overall performance of clean and noisy data sets is represented by a change in the overall classification rate: 80% and 72% for clean and noisy data sets respectively.

From the confusion matrix we can see that the highest True Positive value corresponds to emotions happiness (19.5 and 17.1 respectively for the clean and noisy data set) and surprise (17.3 and 17.9 respectively for the clean and noisy data set). Using also the average recall per class information and average precision per class information we can deduce that these are the emotions that are recognised with the highest accuracy for **both** the clean and noisy data:

- Average recall rate of 91.4% and 82.62%, and an average precision rate of 86.32% and 78.59 % for happiness for clean and noisy data respectively.

- Average recall rate of 84.4% and 81.36%, and an average precision rate of 83.62% and 83.23 % for surprise for clean and noisy data respectively.

This hypothesis is supported by the F_1-measure, which conveys the balance between the precision and the recall. It is 88.32% and 80.15% for happiness for clean and noisy data respectively, and 83.75% and 82.03% for surprise for clean and noisy data respectively.

From the confusion matrix we can see that the lowest True Positive value for the clean data set corresponds to emotion sadness with the value of 8.0, **whereas** the lowest True Positive value for the noisy data set corresponds to emotion anger with the value of 3.0. Using also the average recall per class information and average precision per class information we can deduce that sadness and anger are the emotions that are recognised with the lowest accuracy for the clean and noisy data respectively.

- Average recall rate of 59.17%, and an average precision rate of 73.77% for sadness for the clean data
- Average recall rate of 33.14%, and an average precision rate of 39.81% for anger for noisy data.

This hypothesis is supported by the F_1-measure, is 63.78% for sadness for clean data, and 33.75% for anger for noisy data. In the clean data set the emotion sadness is often being confused with anger or disgust, **whereas** for the noisy data set the emotion anger is often being confused with fear.

Note hear that while the drop in performance is not big for emotions happiness, sadness, disgust and surprise between the clean and noisy data sets, the drop in performance was significant for emotions anger (TP dropped from 9.5 to 3.0) and fear (TP increased from 8.1 to 13.1) between the clean and noisy data sets.

From confusion matrix we can see that emotion anger was mostly confused with the emotion disgust for clean data, while it was mostly confused with the emotion fear for noisy data. When misclassified, the emotion disgust was mostly confused with the emotion sadness for clean data, while it was mostly confused with the emotion fear for noisy data. The emotion fear was mostly confused with the emotion surprise for clean data, while it was mostly confused with the emotion happiness for noisy data. When misclassified, the emotion happiness was mostly confused with the emotion disgust for clean data, while it was mostly confused with the emotion fear for noisy data. The emotion sadness was mostly confused with the emotion anger for both clean and noisy data. When misclassified, the emotion surprise was mostly confused with the emotion fear for both clean and noisy data.

4.2 Ambiguity Question

Each example needs to get only a single emotion assigned to it, between 1 and 6. Explain how you made sure this is always the case in your decision tree algorithm. Describe the different approaches you followed (at least two) to solve this problem and the advantages/disadvantages of each approach.

The *predict_point, predict_point2 and predict_point3* functions take as input a list of decision trees and a single data point, and return the predicted emotion for that data point. To make sure that exactly one emotion is predicted for a data point we needed to handle **three** cases: no trees detect the corresponding emotion, one tree detects the corresponding emotion, and more than one tree detects the corresponding emotion. The second of these cases is the most simple, requiring just selecting the single tree, which predicted its emotion. In the case that no tree detected the corresponding emotion, an emotion was chosen at random. If more than one tree was activated, one of the following approaches was used:

Approach 1: Choose first emotion found.
For this approach the function iterates through the list of trees until it finds a tree which predicts that the corresponding emotion is present. This emotion is immediately returned as the prediction for the data point provided. The advantage of this approach is that it is very efficient as it does not create every tree for every data point. It instead moves to the next data point as soon as it has any emotion predicted for the current one. However, this strategy is heavily biased towards predicting emotions for which the trees are early in the list of trees. To combat this the trees list could be arranged in an order which places commonly 'confused' emotions in such a way as to maximize the likelihood of making a correct prediction when these are confused.

Approach 2: Choose randomly from predicted emotions.
This approach calculates a prediction from each tree for the data point passed in. The emotions which are detected are stored in a list. When the predictions of all trees have been evaluated, an emotion is picked at random from this list. This is returned as the predicted emotion for the point. The advantage of this approach is that all trees are taken into account in predicting the emotion present. The random choice deals well with commonly 'confused' emotions. A disadvantage of this method is that the random choice of predicted emotions does not account for how often each emotion occurs in the training data.

Approach 3: Choose most probable from all emotions.
In this approach the majority value of a leaf node isn't used at all. Instead, a probability is assigned to each node as the proportion of samples at that node which contain the emotion. When predicting the emotion of a data point, the nodes of the tree are iterated through beginning at the root and taking the left or right path at every node depending on the features of the data point until a root node is reached, as usual. During this process the probability value of each node visited is averaged with a

'running average' ie. running_average ← (running_average + current_node.prob)/2. When a leaf node is reached the final value of running_average is returned. For a given data point, this process is done for each tree. The tree which gives the highest value for running_average is taken to be the predicted emotion.
This approach has the advantage that it, in effect, 'prunes' the tree by taking the predictions of non-leaf nodes into account as well as predictions of the leaf nodes. It also has the nice property that nodes further from the leaves contribute less and less to the prediction assigned to the data point. However, this approach could definitely be improved through experimenting with the way that the running average is calculated by giving the current node and previous node contributions different weighting.

Each of these approaches performs much worse on the noisy data than the clean data as can be seen in Figure 7.

	Clean Data	Noisy Data
Approach 1	0.72	0.59
Approach 2	0.73	0.61
Approach 3	0.80	0.72

Figure 7: Classification rate for different ambiguity approaches calculated using 10-fold cross validation.

Approach 3 performed much better than the previous two approaches according to every evaluation measure used. This approach was better able to distinguish between commonly 'confused' emotions and had better generalization to the noisy data set.

4.3 Pruning Question

In this question we have run the pruning_example function, which was provided in the task, using both the clean and noisy datasets. The output figures of the function can be seen below:

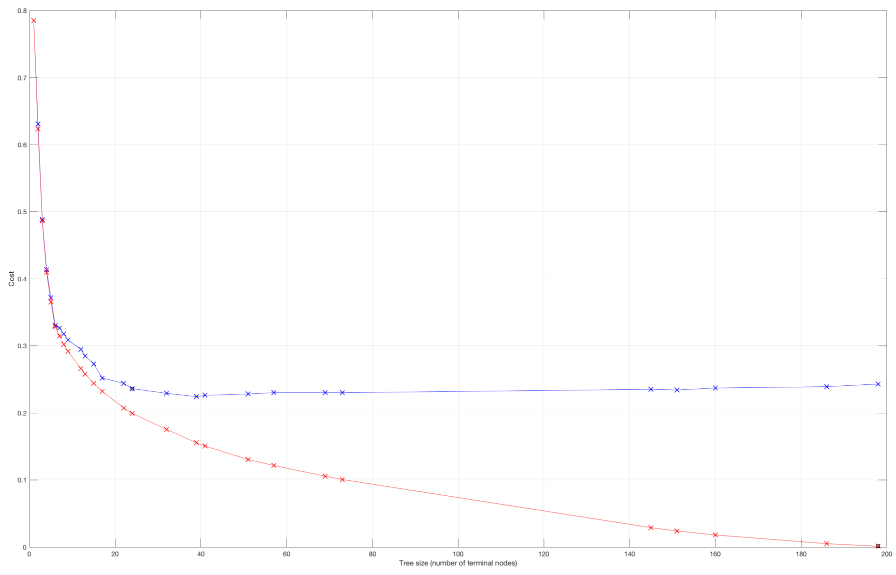

Figure 8: Output of the pruning_example function on the clean data

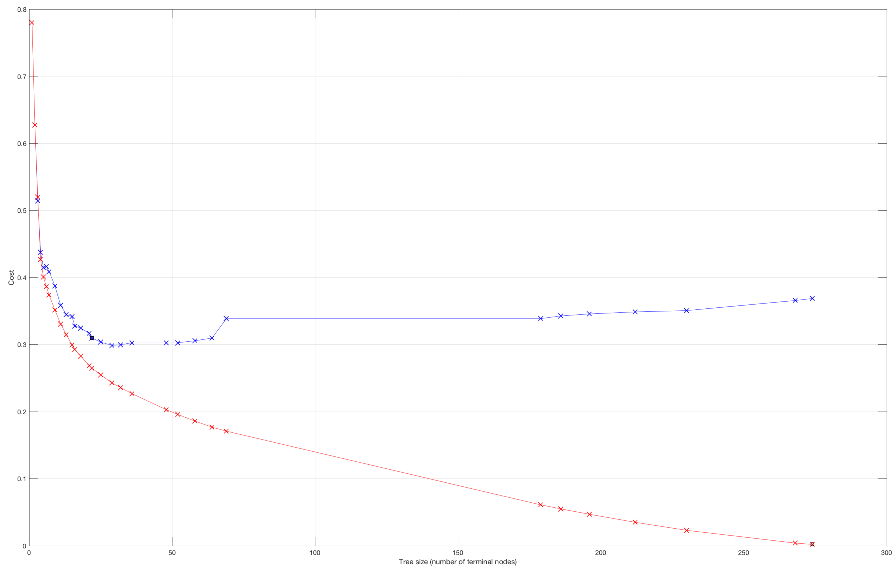

Figure 9: Output of the pruning_example function on the noisy data

Before proceeding with this question, it is worth explaining the concept of pruning firstly. Pruning is the procedure that allows to increase the accuracy of the decision tree by decreasing its complexity (Occam's razor). This is done by removing the nodes of the tree that have the least effect when performing the classification.

How does the *pruning_example* function work? The *pruning_example* function takes in data samples y and x, where y are the response values, and x is a matrix of predictors, and outputs a figure representing the pruning process. It firstly uses an inbuilt MATLAB function *classregtree*, which creates a binary decision tree based on our x and y data. It then uses inbuilt MATLAB functions *test* to compute the costs of the tree using resubstitution and cross-validation methods, as well as to obtain the best level of pruning. It finally uses an inbuilt MATLAB function *prune*, which returns the decision tree pruned to the best level of pruning.

One figure with two different curves should be generated for each dataset: clean and noisy. Include the two figures in your report and explain what these curves are. The two figures, each with two different curves, were generated using the *pruning_example*, and are shown in the figures Fig. 8 and Fig. 9 above. Each of the curves represents the cost of the tree over the test data as the tree is being pruned. The blue curve corresponds to the cost of the tree over the different data sample than the one used to create the original tree, i.e. testing sample. The red curve corresponds to the cost over the same data sample that was used to create the original tree, i.e. training sample.

Explain why the curves have this shape. What is the difference between them? What is the optimal tree size in each case? It is easier to describe curves by looking at them from left to right. As it can be clearly seen from output figures, red curve is "exponentially-shaped" and downward slopping, reaching almost zero as the tree size increases in both Fig. 8 and Fig. 9, i.e. both for clean and noisy data respectively. The blue curves behave slightly differently in each of the plotted figures. For clean data, the curve "exponentially decreases" and then stagnates around roughly the same cost value as the tree sizes increase. For noisy data the curve decreases less smoothly, and after reaching the lowest cost level, it jumps up slightly, and only afterwards stagnates around roughly the same cost value as the tree size increase.

It is easier to interpret the curves by looking at them from right to left, because as we start, the decision tree has its largest size, i.e. the pruning has not started yet. As we proceed with pruning, the number of nodes that are being cut out is increasing, and hence the overall size of the tree is decreasing. Looking at the figure Fig. 9, we can see that as we proceed with the pruning procedure, the blue cost decreases (accuracy increases). The lowest (most optimal) cost level of 0.3097 is achieved when the tree has 64 terminal nodes. Looking at the other figure Fig. 8, a similar tendency, though much less obvious, can be observed: the blue cost is slowly decreasing, reaching its lowest of 0.2301 when the tree has 51 terminal nodes. The difference in the decreasing rate of the blue cost curves between the two figures occurs due to the fact that one corresponds to the cost of the tree over the clean training set (different to test set), and the other one corresponds to the cost of the tree over the noisy training set (different to test set), i.e. due to the nature of the data. When the data is noisy, some examples might have same attributes but different classifications, some attributes might be irrelevant (due to the way the data was acquired), or might be labeled incorrectly. All these result in high probability of misclassification, and hence high cost value. As we perform pruning and remove the those ”faulty” nodes, the accuracy improves significantly. While when the data is clean, we do not have this instances of ”faulty” nodes, and so the reduction is cost is achieved mainly due to the reduction of complexity (Occam's razor).After we pass the optimal tree size, in each case the cost starts increasing again. This occurs due to the fact that we start removing too many nodes, which leads to misclassification, i.e. underfitting occurs. As a result, when the number of nodes is close to 0, the blue cost curves increase dramatically.

The red cost curves behave similarly in both Fig. 8 and Fig. 9 despite the difference in the nature of the data. This occurs due to the fact that in this case the tree was tested on the same data as the one that we used for training. Hence, when the tree is at its largest, i.e. no pruning, we would have an almost perfect fit (we are fitting the same data as we trained on) (cost of 0.00099602 for the number of nodes 198 on clean data and cost of 0.0020 for the number of nodes 274 on noisy data). However, as we proceed with pruning, removing nodes would mean moving further away from our ”perfect” model for the decision tree, and hence leading to an increasing in the cost. The latter would be at its largest when the number of nodes is close to 0.