Data Mining INFOMDM

Assignment 1

Classification Trees, Bagging and Random Forests

Authors:

Lisa Salomons 6951864

Jur Kersten 4168763

Anastasia Nosanova 6095038

October 2019

Abstract

We examined and compared the predictive power of the single decision tree, bagging and random forest classification algorithms. These three models were used to analyse the Eclipse Bug Dataset. The goal was to predict whether or not post-release bugs have been reported. The classification strength of each model was then computed based on its accuracy, precision and recall. For significance testing, we opted for the McNemar's test for classifiers which showed significant differences between the single tree and both the random forest and bagging. However, there was no significant difference between the random forest and bagging.

Utrecht University

1 Introduction

The rapid development of information technology has allowed the collecting of large volumes of data that are needed to be further examined. Therefore, many different techniques to interpret and analyze data have appeared. One of them is to create models that could be used for prediction of objects' class based on corresponding attributes. Building classifiers is one of the most important methods used in data mining.

For this assignment, we had to build three models: single decision tree, bagging and random forest, and compare their predictive powers. The current report aims to provide a brief description of our analysis. We first give a short review of the dataset that was analysed using three algorithms, discuss the methodology, and, after that, state and interpret results.

The data used for this assignment was gathered from the Eclipse bug data set which was made by the Eclipse Foundation. This information was obtained by collecting the bug reports submitted by the users of the Eclipse open-source software (Nov). Specifically, it contains the number of errors for every package and file in the Eclipse releases 2.0, 2.1, and 3.0, that have appeared in the first 6 months before and after each release. For this assignment the package level data that consists of metrics useful for defect prediction was analysed, shown in Table 2. The release 2.0 was used as the training set, and release 3.0 as the test set. In total with the inclusion of the pre-release bugs each set contained 41 columns (Zimmermann et al., 2007).

2 Methodology

All three algorithms use these parameters: number of observations that a node must have to be allowed to be split (*nmin*), the minimum number of observations necessary for a leaf node (*minleaf*), and the number of features that should be considered for each split (*nfeat*). We begin with single classification tree training with $nmin = 15$, $minleaf = 15$, $nfeat = 41$. Then we create the bagging model that has the same parameters as the single tree with the additional $m = 100$, that indicates the number of bootstrap samples to be drawn. Lastly, we build a random forest with the same parameter setting as for bagging, except for $nfeat = 6$.

To examine the quality of each classifier we have to determine whether or not the presence of post-release bug was classified correctly by our models. Thus, to "measure" the strength of classifiers we have to compute the accuracy, precision and recall on each test set, and present confusion matrices, see Table 2. We believe that bagging and random forest will show higher quality scores compared to the single tree. Moreover, the bagging algorithm should have better predictive power than a random forest. A simple explanation for both methods outperforming the single tree is that this algorithm is a lot simpler. Since it only forms one tree and its accuracy is heavily dependent on the splits it makes. Any imperfections will be punished more severely than in the other methods. Whereas Bagging and random forest provides 100 trees and makes a classification based on all these trees. Simply put, more trees should provide higher accuracy, precision and recall. As to the prediction of bagging outperforming that of random forest, there is only a small difference between the two methods. They are both trained on a bootstrapped sample, but where bagging is trained on all attributes the random forest is trained on a specified (fewer) number of attributes which are randomly sampled. Therefore, the latter model is more computationally effective. Moreover, it adds more randomness and is, thus, less prone to overfitting to the training set. However, this randomness can also cause the random forest algorithm to exclude the most predictive attributes, and this may result in a lower qualification score compared to bagging.

Table 1: The attributes of the data-set.

	Abbreviation	Metric	Package level
methods	FOUT	Number of method calls (fan out)	avg, max, total
	MLOC	Method lines of code	avg, max, total
	NBD	Nested block depth	avg, max, total
	PAR	Number of parameters	avg, max, total
	VG	McCabe cyclomatic complexity	avg, max, total
classes	NOF	Number of fields	avg, max, total
	NOM	Number of methods	avg, max, total
	NSF	Number of static fields	avg, max, total
	NSM	Number of static methods	avg, max, total
files	ACD	Number of anonymous type declarations	avg, max, total
	NOI	Number of interfaces	avg, max, total
	NOT	Number of classes	avg, max, total
	TLOC	Total lines of code	avg, max, total
packages	NOCU	Number of files (compilation units)	value

Table 2: Performance metrics used in the analysis.

Performance metric	Definition	Calculation
Accuracy	Ratio of correct classifications to the total number of classifications	(TP + TN) / (TP + TN + FP + FN)*
Recall	Ratio of predicted and observed as defect-prone to the number of packages that actually had bugs	TP / (TP + FN)*
Precision	Ratio of predicted and observed as defect-prone to the number of packages that were predicted as defect-prone	TP / (TP + FP)*

* TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative.

3 Results

3.1 Single Classification Tree

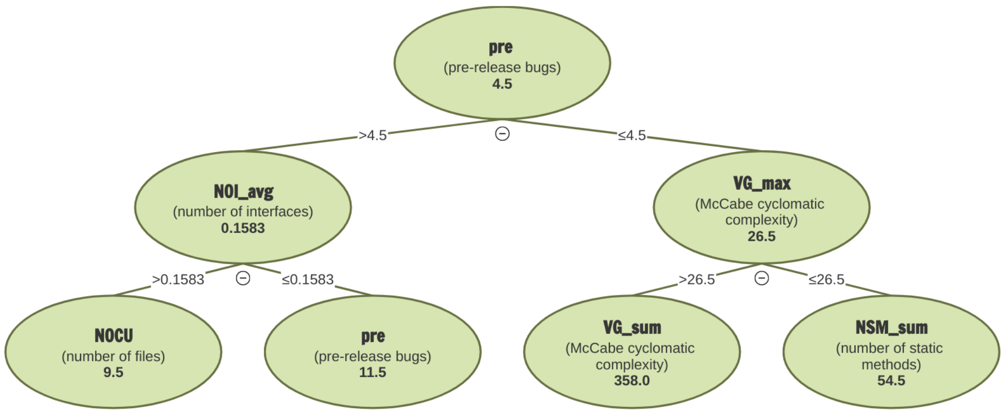

Figure 1: First split for the single classification tree.

To find out whether or not the single tree classification rule makes sense we inspect its' few splits and analyze the matrices on which these splits are performed. The image shown in Figure 1 represents the first splits of the single decision tree. The initial split is performed on the pre-release bugs that were observed during development and testing of a program. This is logical because usually the package goes into the release with some defects being unfixed. Therefore, packages that have the most pre-release bugs are likely to also have a postive value for post-release bugs. This helps in splitting a big group of data to their correct class value and thus reducing the remaining data's impurity. The next split appears on *VGmax*, the maximum McCabe cyclomatic complexity. This metric measures linearly independent paths and returns the complexity of a program (Mccabe, 1976). As a result, the more complex the code gets, the harder it becomes to fix a bug, thus the more complex program tends to have more defects compared to a simpler one. Then our tree splits on McCabe complexity again and on number of static methods. Moreover, this also holds for the splits that are depicted on the left of Figure 1: the bigger and more complex a code gets the more likely it'll become that the package contains bugs. Therefore the classification rule of the single tree algorithm is coherent.

3.2 Single tree, Random Forest & Bagging

The prediction strength of each model is analysed in the following subsection. The quality measures are reported in Table 3 and the confusion matrices can be found in Table 4.

First we start with the single classification tree. It's accuracy is 0.6853, while the perfect value would be 1. Therefore, 69% of the predictions made by a single tree are correct. The precision was found to be 0.6996, and in the ideal case it should be equal to 1. This means that 70% of packages that were predicted to have bugs actually had bugs. Finally, the recall for the single tree is 0.5879, with a best value being close to one. Thus, 59% of packages that had defects were predicted to have defects.

Next, we evaluate the quality measures values for random forest. This algorithm has accuracy equal to 0.7534 (75%). Its precision is 0.7717 (77%) and recall is 0.6805 (68%). Therefore, the performance of a random forest is better than of a single classification tree.

Finally, the bagging has accuracy of 0.7730 (77%), precision 0.8327 (83%) and recall 0.6518 (65%). As we can see, the accuracy of bagging is slightly higher compared to random forest.

Table 3: Quality measures for the three different models.

	Accuracy	Precision	Recall
Single Tree	0.6853	0.6996	0.5879
Random Forest	0.7534	0.7717	0.6805
Bagging	0.7730	0.8327	0.6518

Table 4: Confusion matrices for Single Tree, Random Forest and Bagging.

		Single Tree		Random Forest		Bagging	
		Predicted		Predicted		Predicted	
		0	1	0	1	0	1
Actual	0	269	79	285	63	307	41
Actual	1	129	184	100	213	109	204

3.3 Statistical Analysis

To compare the three different methods we decided to run McNemar's test for classifiers. With this test it is reasonably straightforward to analyse the difference between two classifiers on the basis of relative correctness and relative error. In other words, it calculates the count of test instances that the first model got correct and the second model got incorrect, and vice versa, then compares these and checks whether this difference is statistically significant. This test was chosen since it uses a simple principle utilising a 2x2 matrix of count comparisons between two classifiers and because it is widely used and deemed an appropriate measure for comparison on the basis of performance (May and Johnson, 1997). For our analysis a p-value below 0.05 would mean a rejection of the null-hypothesis (there is no significant difference in performance between classifiers).

Using the McNemar's test for classifiers, a difference in performance was found between the single tree and the random forest (McNemar's chi-squared = 16.33, df = 1, $p < 0.0001$) as well as between the single tree and bagging (McNemar's chi-squared = 31.77, df = 1, $p < 0.0001$) The difference between random forest and bagging however was deemed not significant (McNemar's chi-squared = 2.55, df = 1, p = 0.11). The p-value of the first two tests are smaller than the critical value of 0.05, therefore the performance of the single classification tree is significantly different from the random forest and the bagging algorithm. Since this number is very close to zero we can conclude that in both cases the single classification is outperformed by the other two algorithms, thus the difference in accuracy is significant. However, the p-value of the test between random forest and bagging has a value of 0.11, therefore the random forest does not outperform the bagging in a significant matter, and vice versa. To conclude, there is no significant difference in accuracy between the models since they statistically do not outperform one another.

Even though random forest only uses six attributes, while bagging operates on 41, the results showed no meaningful difference between these two classifiers. A conclusion drawn from this could be that it is possible to grow most of the trees on just 6 attributes and make satisfying splits without the accuracy decline. This pleads for the less computationally heavy random forest classifier to be sufficient for this task. Another possible reason for these classifiers not performing statistically different could be overfitting. Meaning that bagging has a higher chance to overfit the training data which may result in the lower accuracy on the test data. If we were to optimise the algorithms this problem could be tackled by using pruning.

References

May, W. L. and Johnson, W. D. (1997). The validity and power of tests for equality of two correlated proportions. *Statistics in Medicine*, 16(10):1081–1096.

Mccabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320.

Nov, M. M. About the eclipse foundation.

Zimmermann, T., Premraj, R., and Zeller, A. (2007). Predicting defects for eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, PROMISE '07, pages 9–, Washington, DC, USA. IEEE Computer Society.