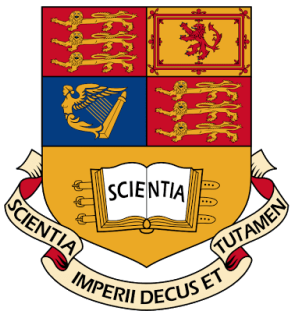


Imperial College London

Machine Learning Image Recognition

Group 35
CBC Helper: Evangelos Ververas

Ao **Shen**
ao.shen17@imperial.ac.uk
CID: 01394238 (as5017)
MSc Computing Science

Zhongyuan **Hau**
zhongyuan.hau17@imperial.ac.uk
CID: 01279806 (zh1316)
MSc Computing Science

Stanley **Loh**
qsl17@imperial.ac.uk
CID: 01441443 (qsl17)
MSc Computing (Specialism)

Shan Xian **Yong**
sam.yong17@imperial.ac.uk
CID: 00756540 (sxy17)
MSc Computing Science

Implementation Details

Program Usage

To run the program, you need the following installed on your system:

- Python 2.7
- Graphviz (especially dot)
- Graphviz Python Library (To install this, run "pip install graphviz --user")

The following command runs the main program using Python 2.7:

```
$ python main.py <datafile> <mode>
```

- **datafile** - filename of either clean or noisy data MAT file. Defaults to clean data set.
- **mode** - integer from 1 to 3: 1 - Random Class Assignment , 2 - Deepest Tree Assignment , 3- Shallowest Tree Assignment

```
$ python rdfmain.py <datafile> <mode>
```

- **datafile** - filename of either clean or noisy data MAT file. Defaults to clean data set.
- **mode** - integer from 1 to 5: 1 - Full training with AdaBoost, 2 - Full training without AdaBoost, 3 - 10-Folds CV with AdaBoost, 4 - 10-Folds CV with AdaBoost, 5 - Validation test on attribute set size

The graphical representations for all the trees trained and used by the program are generated as "Tree*.png" files in the same folder as the program. Likewise, the Pickle files that represent our trees are generated as "*.pkl" files in the same folder. If the pkl files are already available in the same folder, you may run the test tree functionality by invoking the following command with its arguments:

```
$ python test.py <datafile> <treefile>
```

The **test.py** program runs the testTrees function using the provided trees against a given data file. The program will also save the predictions in "predictions.txt".

- **datafile** - filename of either clean or noisy data MAT file. Defaults to clean data set.
- **treefile** - one of the "*.pkl" created by *main.py* or *rdfmain.py* in training mode (provided in Trees subfolder).

NOTE: The RDF tree files are too huge to be included in the submission (each about 30 MB) and hence we only include “clean_rdf.pkl” in the submission. If you wish to run the other trees, use the *rdfmain.py* program to build the other trees.

Tree Construction and Training

ID3 Algorithm

The Iterative Dichotomiser 3 (ID3) algorithm we employed recursively generates our tree by starting from a root node chosen by the best attribute with the highest information gain. This is measured by a reduction in entropy by the chosen attribute. We are "peaking" down the tree to see how well each attribute can classify the remaining labels. This results in 6 decision trees classifying 6 emotions from 45 attributes.

The recursive algorithm has two base cases:

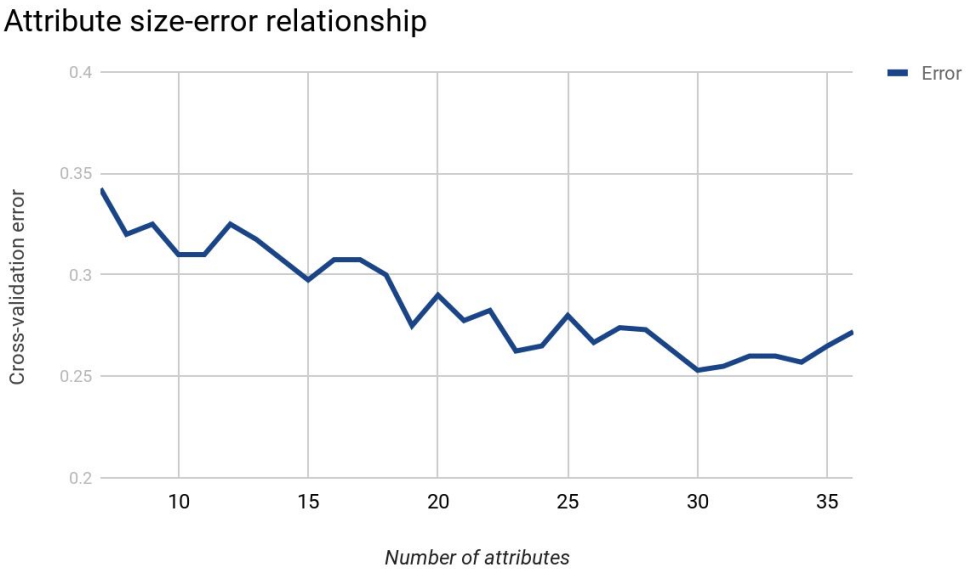
1. When all binary targets are the same, further classification will not yield any further information gain. We create a leaf node and set the label to the binary target.
2. When there are no more attributes to construct anymore: As each attribute is removed from the branch it was tested in, the remaining attributes to work on may run out. This is when we use the majority value of the remaining labels to determine the label of the leaf node.

Random Decision Forest (RDF) and Adaptive Boost (AdaBoost)

Instead of utilizing a single tree to perform classification for each individual class, we can employ multiple smaller trees (i.e. forest) to classify based on majority voting. In order to train different trees within the same class, we employ the method of bootstrap aggregation to randomly sample with replacement from the training set. Single trees trained with the entire training set are highly susceptible to noise fitting. Multiple trees trained through random sampling are more robust to noise as trees are decorrelated by being exposed to different training sets. In addition, the trees are trained with only a subset of the attributes which are chosen at random without replacement for each tree.

From various papers, it is recommended that the size of the attribute subset should be kept between $\sqrt{p}$ and $\frac{p}{3}$ where p is the total number of attributes. This means in our case where p = 45, our range should be from 7 to 15. However our cross-validation results show that larger amount of attributes is optimal instead (20 to 30). We explored using fixed attribute set sizes and random set sizes and the results are detailed below.

We have also augmented the RDF algorithm with the Adaptive Boost (AdaBoost) meta-algorithm. The premise of the meta-algorithm is that several 'weak learners' (individual trees in this case) have their results combined in a weighted sum to give the boosted output. Trees are weighted based on their performance margin on a validation set. Given a sample **x** and its label y' from the validation set, a tree outputs either y = 1 or 0. If y matches y', the weight of tree is increased by 1. If it does not match, the weight is reduced instead by a fixed value (number of classes). These weights are then used for the actual classification of the test set with negatively weighted trees being ignored.



Cross Validation

To maximize the utility of our small dataset, 10-fold cross validation is used on the clean data set to further validate our training and test results. The clean data is split into 10 segments, of which 9/10 will be chosen for training and validation (parameter optimization for Random Forest method), and 1/10 will be used for testing. The training/validation and testing sets will then be alternated throughout the entire data set without repetition, allowing for a slightly different training set, but completely different validation and testing set for us to verify the capabilities of our implementations. The cross-validation results of optimizing the size of the attribute subset in the Random Forest algorithm is described below. Results of validation experiments are discussed throughout this report.

Ambiguity (Combining Trees)

When combining the decisions made by all six trees, there can be zero, one or more classifications. Since there are no "mixed emotions", we outline four ambiguity handling strategies to disambiguate the result. There are two main cases the algorithm may encounter:

Case 1: More than 1 tree labelling the instance

1. **Deepest Tree**: The class that has the max depth is chosen. The deepest tree will fit the training examples more exactly (follow more specific paths). While this may generate more "accurate" results by testing against given data, it may overfit on unknown data.
2. **Shallowest Tree**: Since deep trees can overfit the training examples, using the tree with the minimum depth can better generalize the result, and yield better results on unseen data.
3. **Occurrence (frequency)**: Prior to training a tree in each fold, we count each class in the training data set. The class that has the highest count among all classes is chosen. This assumes that the label distribution of given data set is representative of the overall label distribution.

4. **Random Forest Decision Trees:** Virtually immune to this problem as classification is done by majority voting of forests (by taking the argmax of votes). In the highly unlikely case that the vote count is exactly equal for multiple trees, the class label with the lowest value is chosen.

Case 2: No classification

1. **Random**: Random class is selected among the all classes with uniform probability distribution.
2. **Deepest Tree**: The deepest tree is selected.
3. **Shallowest Tree**: The shallowest tree is selected.
4. **Random Forest Decision Trees:** Again, this algorithm is virtually immune to this problem. A classification will still be done despite all forests having very low votes for their respective classes.

In our tests, all examples were classified by at least one or more trees and hence we did not encounter any Case 2 ambiguity resolution. Our Visualizer recursively explores the tree and builds the data representation using Graphviz Dot syntax. The textual representation is then passed to Graphviz to generated either a PDF or PNG file. The resulting trees from the standard ID3 algorithm are presented (Annex A). The Random Forest algorithm generates too many trees (> 1000) for it to be feasibly included in this report.

Evaluation

Confusion matrix

Predictions were matched against test labels to construct a 6x6 confusion matrix. and the measures (recall, precision, F1 and Classification Rate) were calculated. With the 10 measurements from each fold respectively, the average of each measure across the 10 folds was computed and the results were discussed in the following section. In addition to reporting the above measures, we have also calculated the average Sample Error. The error rate is the percentage of incorrectly classified samples.

Clean Data Set

Max-Depth Ambiguity Handling

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	79	18	8	11	14	2
	2	13	144	3	11	15	12
	3	11	6	79	2	9	12
	4	6	13	10	172	11	4
	5	18	18	4	11	74	7
	6	8	8	14	5	8	164

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	59.3	73.1	65.6	80.8	55.3	79.4	68.9
Precision / %	58.2	69.7	66.7	80.5	55.3	81.8	68.7
F1 / %	58.1	71.2	65.8	80.2	54.4	80.3	68.3
Classification Rate / %	86.7	85.9	90.0	89.5	86.1	89.9	88.0

Correct Classifications = 70.9%
Sample Error Rate = 29.1%

Min-Depth Ambiguity Handling

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	88	11	7	8	12	6
	2	19	137	7	14	12	9
	3	6	7	86	2	4	14
	4	7	14	4	178	9	4
	5	24	11	9	14	66	8
	6	2	6	12	10	9	168

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	66.4	68.7	72.0	83.6	49.0	81.3	70.2
Precision / %	61.9	73.0	68.7	79.2	57.5	80.6	70.1
F1 / %	62.6	70.4	70.0	80.5	52.1	80.7	69.4
Classification Rate / %	87.6	86.8	90.9	89.4	86.5	90.0	88.5

Correct Classifications = 72.0%
Sample Error Rate = 28.0%

Most Occurrence Class Ambiguity Handling

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	78	23	4	9	14	4
	2	11	144	7	16	8	12
	3	10	7	71	3	12	16
	4	2	14	5	186	5	4
	5	15	16	6	14	72	9
	6	3	7	8	7	7	175

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	58.6	73.0	58.9	87.1	53.7	84.6	69.3
Precision / %	65.1	68.5	71.3	79.4	59.6	79.7	70.6
F1 / %	60.8	70.3	63.8	82.5	55.8	81.9	69.2
Classification Rate / %	88.4	85.7	90.3	90.3	87.3	90.5	88.8

Correct Classifications = 72.3%
Sample Error Rate = 27.7%

Random Forest - 300 Trees - 30 Attribute

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	97	16	3	4	10	2
	2	7	179	0	8	4	0
	3	4	3	91	2	1	18
	4	0	3	0	211	0	2
	5	6	23	3	4	91	5
	6	0	2	3	4	0	198

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	73.5	90.4	76.5	97.7	68.9	95.7	83.8
Precision / %	85.1	79.2	91.0	90.6	85.8	88.0	86.6
F1 / %	78.9	84.4	83.1	94.0	76.5	91.7	84.8
Classification Rate / %	94.3	92.9	95.9	97.0	93.9	96.0	95.0

Correct Classifications = 86.3%
Sample Error Rate = 13.7%

Random Forest - 300 Trees - 30 Attribute (AdaBoost)

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	99	15	6	4	5	3
	2	17	155	3	11	7	5
	3	4	2	97	3	0	13
	4	0	6	0	207	0	3
	5	11	10	5	9	84	13
	6	1	3	8	3	1	191

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	75.0	78.3	81.5	95.8	63.6	92.3	81.1
Precision / %	75.0	81.2	81.5	87.3	86.6	83.8	82.6
F1 / %	75.0	79.7	81.5	91.4	73.4	87.8	81.5
Classification Rate / %	92.7	91.3	95.0	95.5	93.2	94.0	93.6

Correct Classifications = 86.6%
Sample Error Rate = 13.4%

Random Forest - 1000 Trees - `rand(7, 32)` Attribute

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	97	16	3	4	10	2
	2	7	179	0	8	4	9
	3	4	3	91	2	1	18
	4	0	3	0	211	0	2
	5	6	23	3	4	91	5
	6	0	2	3	4	0	198

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	73.5	90.4	76.5	97.7	68.9	95.7	83.8
Precision / %	85.1	79.2	91.0	90.6	85.8	88.0	86.6
F1 / %	78.9	84.4	83.1	94.0	76.5	91.7	84.8
Classification Rate / %	94.3	92.9	95.9	97.0	93.9	96.0	95.0

Correct Classifications = 86.3%
Sample Error Rate = 13.7%

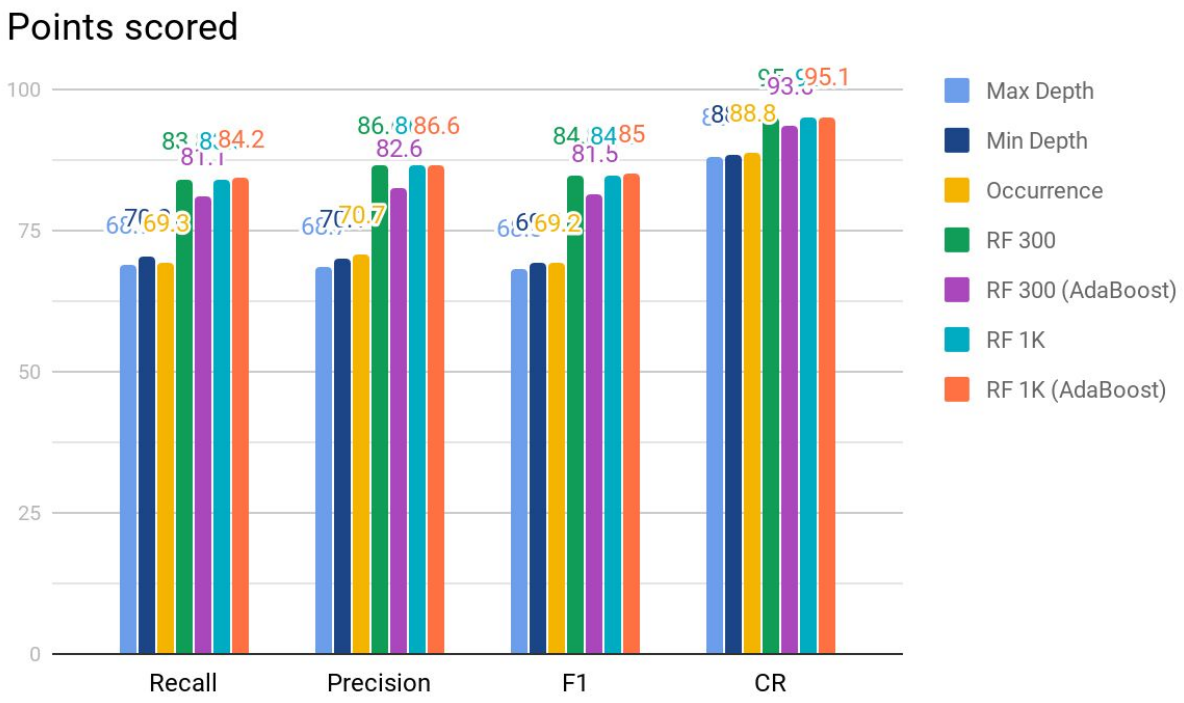
Random Forest - 1000 Trees - `rand(7, 32)` Attribute (AdaBoost)

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	100	15	3	4	8	2
	2	8	179	0	6	5	0
	3	3	3	91	2	1	19
	4	1	4	0	209	0	2
	5	6	18	4	6	93	5
	6	0	2	4	3	1	197

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	75.8	90.4	76.5	96.8	70.5	95.2	84.2
Precision / %	84.7	81.0	89.2	90.9	86.1	87.6	86.6
F1 / %	80.0	85.4	82.4	93.7	77.5	91.2	85.0
Classification Rate / %	94.6	93.4	95.7	96.9	94.1	95.8	95.1

Correct Classifications = 86.6%
Sample Error Rate = 13.4%

Clean Data Set Analysis



Across all methods, we note that the various measures (e.g. recall/precision/F1) for classes 1, 3 and 5 are noticeably worse than the other 3 classes. This is likely due to the fact that classes 1, 3 and 5 are slightly underrepresented compared to the other classes in the training set. The sample counts of the entire training set are: [132 198 119 216 132 207]. Having less positive examples to train on, these classifiers generally performed worse and hence resulted in poorer statistical measures. The converse could also be seen in the

data where classes 4, 6 (and to a lesser degree, 2) have noticeably higher statistical measures owing to a higher positive sample count for their respective classes.

The gap in the statistical measures of the underrepresented classes is bridged significantly when we used the Random Forest algorithm (some measures having low 60s are boosted to values as high as the 80s). The performance across all classes is noticeably greater than the ID3 algorithms (approximately 15% correct predictions).

Noisy Data Set

Max-Depth Ambiguity Handling

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	21	14	20	7	18	8
	2	18	131	14	14	8	2
	3	18	21	102	14	16	16
	4	13	14	17	143	5	17
	5	17	11	9	8	54	11
	6	14	9	28	12	16	141

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	23.6	69.3	53.4	68.7	48.9	64.0	54.7
Precision / %	23.3	65.7	52.7	71.9	46.3	71.7	55.3
F1 / %	22.6	67.2	52.4	69.6	46.7	67.2	54.3
Classification Rate / %	80.0	82.5	77.3	82.9	83.2	81.5	81.2

Correct Classifications = 59.1%
Sample Error Rate = 40.9%

Min-Depth Ambiguity Handling

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	21	7	19	13	18	10
	2	15	125	16	16	8	7
	3	13	21	100	21	13	19
	4	7	14	12	157	6	13
	5	21	12	7	8	51	11
	6	12	8	15	16	16	153

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	22.4	65.5	51.7	75.6	46.8	69.6	55.3
Precision / %	24.0	66.4	56.6	68.2	46.1	71.8	55.5
F1 / %	22.8	65.6	53.7	71.1	44.9	70.0	54.7
Classification Rate / %	81.7	83.0	79.5	82.7	83.5	82.6	82.2

Correct Classifications = 60.6%
Sample Error Rate = 39.4%

Most Occured Class Ambiguity Handling

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	21	9	19	13	15	11
	2	8	130	15	18	7	9
	3	15	20	99	23	7	23
	4	2	10	8	164	10	15
	5	16	9	9	13	51	12
	6	7	9	12	10	13	169

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	24.5	68.9	52.6	78.4	46.8	76.4	57.9
Precision / %	33.5	68.8	62.5	67.7	51.5	70.3	59.1
F1 / %	26.6	68.5	56.5	72.3	47.2	72.8	57.3
Classification Rate / %	84.6	84.7	81.0	83.9	85.1	84.0	83.9

Correct Classifications = 63.3%
Sample Error Rate = 36.7%

Random Forest - 300 Trees - 30 Attribute

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	25	12	23	10	12	6
	2	3	161	14	5	2	2
	3	4	13	144	11	5	10
	4	2	7	9	178	5	8
	5	11	11	13	3	62	10
	6	2	2	12	8	3	193

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	28.4	86.1	77.0	85.2	56.4	87.7	70.1
Precision / %	53.2	78.2	67.0	82.8	69.7	84.3	72.5
F1 / %	37.0	81.9	71.6	84.0	62.3	86.0	70.5
Classification Rate / %	90.0	91.5	87.0	91.8	91.1	92.4	90.6

Correct Classifications = 76.2%
Sample Error Rate = 23.8%

Random Forest - 300 Trees - 30 Attribute (AdaBoost)

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	26	11	22	11	13	5
	2	3	157	17	4	4	2
	3	5	13	141	10	8	10
	4	3	7	9	176	5	9
	5	2	11	11	3	65	10
	6	19	3	10	8	5	192

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	29.5	84.0	75.4	84.2	59.1	87.3	69.9
Precision / %	53.1	77.7	67.1	83.0	65.0	84.2	71.7
F1 / %	38.0	80.7	71.0	83.6	61.9	85.7	70.2
Classification Rate / %	89.9	91.0	86.8	91.6	90.4	92.2	90.3

Correct Classifications = 75.6%
Sample Error Rate = 24.4%

Random Forest - 1000 Trees - `rand(7, 32)` Attribute

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	14	15	25	12	13	9
	2	2	164	14	4	1	2
	3	3	14	125	14	5	26
	4	1	6	9	182	3	8
	5	7	10	13	5	56	19
	6	2	1	9	7	3	198

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	15.9	87.7	66.8	87.1	50.9	90.0	66.4
Precision / %	48.3	78.1	64.1	81.2	69.1	75.6	69.4
F1 / %	23.9	82.6	65.4	84.1	58.6	82.2	66.1
Classification Rate / %	89.3	91.5	84.8	91.5	90.3	89.6	89.5

Correct Classifications = 73.8%
Sample Error Rate = 26.2%

Random Forest - 1000 Trees - `rand(7, 32)` Attribute (AdaBoost)

		Prediction Class					
		1	**2**	**3**	**4**	**5**	**6**
True	**1**	19	16	21	9	16	7
	2	2	167	12	3	1	2
	3	4	18	125	11	6	23
	4	2	8	8	182	2	7
	5	10	11	11	6	57	15
	6	2	2	8	6	3	199

Class	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Recall / %	21.6	89.3	66.8	87.1	51.8	90.5	67.8
Precision / %	48.7	75.2	67.6	83.9	67.1	78.7	70.2
F1 / %	29.9	81.7	67.2	85.4	58.5	84.1	67.8
Classification Rate / %	89.4	90.9	86.0	92.4	90.2	90.9	90.0

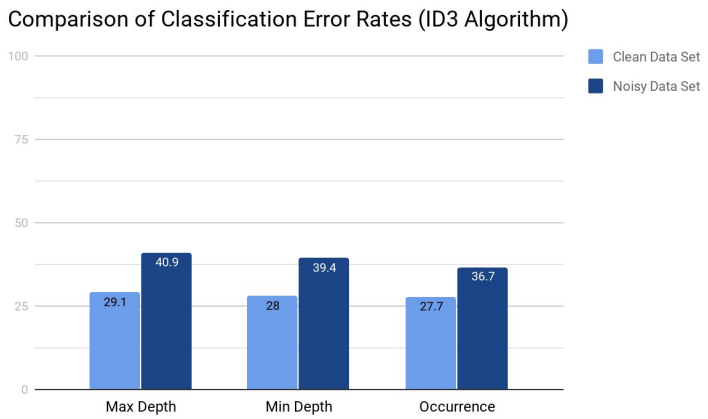
Correct Classifications = 74.8%
Sample Error Rate = 25.2%

Noisy Data Set Analysis

For both the ID3 and Random Forest algorithms, it could be observed from the measures that classes 1 and 5 performs most poorly. This was attributed to the under-representation of the two classes in the distribution of the noisy data set as seen from the sample count for noisy data set: [88 187 187 209 110 220]. We a smaller positive sample to train on, the trees would not perform as well as the others. Conversely, classes 4 and 6 are best performing, also due to their larger positive sample size.

Overall Data Set Analysis - Clean vs Noisy Data Set

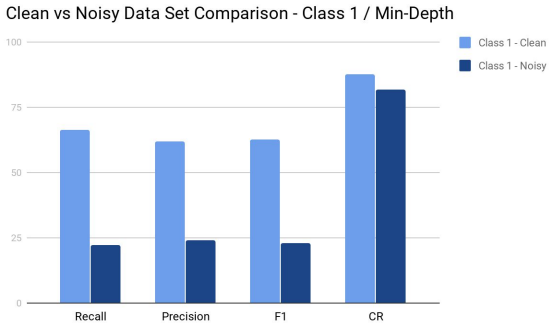
In general, all statistical measures show better results on clean data set compared to noisy data set. The degraded performance using the noisy data set for training can be attributed to two factors: 1) conflicting training examples 2) overfitting of models to noise. Training of the decisions trees on noisy data would result in the fitting of our classifier to noisy data labels, resulting in poor predictions on unseen data. We will now compare the error rates and per-emotion class measures between the two data set.

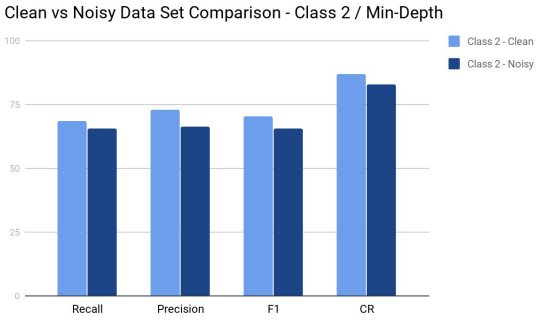


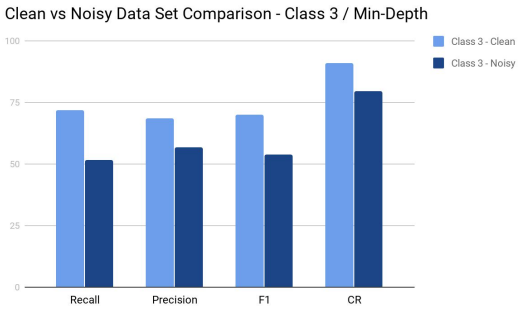
The multi-class binary tree classifiers cross-validated on the clean data set showed better average measures in comparison to the classifier cross-validated with noisy data set. The former classifier reports an average error rate ranging from 27.7% to 29.1% as compared to the classifier trained with the noisy data set with average error rate ranging from 36.7% to 40.9%. All other average measures (precision, recall, F1 and CR) also reflected the differences.

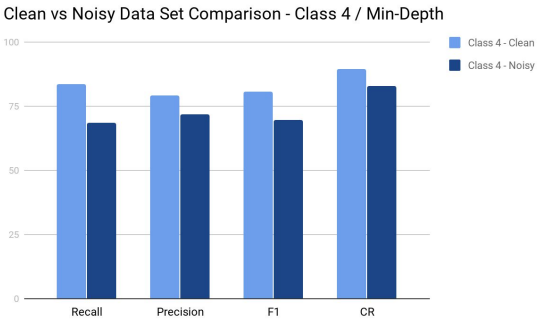
Emotion / Class Comparison

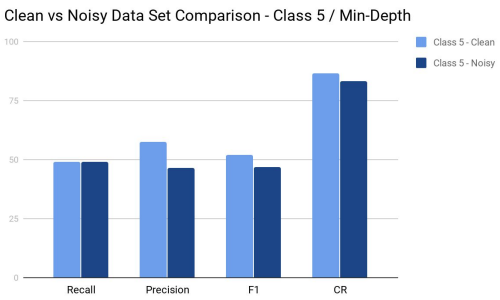
For simplicity, we compare the measures between the clean and noisy data set for each class using just the min-depth ambiguity handling strategy in the charts below. As with our earlier analysis between the two data sets, each class individually shows better performance measures in clean data set than in noisy data.

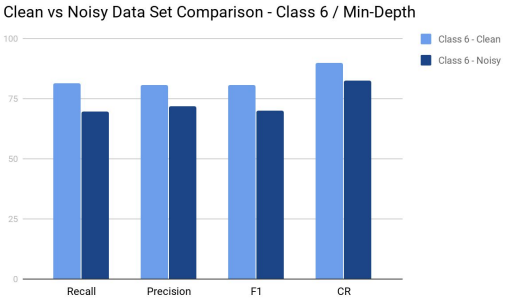












However, Class 1 shows a greater difference in performance measures between the two data sets as it is likely that the number of AUs needed to represent the emotion is greater. Hence it is much more susceptible to noise and misclassification.

Methodology Analysis

Ambiguity Handling Strategies

For the ID3 algorithm in both clean and noisy data set, the occurrence-based ambiguity handling strategy performed slightly better than the max-depth and min-depth ambiguity handling strategy. However, the occurrence method is biased towards the training set as it uses the labels to build the occurrence (i.e. distribution). In cross-validation, we considered this bias in occurrence-based ambiguity handling strategy and made sure that the occurrence information is taken from the validation set (i.e. without considering the test set). However when performing predictions on the entire data set, the occurrence is taken from the labels of the data set, which contributes to this bias.

The averages of the min-depth ambiguity strategy performed better than the max depth as the max-depth ambiguity strategy might use deeper trees to disambiguate, which may be trees that are overfitting. Considering the possibility of bias in the occurrence strategy, the min-depth ambiguity handling strategy may be the best method with the lowest average error rates out of the remaining two strategies.

ID3 vs RDF

RDF utilizes bootstrap aggregation and attribute subsetting (described in page 3) to generate multiple trees for use in classification. Independent deep decision trees trained by the ID3 algorithm tend to have low bias but very high variance, i.e. overfitting to the training set. RDF averages the results of multiple trees that are trained on different subsets of the training data using different attribute sets, resulting in decreased variance of the overall classifier (with a slight increase in bias). This results in a more robust classifier that is more resilient against noise.

The AdaBoost meta-algorithm should, in theory, improve the classification rate of large numbers of 'weak learners' (decision trees in our context) by weighing them based on their classification results on a validation set. However, our results show that there is no statistically significant improvement in the results of our cross-validation process. We note that this could be attributed to two factors: 1) our assumption that the decision trees are 'weak learners' and 2) the small size of the training set. The trees trained by the RF algorithm using 30 attributes are likely to be considered relatively deep to not qualify as a 'weak learner'. It is likely that the AdaBoost would work better with shallower trees. However, it quickly becomes infeasible to train large numbers of trees (upwards of several thousands) to sufficiently cover the entire attribute spectrum. As for point number 2, we are using a portion of the training data to perform validation of the weight parameter, resulting in less training samples for the trees. As mentioned earlier, we do not have sufficient samples to train for some of the classes. It could be possible that the gains in accuracy provided by the AdaBoost algorithm is negated by the loss in accuracy due to a decrease in training samples which explains the similar error rates obtained with the non-AdaBoosted RDF implementation.

Pruning Example

Analysis

The pruning_example function builds decision trees and tests them. The test outputs are then used to graph the tree size (number of leaf nodes) against the cost (likely classification error rate). The blue line and crosses plot the results based on cross-validation on a test set while the red line and circles are based on resubstitution.

On both clean and noisy datasets, small trees incurred high costs (error rates) because these trees are too general (underfit) and tend to incorrectly classify the data. As the tree size increases, the error rate decreases rapidly until 40 for clean and 25 for noisy data as indicated by the square boxes or "best". Nonetheless, the cost will continue to slowly drop until 50 for both, at which point pruning, or branch-trimming may lead to better generalization.

We also observed that in cross-validation, the trees on clean data were able to reach error rates below 0.3, while trees on noisy data troughed above 0.3. The difference between these troughs may indicate the

impact noise during training/testing or the presence of contradictory attributes/labels. This uncertainty leads to greater chances of misclassification.

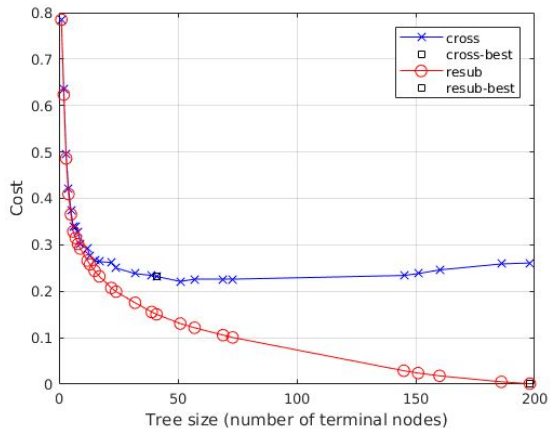
Resubstitution

Resubstitution tests the trained models using the same data set. As expected, the red line drops rapidly and eventually reaches 0 as the trees get larger. Resubstitution is biased and its result are less meaningful because it self-validates, telling nothing about model performance on unseen data. Nonetheless, it is a good way to check if the training algorithm has succeeded. It can also be used to detect underfitting.

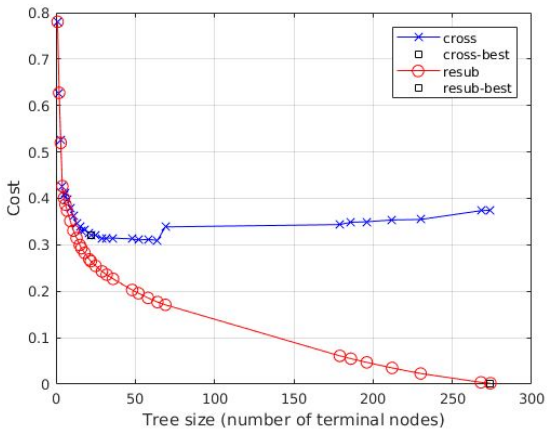
Overfitting

As the tree size grows, the path to the terminal nodes becomes longer and classifications become more specific. However, when we allow our clean tree to pass 60, and noisy tree pass 50 nodes, error rates begin to increase noticeably. Deeper trees are trained to better fit the training data (as evidenced by the monotonically decreasing resubstitution curves). If done excessively, these trees become worse at performing correct predictions on unseen data, a phenomenon known as overfitting. The problem of overfitting is exacerbated for the noisy data set (errors higher than 0.35 for noisy data compared to ~0.25 for clean data). Given a noisy training set, a deep decision tree will use its learning flexibility to learn a model that incorporates the noise structure present in the training set. This highly specific model is not representative on the true distribution and is more likely to make an incorrect prediction on unseen data, resulting in larger cross-validation test error.

Output of prune_example on Clean Data

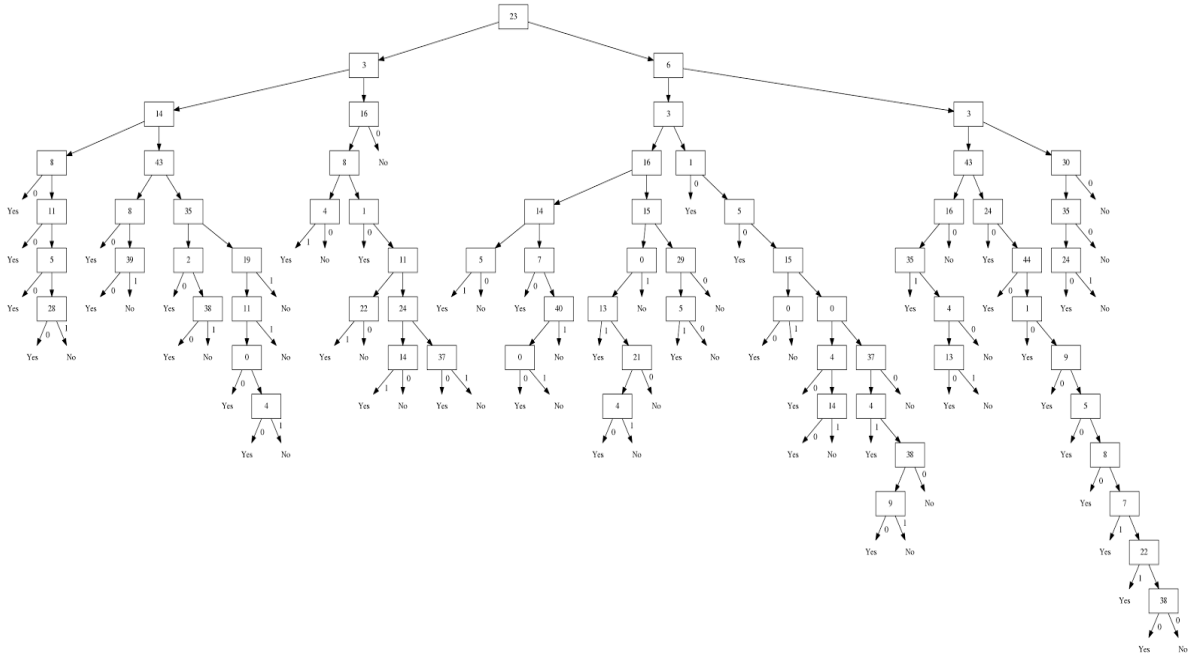


Output of prune_example on Noisy Data

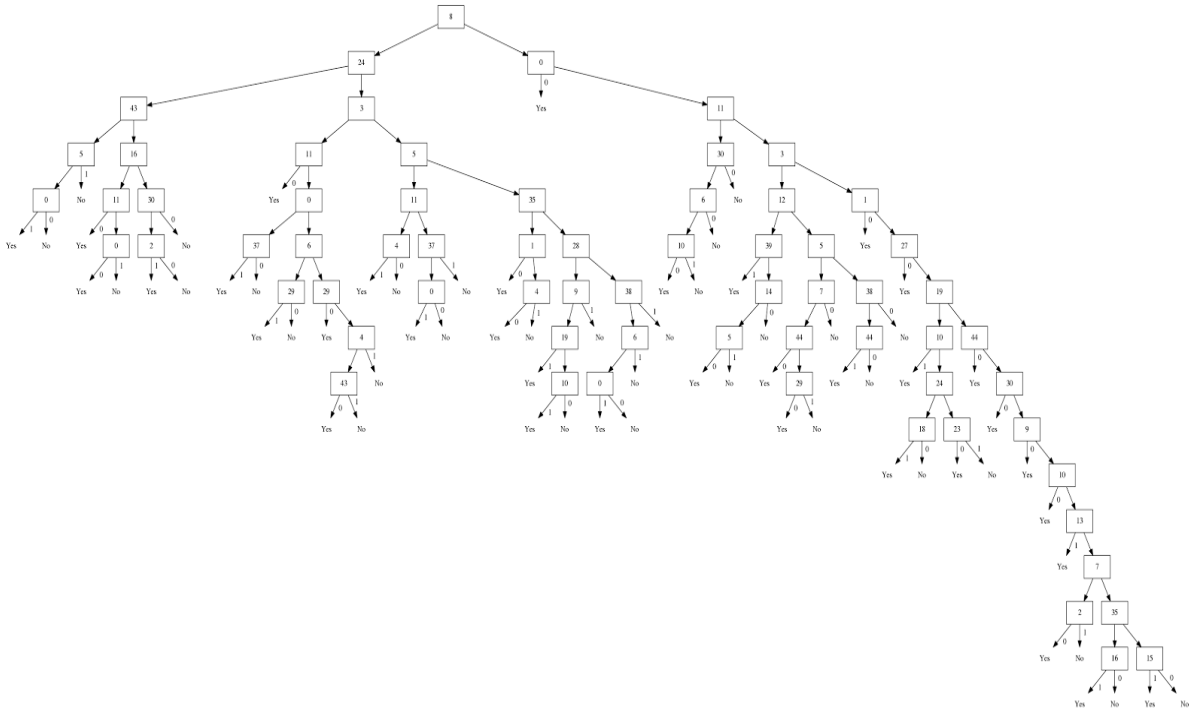


Annex A

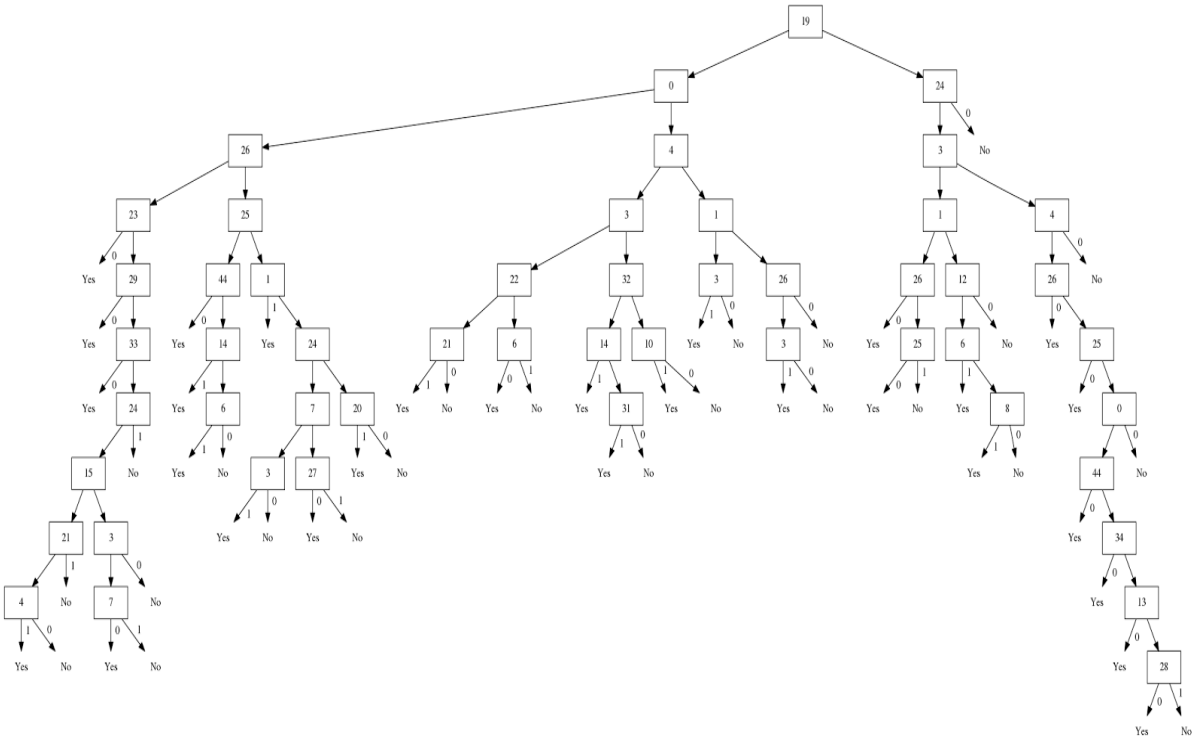
Clean Data - Max Depth - Random



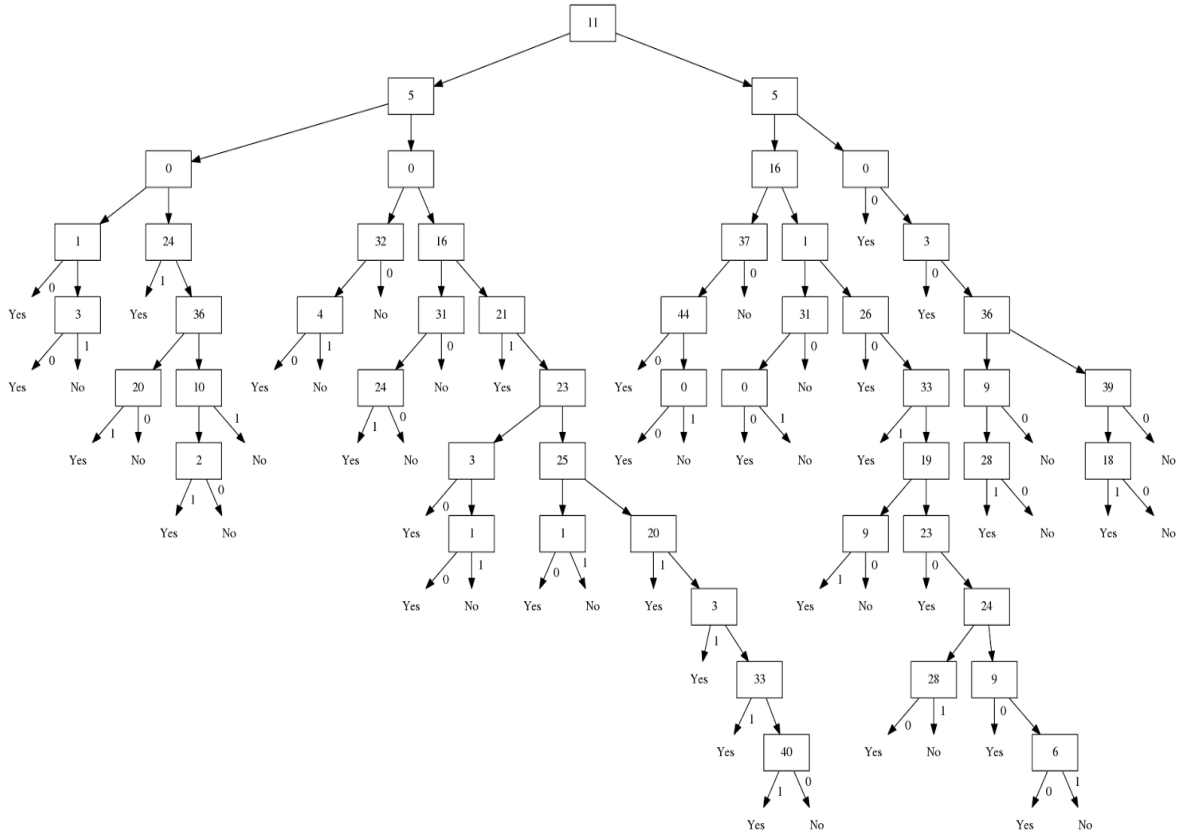
Best Tree by Depth for Class 0 using Random resolution



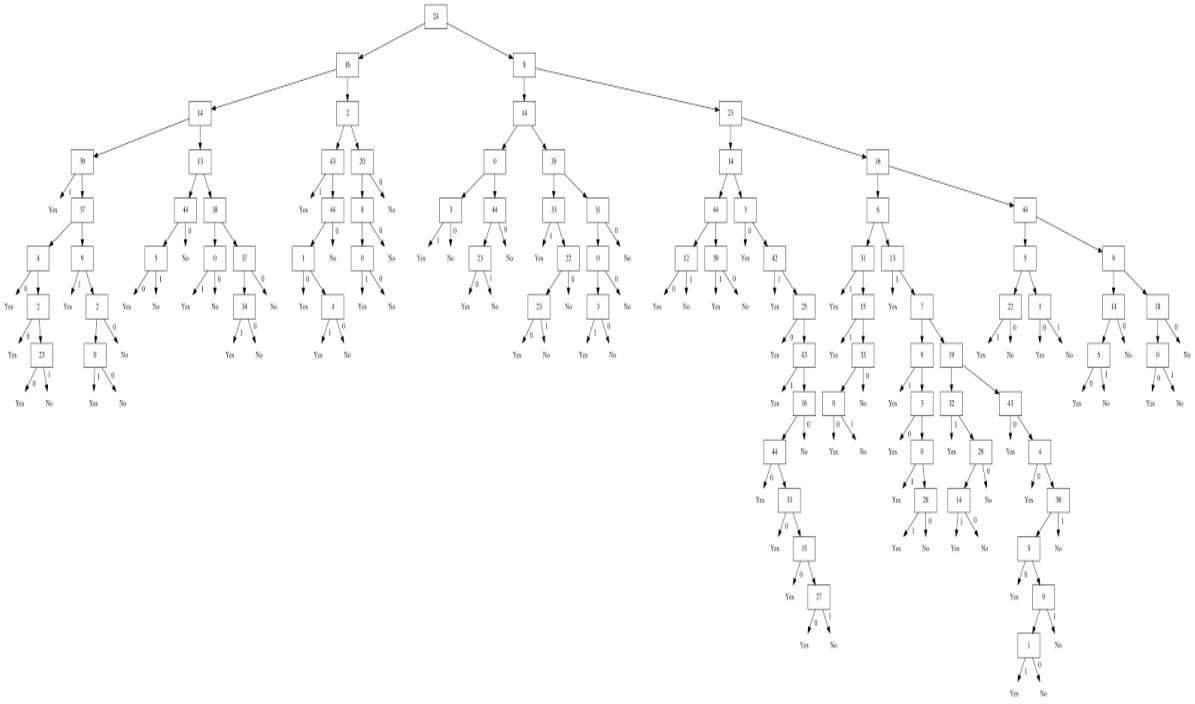
Best Tree by Depth for Class 1 using Random resolution



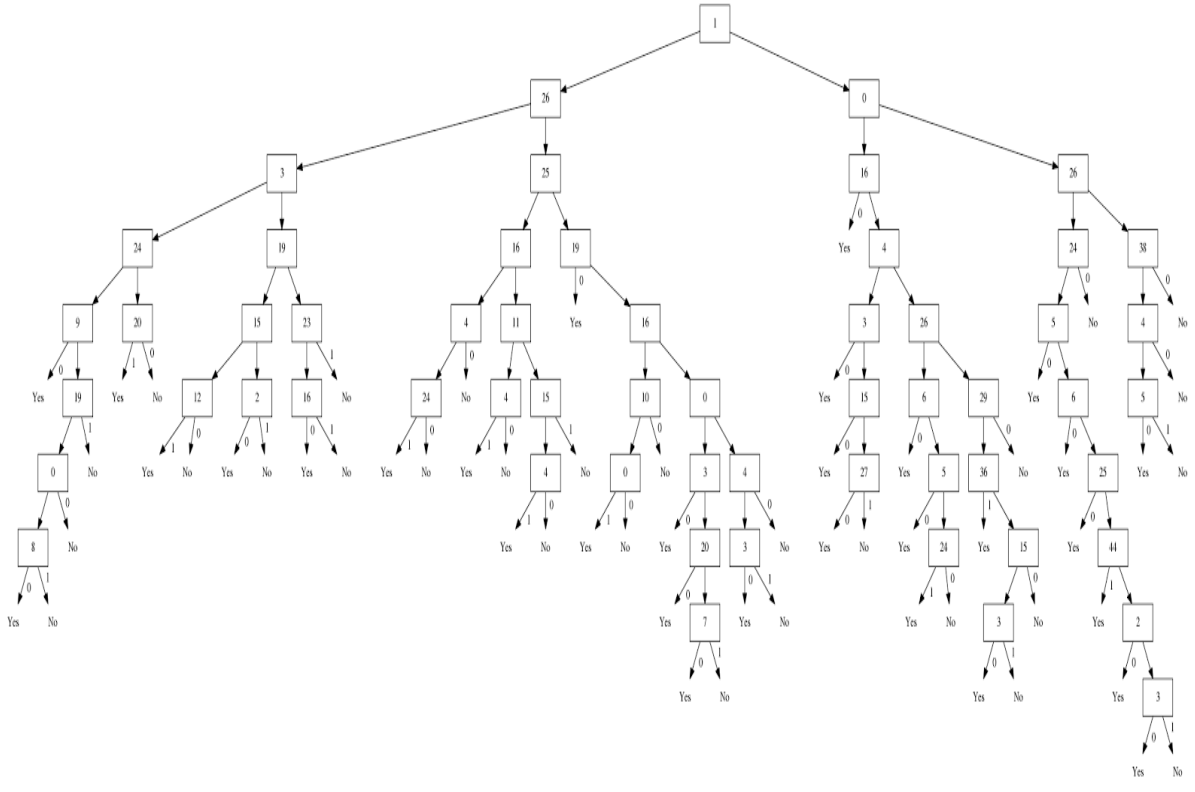
Best Tree by Depth for Class 2 using Random resolution



Best Tree by Depth for Class 3 using Random resolution

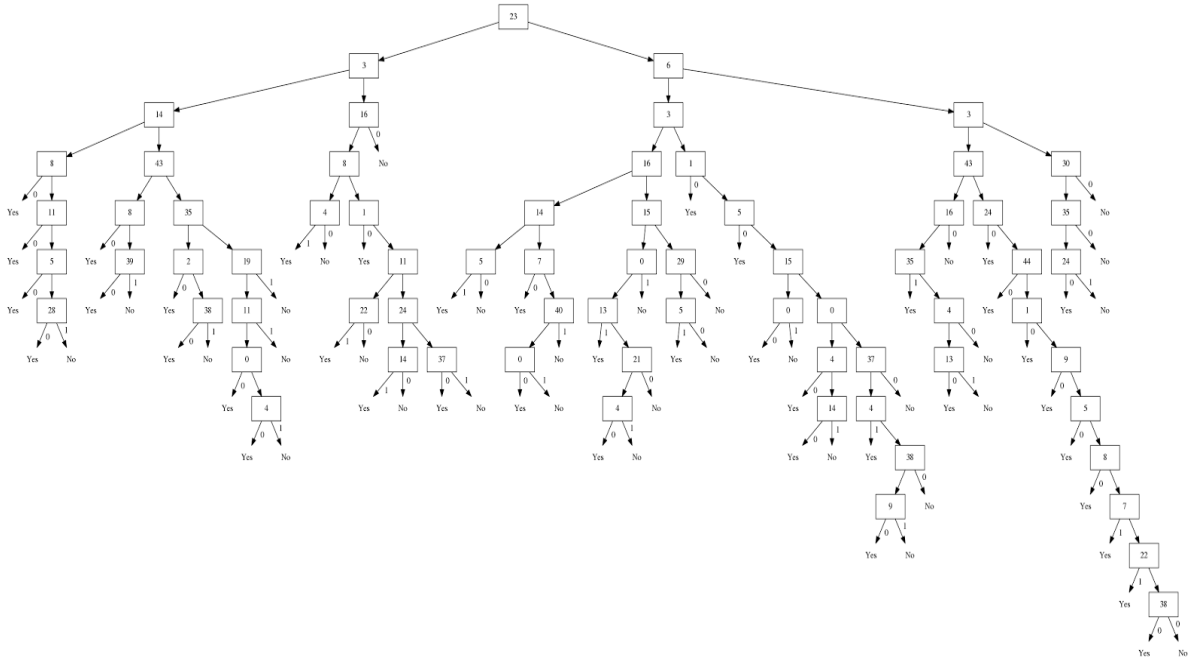


Best Tree by Depth for Class 4 using Random resolution

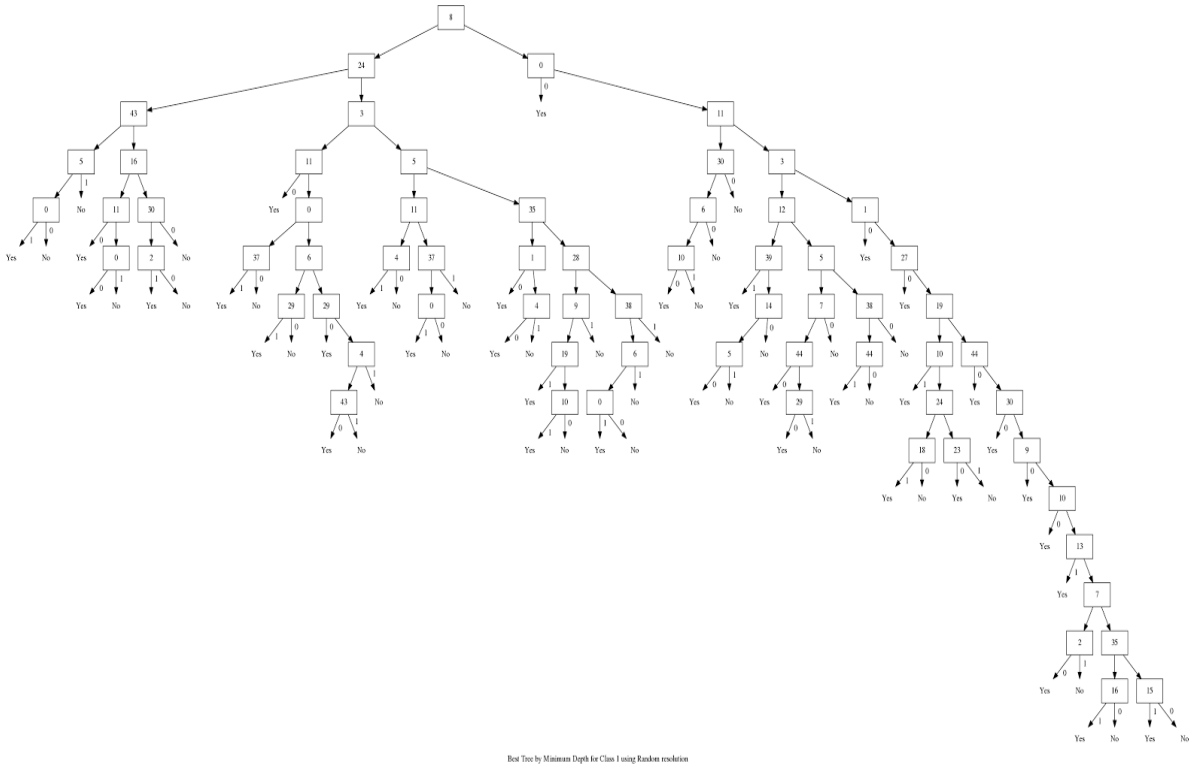


Best Tree by Depth for Class 5 using Random resolution

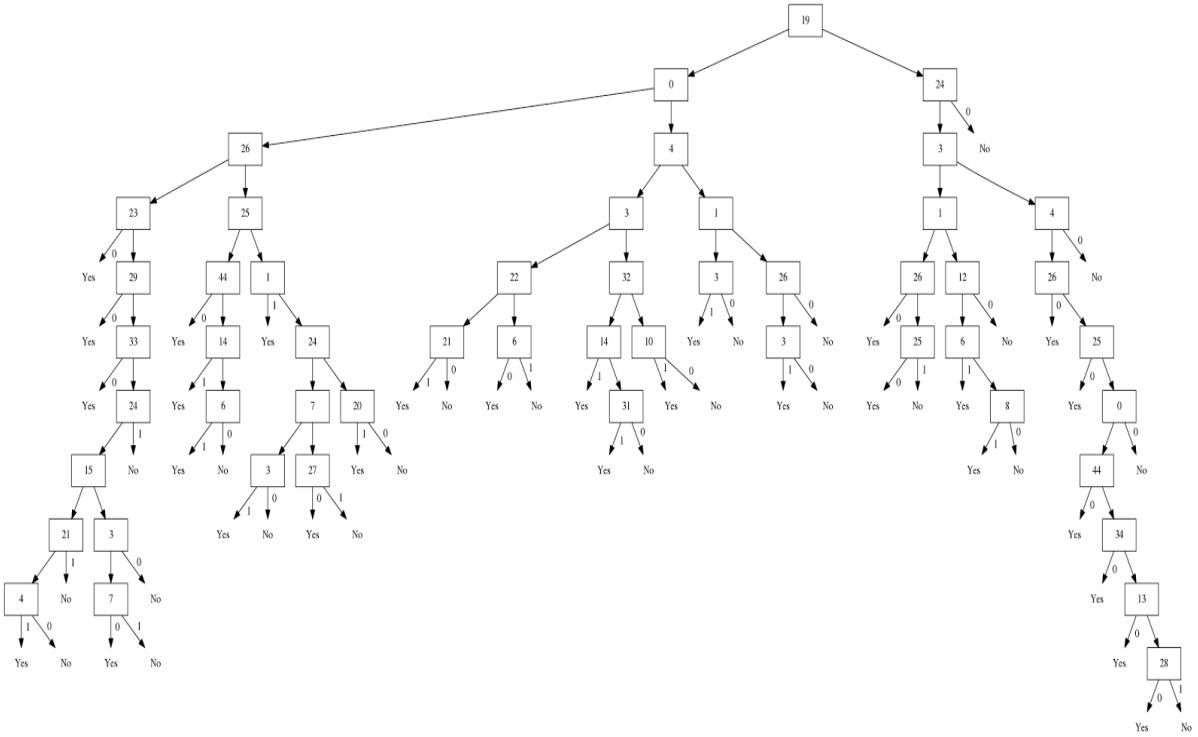
Clean Data - Min Depth - Random



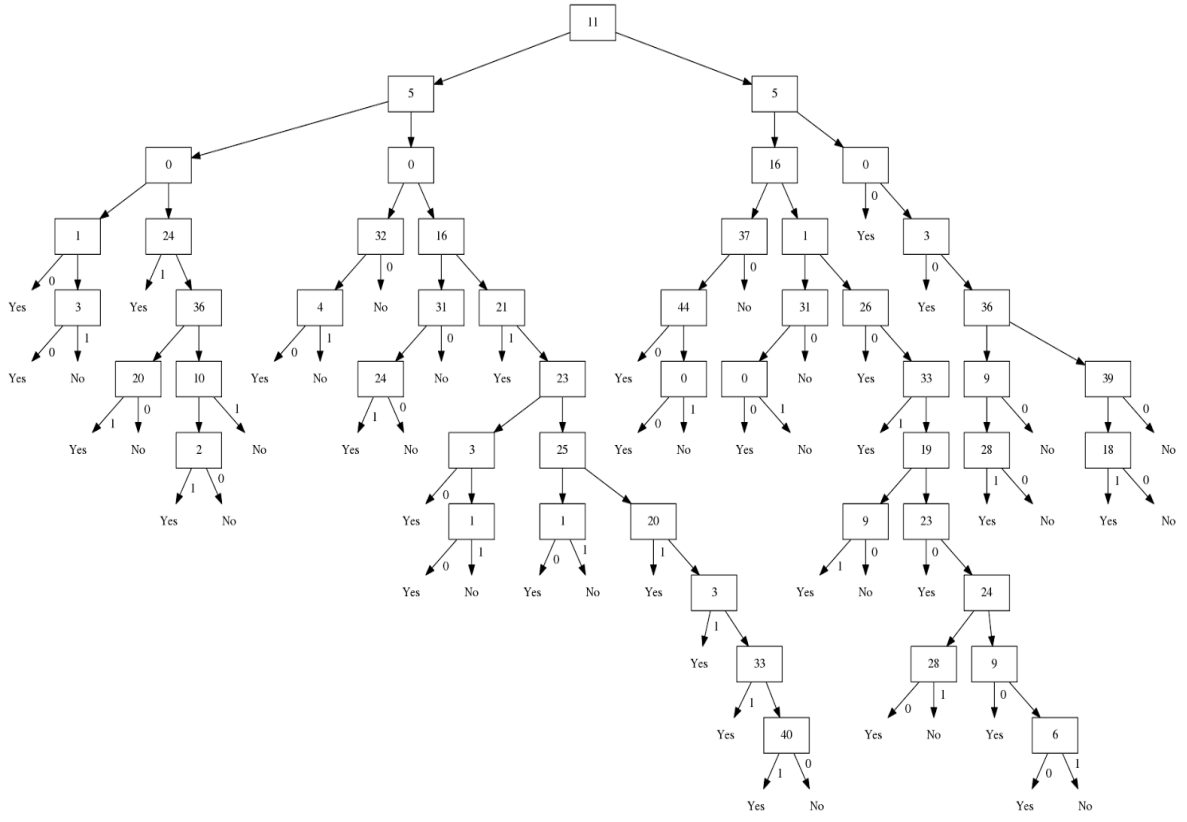
Best Tree by Minimum Depth for Class 0 using Random resolution



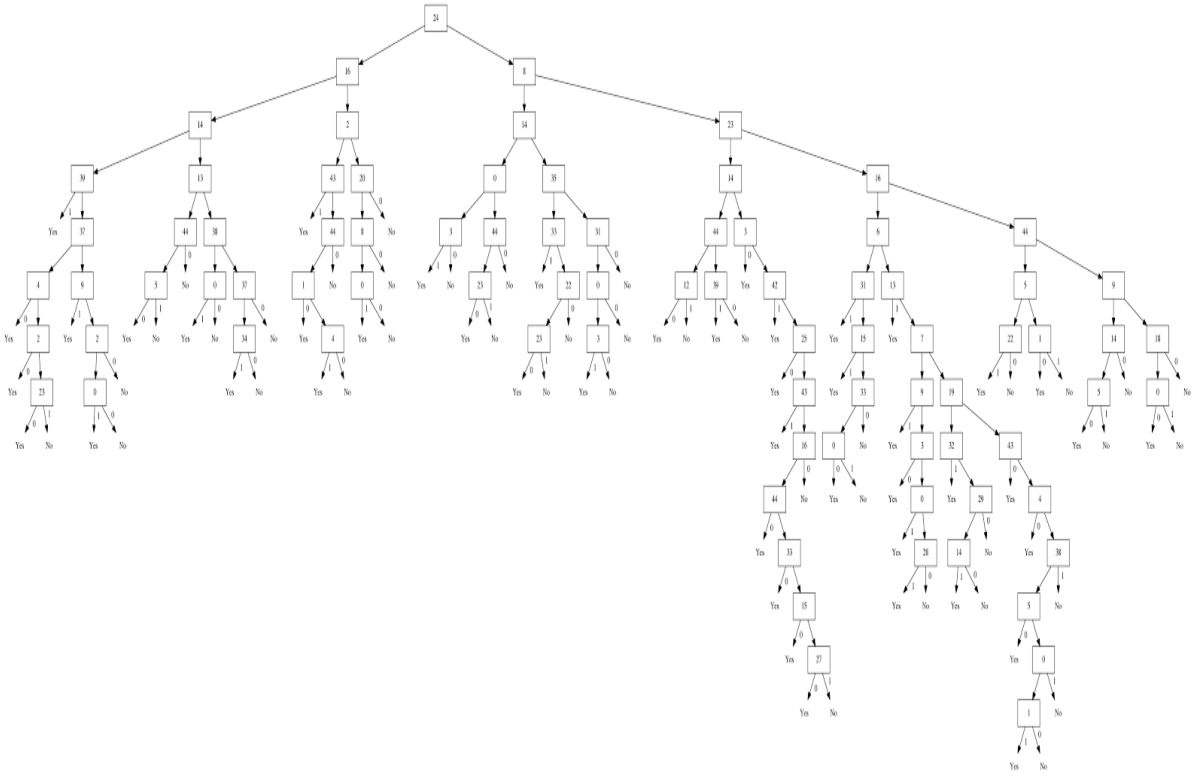
Best Tree by Minimum Depth for Class 1 using Random resolution



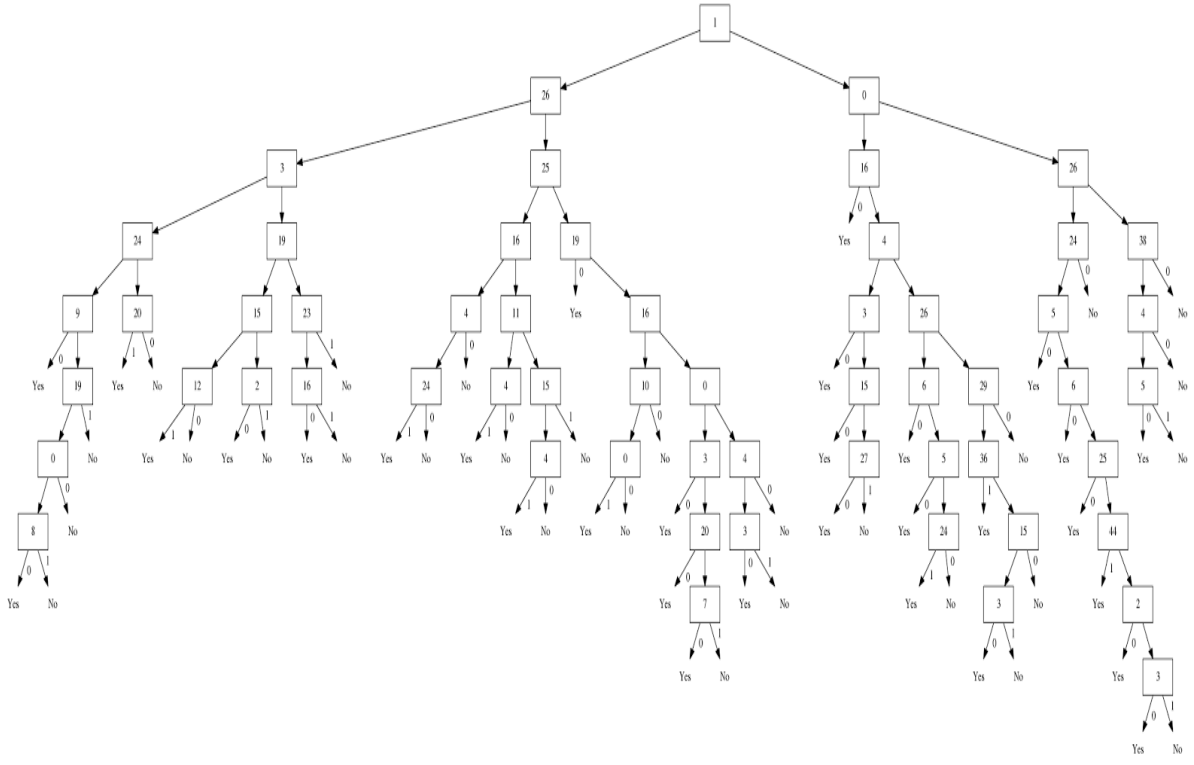
Best Tree by Minimum Depth for Class 2 using Random resolution



Best Tree by Minimum Depth for Class 3 using Random resolution

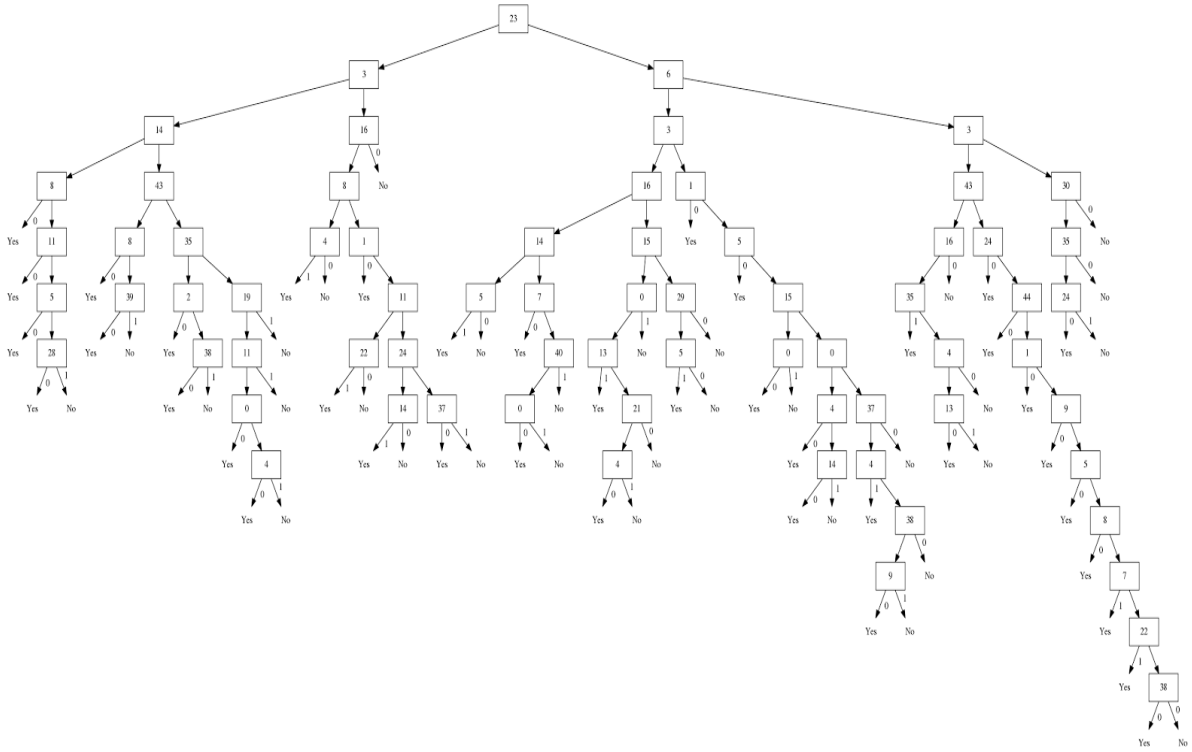


Best Tree by Minimum Depth for Class 4 using Random resolution

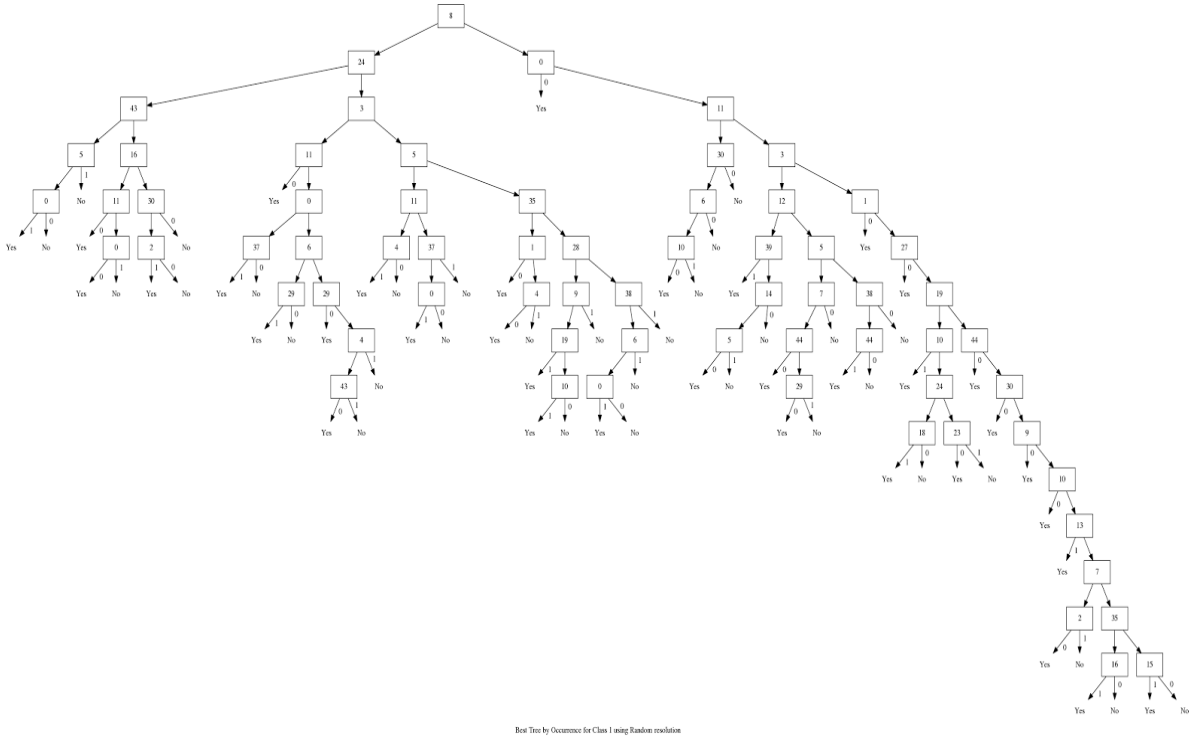


Best Tree by Minimum Depth for Class 5 using Random resolution

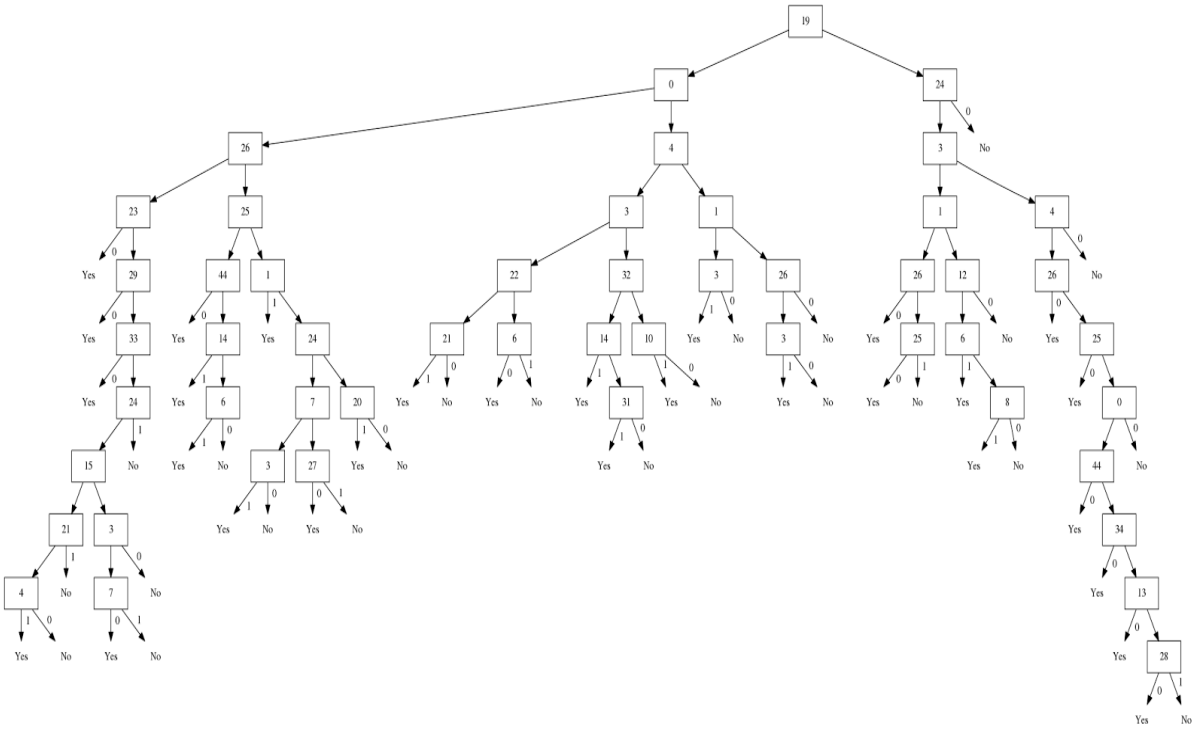
Clean Data - Occurrence - Random



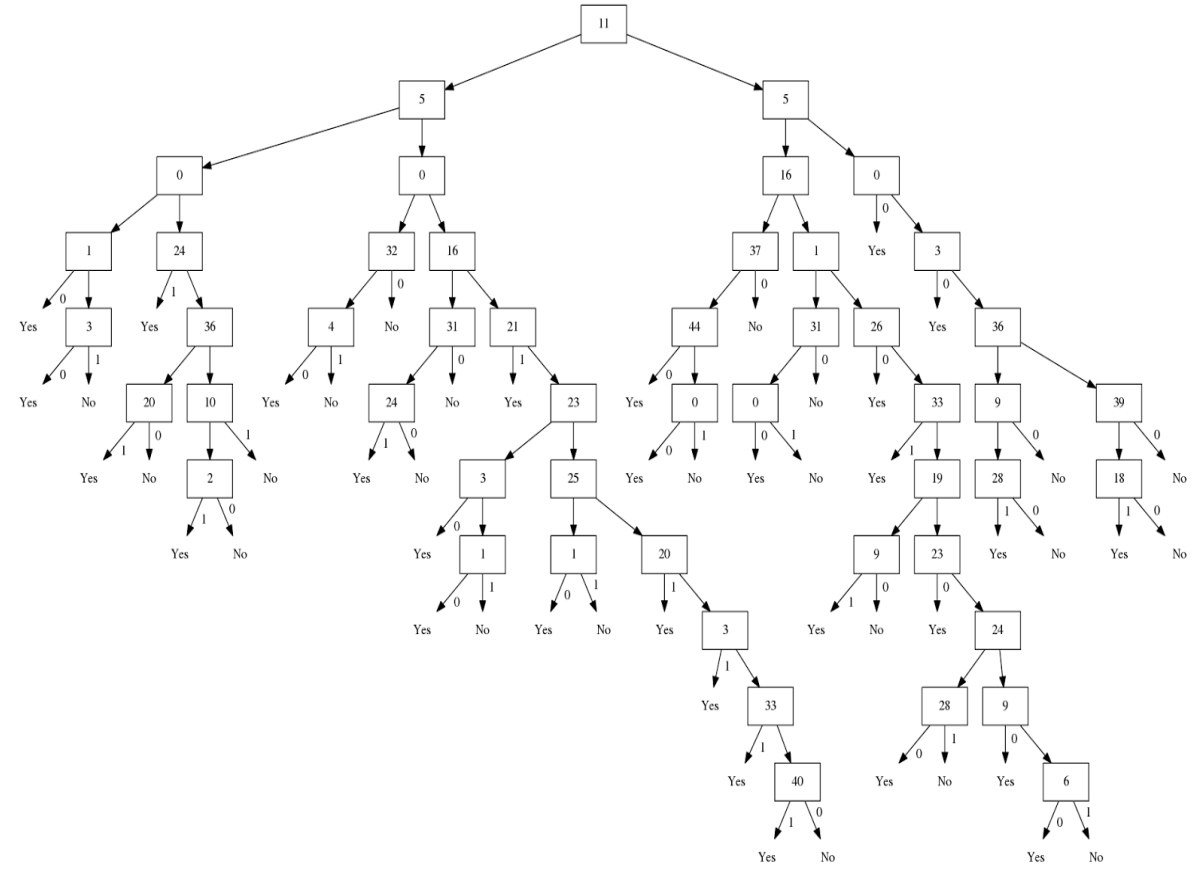
Best Tree by Occurrence for Class 0 using Random resolution



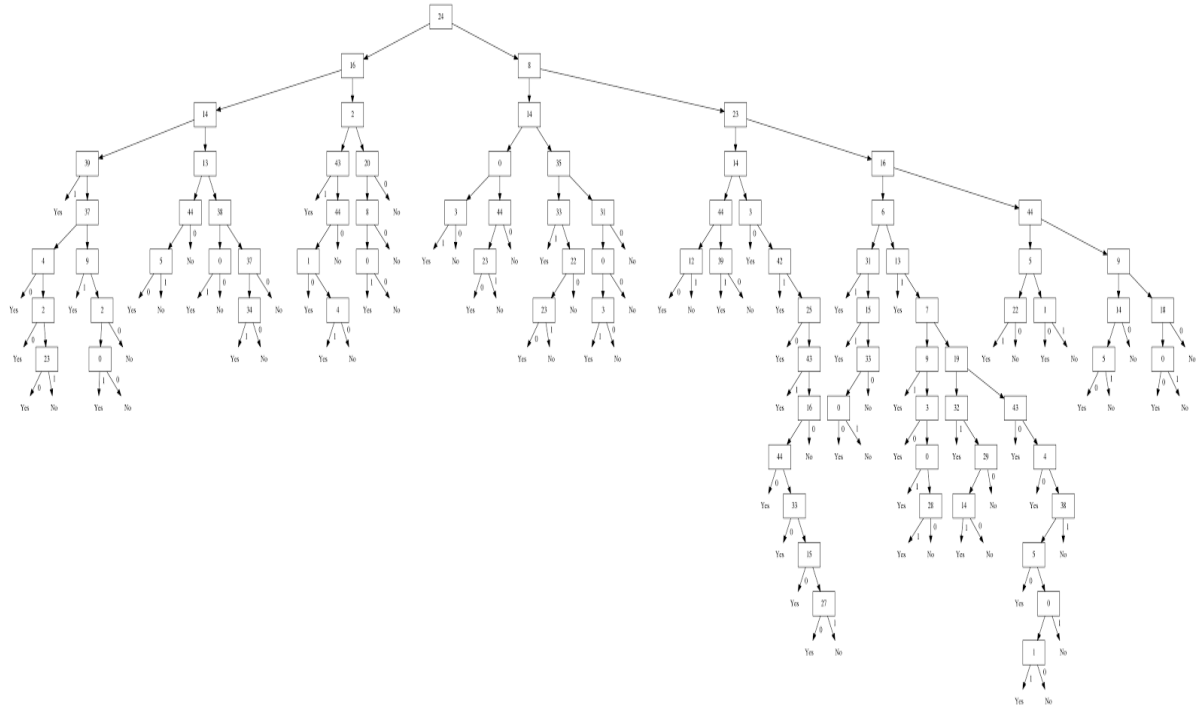
Best Tree by Occurrence for Class 1 using Random resolution



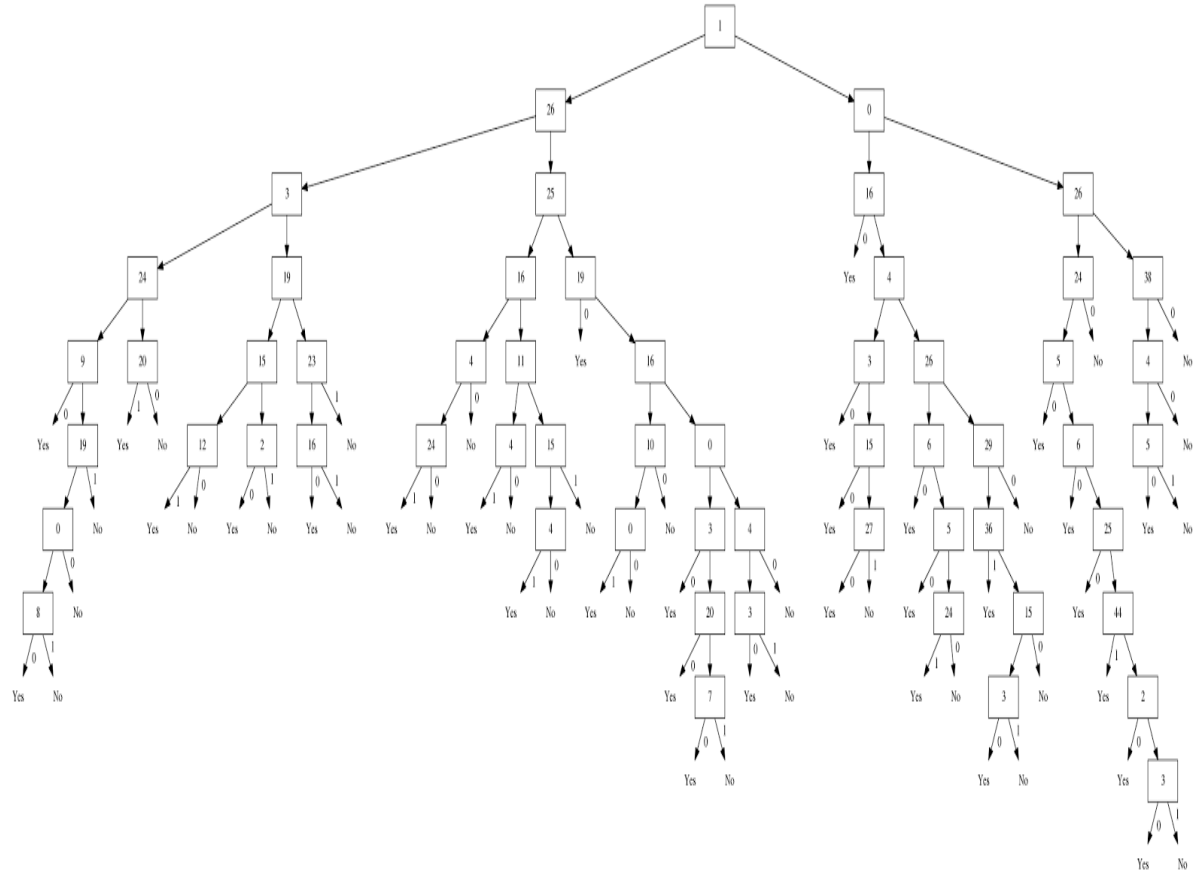
Best Tree by Occurrence for Class 2 using Random resolution



Best Tree by Occurrence for Class 3 using Random resolution



Best Tree by Occurrence for Class 4 using Random resolution



Best Tree by Occurrence for Class 5 using Random resolution