CS412: Introduction to Data Mining, Fall 2017, Homework 4

Name: Nestor Alejandro Bermudez Sarmiento (nab6)

Worked individually

Introduction

This report covers the implementation details and results of two different classification methods applied in four different data sets. The first method explored is Decision Trees: a tree data structure that encodes every possible outcome of a decision. There are many ways to construct such tree. Decision Trees are quite popular because their performance is decent but more importantly, they are easily interpretable by humans.

As in any classification method, over-fitting is a problem. When working with Decision Trees one way to alleviate this problem is to reduce the depth of the tree or set minimum sizes required to split a node or of the split branch itself. These are the hyper-parameters of the classifier and in the following section I'll provide the values used for the different data sets.

The second classification method covered in this report is an ensemble method called Random Forest first developed by Tim Kan Ho in 1995 and later extended by Leo Brieman. Random Forest provide the advantage of 'smoothing out' the error and over-fitting present in normal Decision Trees based on the idea that as the number of trees increase, the chances of over-fitting the same values multiple times decreases.

The hyper-parameters for Random Forest include the number of trees, how much data bagging should be performed, what percentage of the features should be considered for every tree and the hyper-parameters for the Decision Tree itself. It is common to not limit the depth of the tree when using Random Forest because the randomness and the number of trees takes care of the over-fitting problem in this case. I will explore different configurations of hyper-parameters to confirm this.

Implementation details

The classification methods have been implemented using Python and nothing more than the features available in the standard library of the language in order to better grasp the details of the algorithms. I decided to use Python 3 over Python 2 because of the introduction of *lru_cache* which provides an easy way to do some basic caching and enhance performance.

The code follows a very object oriented structure. You will find classes that handle reading the content from the file and creating an internal representation; a **Dataset** class that wraps

the reader class and provides helper methods to split the data by attribute or class and get info about the examples in it; calculate the Gini index; get the metrics given a confusion matrix and, print out the results.

The main programs, **DecisionTree.py** and **RandomForest.py**, take on multiple flags that change the behavior. When run without flags it uses the parameters mentioned in the next section and output simply the confusion matrix. They support flags to calculate and display the different metrics per class, do a grid search to find the appropriate parameters and, re-sample the training data.

The code makes use of the *multiprocessing* package to run some code in parallel. In the case of the Decision Tree, the branch construction happens in parallel for some of the data sets and, in the case of Random Forest, the trees are built in parallel.

Decision Tree

Parameter choices

As mentioned in the Introduction, the hyper-parameters for Decision Tree that I used are: depth, minimum split size (*minsplit*) and minimum leaf size (*minleaf*). The *minleaf* parameter indicates that if, after splitting, the node would contain less than the given amount it should be classify test examples based on the most frequent sibling branch. The following table summarizes the values chosen for every data set.

		Hyper-parameters		
		Depth	Minsplit	Minleaf
Data set	Balance scale	-1	2	1
	Led	5	1	1
	Nursery	-1*	1	-1*
	Synthetic Social	13	66	1

Table 1: Hyper-parameters for every data set evaluated. The value -1 means unbounded.

The values for the parameters were found using Grid Search with increments of 1. In the case of the Nursery data set, the search was interrupted early after seeing that the result with default parameters had a very high accuracy and the parameters so far did not improve it. For all the data sets, the search would be stopped if the execution time took more than 180 seconds.

An additional parameter used is a boolean flag that decides whether to use parallelism to build the tree or not. This flag is **False** for all the data sets except Synthetic Social. The reason for this is that the first three data sets are small enough that the parallelism actually hurts performance because of the hidden cost of shared memory and locks. In the case of Synthetic Social this added cost is outweighed by the benefit of building the branches in parallel.

Random Forest

Parameter choices

On top of the hyper-parameters for the Decision Tree I have three new ones for Random Forest: number of trees (*size*), data bagging (*dbag %*) and feature bagging (*fbag %*). The following table summarizes all the chosen values.

		Hyper-parameters					
		Depth	Minsplit	Minleaf	Size	DBag %	FBag %
Data set	Balance scale	-1	3	8	41	60	25
	Led	-1	1	1	51	67	100
	Nursery	-1	1	-1	101	70	90
	Synthetic Social	-1	3	1	70	96.50	3.16

Table 2: Hyper-parameters for every data set evaluated

For all the data sets except Synthetic Social the values where chosen using Grid Search on the size, dbag and fbag %. The values were constantly incremented by 1 on every case and the search would skip a dimension once the execution time exceeded 180 seconds. This was a valid argument because the execution time increases as the value of each of those parameters increase. The other three parameters (depth, minsplit and minleaf) were chosen by first trying the values from the single Decision Tree analysis; if those didn't work I tried unbounded values; if even after that the accuracy would not improve I did local grid search on those two (with the already fixed values for the other 3 parameters).

For the Synthetic Social data set I tried to use Grid Search but it proven too impractical, the accuracies gotten were really inconsistent. Finally I decided to use a random search on the dbag and fbag % parameters using a fixed size (10). For the parameter values that passed a threshold (60% accuracy) I would then run 10 iterations and get the mean, min and max accuracy obtained from those 10 iterations. Finally I chose the values that gave a high accuracy and low distance between the min and max accuracies.
Once I fixed the dbag and fbag parameters I tried different sizes. Overall, as the size increased the accuracy did too so I decided to take a size that was large enough give a good accuracy and still run under the time constraint imposed (180 seconds).

Results

Decision Tree

Balance Scale Data set

Lets first look at the metrics for the test data set.

		Predicted class		
		1	2	3
Expected class	1	0	14	8
	2	2	81	19
	3	12	16	73

Table 3: Confusion matrix for test data.

Now, the metrics calculated from the confusion matrix.

	Class		
	1	2	3
Accuracy	84.0	77.33	75.56
F-1 Score	0.00	76.06	71.64
F-0.5 Score	0.00	78.03	72.42
F-2 Score	0.00	74.18	72.85
Precision	0.00	72.97	73.00
Recall	0.00	79.41	72.28
Specificity	93.1	75.61	78.23

Table 4: Metrics (percentages) per class for test data set.

Overall accuracy: **68.44%**

Now the results of evaluating the model using the training data set.

		Predicted class		
		1	2	3
Expected class	1	8	7	12
	2	4	166	16
	3	2	15	170

Table 5: Confusion matrix for train data.

And the metrics for the training set confusion matrix.

	Class		
	1	2	3
Accuracy	93.75	89.50	88.75
F-1 Score	39.02	88.77	88.31
F-0.5 Score	32.79	89.06	89.85
F-2 Score	48.19	88.49	86.82
Precision	57.14	88.30	85.86
Recall	29.63	89.25	90.91
Specificity	98.39	89.72	86.85

Table 6: Metrics (percentages) per class for training data set.

Overall accuracy: **86%**

LED Data set

Measurements for the evaluation using the test data.

		Predicted class	
		1	2
Expected class	1	269	82
	2	80	703

Table 7: Confusion matrix for test data.

Per class metrics:

	Class	
	1	2
Accuracy	85.71	85.71
F-1 Score	76.86	89.67
F-0.5 Score	76.73	89.74
F-2 Score	76.99	89.6
Precision	77.08	89.55
Recall	76.64	89.78
Specificity	89.78	76.64

Table 8: Metrics (percentages) per class for training data set.

Overall accuracy: **85.71%**

Measurements when evaluating the classifier with the training data.

		Predicted class	
		1	2
Expected class	1	483	155
	2	158	1291

Table 9: Confusion matrix for train data.

Per class metrics:

	Class	
	1	2
Accuracy	85.00	85.00
F-1 Score	75.53	89.19
F-0.5 Score	75.63	89.13
F-2 Score	75.42	89.24
Precision	75.35	89.28
Recall	75.71	89.10
Specificity	89.10	75.71

Table 10: Metrics (percentages) per class for training data set.

Overall accuracy: **85%**

Nursery Data set

Measurements when evaluating the classifier with the test data.

		Predicted class				
		1	2	3	4	5
Expected class	1	1539	41	16	0	0
	2	30	95	0	0	3
	3	34	0	1502	0	0
	4	0	0	0	1605	0
	5	0	0	0	0	0

Table 11: Confusion matrix for test data.

Per class metrics:

	Class				
	1	2	3	4	5
Accuracy	97.49	98.42	99.01	100.0	99.94
F-1 Score	96.18	71.16	98.43	100.0	0.00
F-0.5 Score	96.26	72.3	98.16	100.0	0.00
F-2 Score	96.11	70.06	98.70	100.0	0.00
Precision	96.06	69.34	98.88	100.0	0.00
Recall	96.3	73.08	97.98	100.0	0.00
Specificity	98.07	99.11	99.49	100.0	99.94

Table 12: Metrics (percentages) per class for test data set.

Overall accuracy: **97.43%**

Measurements when evaluating the model on the training data set.

		Predicted class				
		1	2	3	4	5
Expected class	1	2670	0	0	0	0
	2	0	198	0	0	0
	3	0	0	2508	0	0
	4	0	0	0	2715	0
	5	0	0	0	0	2

Table 13: Confusion matrix for train data.

Per class metrics:

	Class				
	1	2	3	4	5
Accuracy	100.0	100.0	100.0	100.0	100.0
F-1 Score	100.0	100.0	100.0	100.0	100.0
F-0.5 Score	100.0	100.0	100.0	100.0	100.0
F-2 Score	100.0	100.0	100.0	100.0	100.0
Precision	100.0	100.0	100.0	100.0	100.0
Recall	100.0	100.0	100.0	100.0	100.0
Specificity	100.0	100.0	100.0	100.0	100.0

Table 14: Metrics (percentages) per class for training data set.

Overall accuracy: **100%**

Synthetic Social Data set

Measurements when evaluating the classifier with the test data.

		Predicted class			
		1	2	3	4
Expected class	1	117	37	74	40
	2	23	111	55	56
	3	33	39	139	21
	4	54	59	17	125

Table 15: Confusion matrix for test data.

Per class metrics:

	Class			
	1	2	3	4
Accuracy	73.9	73.1	76.1	75.3
F-1 Score	47.27	45.21	53.77	50.30
F-0.5 Score	45.03	45.27	57.30	49.52
F-2 Score	49.74	45.16	50.66	51.10
Precision	51.54	45.12	48.77	51.65
Recall	43.66	45.31	59.91	49.02
Specificity	84.97	82.12	80.99	84.30

Table 16: Metrics (percentages) per class for test data set.

Overall accuracy: **49.2%**

Measurements when evaluating the model on the training data set.

		Predicted class			
		1	2	3	4
Expected class	1	370	87	170	105
	2	65	428	160	102
	3	100	149	479	40
	4	130	151	34	430

Table 17: Confusion matrix for training data.

Per class metrics:

	Class			
	1	2	3	4
Accuracy	78.1	76.2	78.23	81.27
F-1 Score	52.97	54.52	59.47	60.48
F-0.5 Score	51.49	55.8	61.17	58.79
F-2 Score	54.54	53.3	57.85	62.26
Precision	55.64	53.53	56.82	63.52
Recall	50.55	56.69	62.37	57.72
Specificity	86.99	82.76	83.69	89.05

Table 18: Metrics (percentages) per class for training data set.

Overall accuracy: **56.9%**

Random Forest

Balance Scale Data set

Lets first look at the metrics for the test data set.

		Predicted class		
		1	2	3
Expected class	1	0	10	12
	2	0	93	9
	3	0	3	98

Table 19: Confusion matrix for test data.

Now, the metrics calculated from the confusion matrix.

	Class		
	1	2	3
Accuracy	90.22	90.22	89.33
F-1 Score	0.00	89.42	89.09
F-0.5 Score	0.00	90.47	93.69
F-2 Score	0.00	88.40	84.92
Precision	0.00	87.74	82.35
Recall	0.00	91.18	97.03
Specificity	100.0	89.43	83.43

Table 20: Metrics (percentages) per class for test data set.

Overall accuracy: **84.89%**

Now the results of evaluating the model using the training data set.

		Predicted class		
		1	2	3
Expected class	1	0	12	15
	2	0	169	17
	3	0	6	181

Table 21: Confusion matrix for train data.

And the metrics for the training set confusion matrix.

	Class		
	1	2	3
Accuracy	93.25	91.25	90.5
F-1 Score	0.00	90.62	90.50
F-0.5 Score	0.00	90.76	94.17
F-2 Score	0.00	90.47	87.10
Precision	0.00	90.37	84.98
Recall	0.00	90.86	96.79
Specificity	100.0	91.59	84.98

Table 22: Metrics (percentages) per class for training data set.

Overall accuracy: **87.5%**

LED Data set

Measurements for the evaluation using the test data.

		Predicted class	
		1	2
Expected class	1	268	83
	2	71	712

Table 23: Confusion matrix for test data.

Per class metrics:

	Class	
	1	2
Accuracy	86.42	86.42
F-1 Score	77.68	90.24
F-0.5 Score	76.88	90.65
F-2 Score	78.50	89.83
Precision	79.06	89.56
Recall	76.35	90.93
Specificity	90.93	76.35

Table 24: Metrics (percentages) per class for training data set.

Overall accuracy: **86.42%**

Measurements when evaluating the classifier with the training data.

		Predicted class	
		1	2
Expected class	1	479	159
	2	135	1314

Table 25: Confusion matrix for train data.

Per class metrics:

	Class	
	1	2
Accuracy	85.91	85.91
F-1 Score	76.52	89.94
F-0.5 Score	75.65	90.38
F-2 Score	77.41	89.50
Precision	78.01	89.21
Recall	75.08	90.68
Specificity	90.68	75.08

Table 26: Metrics (percentages) per class for training data set.

Overall accuracy: **85.91%**

Nursery Data set

Measurements when evaluating the classifier with the test data.

		Predicted class				
		1	2	3	4	5
Expected class	1	1549	25	22	0	0
	2	37	93	0	0	0
	3	28	0	1508	0	0
	4	0	0	0	1605	0
	5	0	0	0	0	0

Table 27: Confusion matrix for test data.

Per class metrics:

	Class				
	1	2	3	4	5
Accuracy	97.70	98.73	98.97	100.0	100.0
F-1 Score	96.51	75.00	98.37	100.0	0.00
F-0.5 Score	96.84	72.88	98.25	100.0	0.00
F-2 Score	96.19	77.24	98.48	100.0	0.00
Precision	95.97	78.81	98.56	100.0	0.00
Recall	97.06	71.54	98.18	100.0	0.00
Specificity	98.01	99.47	99.34	100.0	100.0

Table 28: Metrics (percentages) per class for test data set.

Overall accuracy: **97.7%**

Measurements when evaluating the model on the training data set.

		Predicted class				
		1	2	3	4	5
Expected class	1	2666	0	4	0	0
	2	2	196	0	0	0
	3	6	0	2502	0	0
	4	0	0	0	2715	0
	5	0	0	0	0	0

Table 29: Confusion matrix for train data.

Per class metrics:

	Class				
	1	2	3	4	5
Accuracy	99.85	99.95	99.88	100.0	99.98
F-1 Score	99.78	98.99	99.80	100.0	0.00
F-0.5 Score	99.82	98.99	99.78	100.0	0.00
F-2 Score	99.73	98.99	99.82	100.0	0.00
Precision	99.70	98.99	99.84	100.0	0.00
Recall	99.85	98.99	99.76	100.0	0.00
Specificity	99.85	99.97	99.93	100.0	100.0

Table 30: Metrics (percentages) per class for training data set.

Overall accuracy: **99.83%**

Synthetic Social Data set

For this data set I'll present two different results. One of them with the maximum number of trees that would still run under 180 seconds in my computer and the other one executed in EWS also under the time constraint.

When running in my computer I could set the number of trees to 200 and the execution took **168.36** seconds. The results are these:

		Predicted class			
		1	2	3	4
Expected class	1	208	3	20	37
	2	9	164	32	40
	3	24	24	178	6
	4	20	22	2	211

Table 31: Confusion matrix for test data using 200 trees.

Per class metrics for evaluation using the test data:

	Class			
	1	2	3	4
Accuracy	88.70	87.00	89.20	87.30
F-1 Score	78.64	71.62	76.72	76.87
F-0.5 Score	68.73	45.27	76.72	80.29
F-2 Score	79.27	74.75	76.72	73.22
Precision	79.69	77.00	76.72	71.77
Recall	77.61	66.94	76.72	82.75
Specificity	92.76	93.51	92.97	88.86

Table 32: Metrics (percentages) per class for test data set using 200 trees.

Overall accuracy: **76.1%**

I also ran the same setup against the training data set and these are the results.

		Predicted class			
		1	2	3	4
Expected class	1	731	0	0	1
	2	0	755	0	0
	3	0	0	768	0
	4	0	0	0	745

Table 33: Confusion matrix for training data using 200 trees.

Per class metrics for evaluation using the training data:

	Class			
	1	2	3	4
Accuracy	99.97	100.0	100.0	99.97
F-1 Score	99.93	100.0	100.0	99.93
F-0.5 Score	99.89	100.0	100.0	99.97
F-2 Score	99.97	100.0	100.0	99.89
Precision	100.0	100.0	100.0	99.87
Recall	99.86	100.0	100.0	100.0
Specificity	100.0	100.0	100.0	99.96

Table 34: Metrics (percentages) per class for training data set using 200 trees.

Overall accuracy: **99.97%**

When running in EWS I had to lower the number of trees to make it run under 180 seconds. Instead of using 200 trees I used 70. The execution time was 147.01 seconds These are the results:

		Predicted class			
		1	2	3	4
Expected class	1	205	6	28	29
	2	12	165	30	38
	3	24	26	180	2
	4	30	20	1	204

Table 35: Confusion matrix for test data using 70 trees.

Per class metrics for evaluation using the test data:

	Class			
	1	2	3	4
Accuracy	87.10	86.8	88.90	88.00
F-1 Score	76.07	71.43	76.43	77.27
F-0.5 Score	76.32	68.92	77.12	78.89
F-2 Score	75.81	74.12	75.76	75.72
Precision	75.65	76.04	75.31	74.73
Recall	76.49	67.35	77.59	80.0
Specificity	90.98	93.11	92.32	90.74

Table 36: Metrics (percentages) per class for test data set using 70 trees.

Overall accuracy: **75.4%**

Finally, lets look at the results of running the Random Forest with 70 trees and evaluate it using the training data.

		Predicted class			
		1	2	3	4
Expected class	1	732	0	0	0
	2	0	755	0	0
	3	0	0	768	0
	4	0	0	0	745

Table 37: Confusion matrix for training data using 70 trees.

Per class metrics for evaluation using the training data:

	Class			
	1	2	3	4
Accuracy	100.0	100.0	100.0	100.0
F-1 Score	100.0	100.0	100.0	100.0
F-0.5 Score	100.0	100.0	100.0	100.0
F-2 Score	100.0	100.0	100.0	100.0
Precision	100.0	100.0	100.0	100.0
Recall	100.0	100.0	100.0	100.0
Specificity	100.0	100.0	100.0	100.0

Table 38: Metrics (percentages) per class for training data set using 70 trees.

Overall accuracy: **100.0%**

Conclusions

It is evident that in general ensemble methods, specifically Random Forest, yield a better performance than a single Decision Tree, as shown in the charts below.

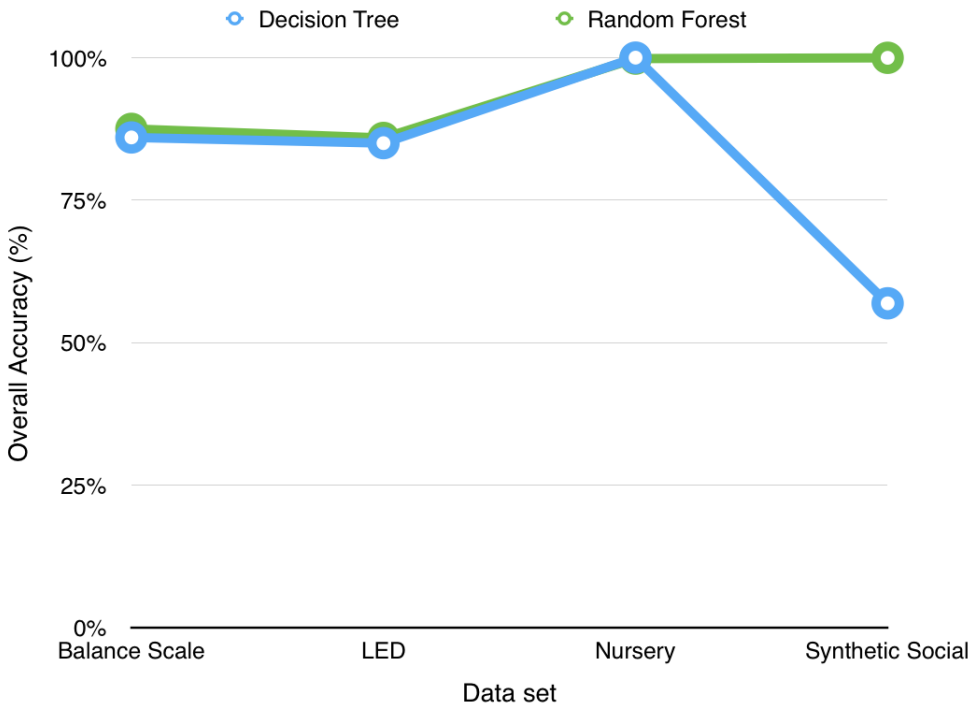

Figure 1: Comparison of overall accuracy when evaluated in the training data set.

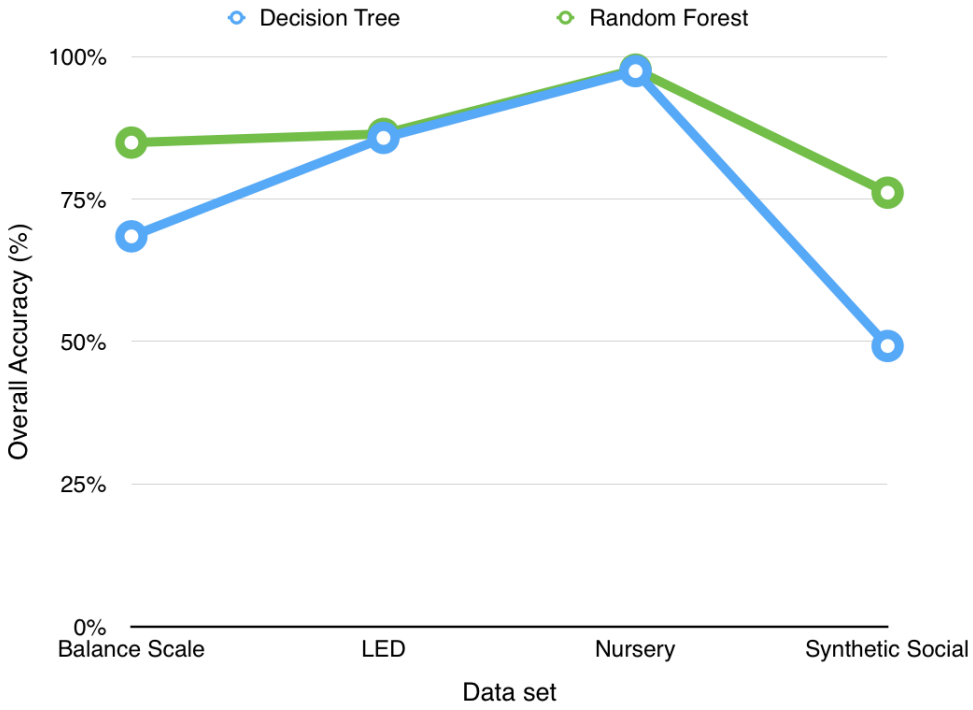

Figure 2: Comparison of overall accuracy when evaluated in the test data set.

From the four data sets considered the Nursery data set did not improve considerable when using Random Forest. Another thing to note is that there are only two examples of the class 5 in this data set. The accuracy of the Random Forest oscilated between 97.4 and 97.7% depending on whether those two elements ended up included in the data bag for some of the trees or not. Being higher when not included, which makes sense since there are no examples for the class in the test data so there is no chance of miss-classification.

Everything was executed in a MacBook Pro Mid 2015, 16GB RAM, i7 Processor and validated that would run under time constraints in the EWS Linux system (shared 128GB RAM, 24 cores). Even though the EWS system is more powerful the performance was better in my personal laptop. It seems to be a cap on the CPU time you can use simultaneously in EWS. When using the 24 cores I was only using like 15% on each but when I changed to using only 5 cores they were all performed at 95%.

Balance Scale Data set

There was a 23% improvement when using Random Forest. As we can see in the confusion matrix for the decision tree, the main problem was that many samples where classified as class 1 and all of them were incorrectly done so. The reason for this is that there are only 27 examples of class 1 in our training data set or 6% of the data; which is not good enough to characterize it. When using Random Forest, the classifier never predicted a class 1 (because

of how rare it was) effectively improving the accuracy and specificity for class 1. So, Random Forest works in this case because it reduces the chances of predicting a class one by multiple trees given that they are rare occurrences.

LED Data set

For this data set Random Forest did improve performance but by a very small amount. Which makes sense considering that we are including all the features (no feature bagging) in the Random Forest. Another reason is that the data set only contained 7 attributes and Decision Tree was using a depth of 5 so we were only leaving out two attributes. The other hyper-parameters remained the same between Decision Tree and Random Forest. Note that we are not really over-fitting the data; we are achieving roughly the same accuracy for the training and test data set which probably indicates that there isn't a perfectly clear way to split the space in two.

An important note is that the source of this data set mentions that a Naive Bayes achieves 74% accuracy so Decision Tree and Random Forest definitely out-perform Naive Bayes in this case.

Nursery Data set

Although the results for Random Forest show an slight improvement over the Decision Tree, there were some cases in which the overall accuracy was slightly below 97.43%. The only logical explanation for this is that, because of the number of trees and data bagging, there were some branches that were noisy.

For this particular data set the improvement is so small that, in my opinion, does not justify the added computational cost of running Random Forest with 101 trees.

Synthetic Social Data set

For this particular data set the improvement given by Random Forest is considerable. The classifier went from **49.2%** to **76.1%**; that represents a **54%** improvement! It is worth noting that we definitely over-fitting our data since the accuracy for the training data set is a perfect 100.0%. Attempts to fix this problem were made by increasing the size of the leaves and minimum size required to split a node but it did not make any improvement.

Annex: Class distributions

Looks like the data was split while keeping the proportion of examples for each class as shown in the charts below.

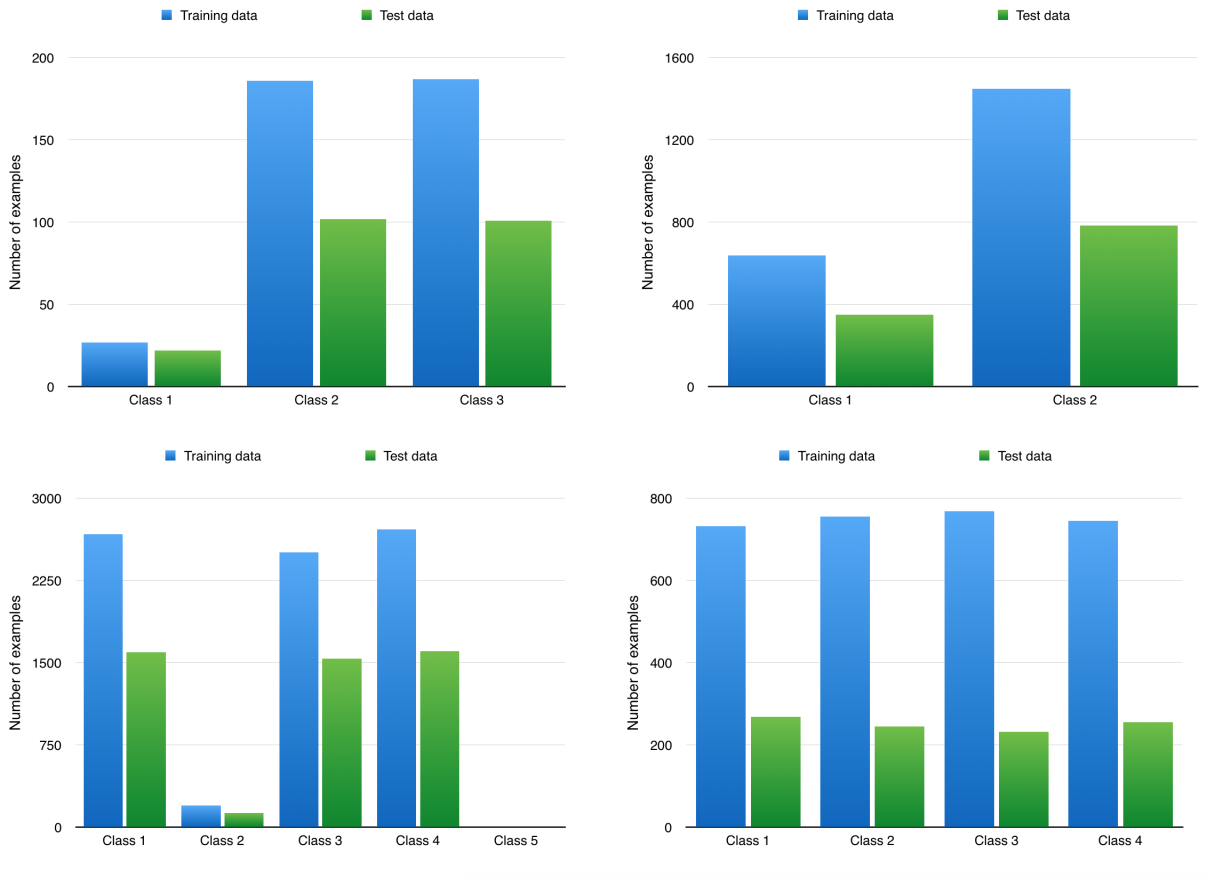

Figure 3: From left to right, top to bottom: class distribution for Balance Scale, LED, Nursery and Synthetic Social data sets.