Assignment 1: Classification Trees, Bagging and Random Forests

Leek, Craig (5689279)
c.q.h.d.leek@uu.nl

De Lange, Thomas (6495567)
t.r.c.delange@students.uu.nl

October 2018

1 Introduction

In this short review, we would like to test the predictive power of a classification tree designed by us in R. We would like to do this by looking if it is possible to predict the occurrence of a post-release bug. We will start with a short review on the data-set, after that, we will look at the methods used and in conclusion, we will state the results and their implications.

The data-set used is part of the Eclipse data-set provided by the Eclipse Foundation. The Eclipse Foundation obtained the data via a system that tracks bugs that are reported by community members. The bugs are found in the open source software that the Foundation provides. There are over 350 open source projects for a wide range of technologies and domains. The aim of the foundation is to drive business sustainability and profitability (Eclipse, 2018). For the training set, the package level data will be analyzed, this is part of release 2.0. The data of release 3.0 will be used as a test set.

For this analysis, only a portion of the total data-set is used. It consists of the metrics that can be found in table 1. On a file level the value is used for ACD, NOI, NOT, and TLOC. For all other metrics the average, maximum and total values are an element of the data-set. With the inclusion of pre-release bugs, this gives a total of 41 columns. The chosen data corresponds to the paper of Zimmerman, Premraj and Zeller (Zimmermann, Premraj, & Zeller, 2007).

2 Methods

To examine the strength of the predictive model we look at whether or not a post-release bug has occurred and compare it to the predictions obtained by using our classification tree.

The following three measurements will be used to analyze the quality of a classification.

	Abbreviation	Metric
methods	FOUT	Number of method calls (fan out)
	MLOC	Method lines of code
	NBD	Nested block depth
	PAR	Number of parameters
	VG	McCabe cyclomatic complexity
Classes	NOF	Number of fields
	NOM	Number of methods
	NSF	Number of static fields
	NSM	Number of static methods
files	ACD	Number of anonymus type declarations
	NOI	Number of interfaces
	NOT	Number of classes
	TLOC	Total lines of code

Table 1: Content of the data-set

1. Accuracy, indicates the amount of correct classifications and can be calculated as followed $(TP+TN)/(TP+TN+FP+FN)$. (TP = True positive, TN = true negative, FP = false positive, FN = false negative).

2. Recall, that states the relation between TP and the amount of the total defects $TP/(TP+FN)$.

3. Precision, which shows the relation between true positives and the total number of files that are defect-prone $TP/(TP+FP)$.

We will use our algorithm in three distinct way's. First, we will train a single classification tree using the parameters nmin = 15, minleaf = 15 and nfeat = 41. Secondly, we will build a model using bagging with the same parameters as in our single tree with an addition of m = 100. Thirdly, we will use a random forest with the same parameters as in the second model except for nfeat = 6.

Our predictions are that quality scores will increase when we use the bagging and random forest algorithm compared to a single tree. We also think that bagging will have a better score than a random forest. This because the split with bagging always uses the optimal variable opposed to the local optimum used with random forest.

3 Results and Conclusion

In this section, we will show the results obtained from our analysis. We will start by discussing the splits done by a single tree. After that, we will look at the results of the different quality measures that we named.

In figure 1 the first three splits of the classification tree are depicted. The first split is done on the pre-release bugs. It is not unthinkable that bugs may not be fixed before a product goes into production. By this standard, knowing the amount of pre-release bugs is a good indicator for post-release bugs. The second and third split is done on the maximum McCabe cyclomatic complexity,

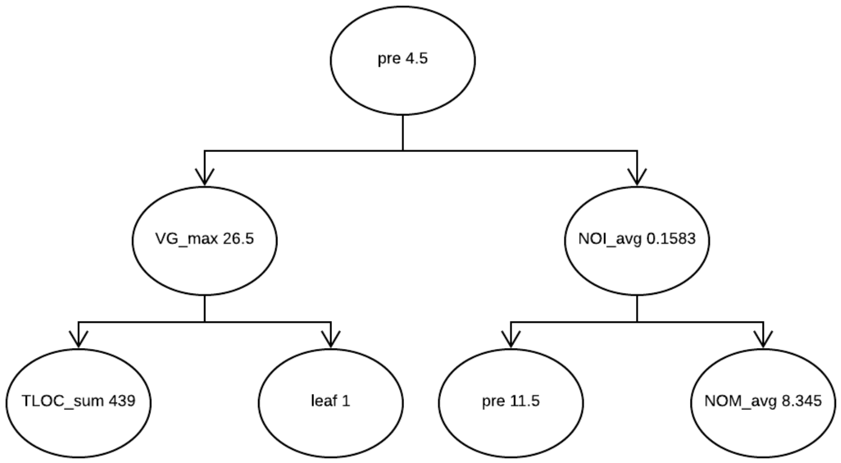

Figure 1: First splits

it looks at the number of linearly independent paths (McCabe, 1976) and gives a complexity score based on that. The more complex a program gets, seems to be a good indication of how hard it would be to fix a bug, and with this assumption, a more complex program may contain more bugs. The third split is done on the number of interfaces. The last splits are on the sum of the total lines of code again on pre-release bugs and on the average number of methods. A larger average number of lines of code yields more code that must be validated and a larger program often is more complex. The average number of methods is quite similar to the total lines of code. But using more methods may also result in less nesting making code easier to interpret.

We will now explain the results obtained with a single tree. The confusion matrices can be found in table 2 results for the quality measure in table 3. The precision of a single tree of 0.7721 is found. With a perfect precision being 1. This means that 77% of the files that we predicted to have a defect did have a defect. The recall, in this case, is 0.6709, again we would like to see a value close to 1. The implication of this result is that 67% of the files that had defects were predicted to have a defect. The accuracy is 0.7504. Here 1 also means a perfect score. The implication of this accuracy is that 75% of the predictions are correct (Zimmermann et al., 2007).

Secondly, we will look at the results of Bagging which can be found in table 2 and 3. Precision is 84% (0.8366), Recall is 69% (0.6869) and accuracy is 79% (0.7882). Like we expected the results obtained by using bagging seem to have more predictive power than that of a single tree.

The final part of this paper looks at a random forest. The quality measures are scored 76% (0.7649) for precision 70% (0.6965) for recall and 76% (0.7549) for accuracy. The result of the random forest is less than that of bagging. It, however, performs slightly better than a single classification tree on accuracy

					Data				
		Single tree			Bagging			Random	forest
		0	1		0	1		0	1
Prediction	0	286	103	0	306	98	0	281	95
	1	62	210	1	42	215	1	67	218

Table 2: Confusion matrix

	Accuracy	Precision	Recall
Single tree	0.7504	0.7721	0.6709
Bagging	0.7882	0.8366	0.6869
Random Forest	0.7549	0.7649	0.6965

Table 3: Quality measuers

and recall.

References

Eclipse, F. (2018). *About the Eclipse Foundation.* Retrieved 2018-1-11, from `https://www.eclipse.org/org/`

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on software Engineering*(4), 308–320.

Zimmermann, T., Premraj, R., & Zeller, A. (2007). Predicting defects for eclipse. In *Predictor models in software engineering, 2007. promise'07: Icse workshops 2007. international workshop on* (pp. 9–9).