Project 1

Dal Col Giada 0093122
Keller Dirk 4282264

14.10.2022

1 Description of the data

This project focuses on predicting which components of a software architecture in Eclipse [3] are more failure-prone than others. More specifically, the future releases version of each individual component in the software architecture will be classified to be bug susceptible or not, based on the occurrence of bugs in a previous version and other important features of the software architecture. The occurrence of bugs in the pre released version is binary feature which records if there is at least one pre-release defects: specifically, it takes value 1 if there is at least one pre-released bug, 0 otherwise. For the characteristics of the software architecture two empirical data sets from Eclipse are used that describe a set of certain metrics with the purpose of tracking defects in packages and classes in pre- and post-released version of this programming environment. The data set contain 40 variables - metrics describing the components of Eclipse; some are related to methods, others to classes, files and packages and for each of them (except one, namely NOCU), the metrics are qunatified at file or package level as either a single value or th average, total and maximum value. All the variables are complexity metrics for example, some features include the "Number of method calls", "Number of static methods " or the "Total lines of code". For an overview of the variables see tabel 1. The post-released errors are included as the ground truth labels for classification problem. Since he post release errors are coded are coded numerically, they are transformed to a binomial representation, where they are coded as 1 when one or multiple bug are present, 0 when no defects are detected.

Importantly, in order to predict the future, the eclipse data set data set 2.0 ($n = 378$) is used as a training set and the eclipse data set data set 3.0 ($n = 661$) will be used as test set, resulting a training-test split of 1:2. Only the pre-, post-release and the software architecture characteristics in table 1 are used in the analysis; all other features are deleted from the data set.

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

Table 1: Metrics in the Eclipse data set.

2 First splits

In Figure 1 we can see an over-simplified decision tree, pruned after the first split on the left child node and after the second split in the right child node; the distribution of the class label is saved in each node and the split rules on the edges.
The feature "pre" yields the best impurity reduction of the root node with a split value of 4.5: this variable detects if a bug is present in the pre-released version. Errors in this early version are an important predictor of the errors occurring in the post-released version: a package with an error rate higher than 4.5 errors in the pre-released version is the predictor for identifying an error-prone post-released version of the package. Probably these packages constitute an overall ill-designed architecture that might be inflexible. As a consequence, the classification obtained assigning to the majority class in the left child node L1 is very reasonable.
On the right child node (R1) the feature that reduces the impurity the most is "VG_max" which stands for the maximal value of the McCabe's cyclomatic complexity. The cyclomatic number of a graph with n vertices, e edges and p connected components is $n - e + p$ [1]; associating a program with its control-

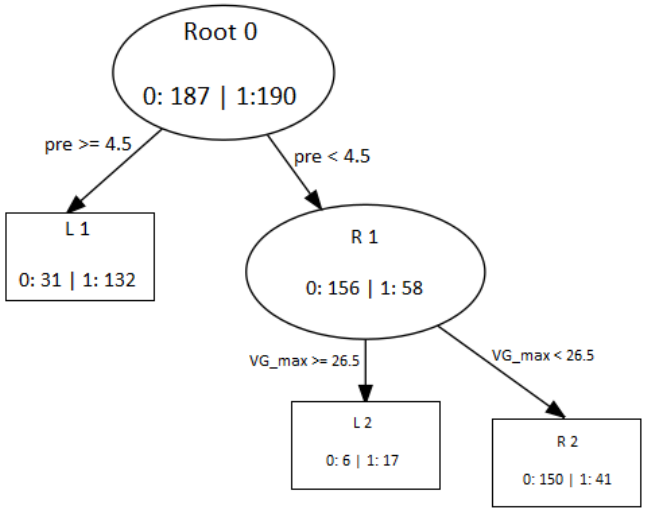

Figure 1: A picture of the first two splits of the single tree

flow graph, it's possible to use this definition to compute the complexity measure stated before. The McCabe's cyclomatic complexity measures the number of linearly independent paths through a program's source code. A set of paths is said to be *linearly dependent* if there is a subset of paths where their edge sets have empty symmetric difference, which means there are not any non-common edges. For instance, if the source code contains an IF statement, then there are two possible paths to follow, the one when the condition evaluates TRUE and the one when evaluates FALSE: paths have non-common edges since the condition can not be at the same time true and false, therefore they are linearly independent and the complexity results to 2.
The cyclomatic complexity is demonstrated to be positively correlated with the frequency of defects occurring in a function or method. Thus, this metric is also used to identify risk of defects in a software program and, consequently, functions and methods that have the highest complexity are more likely to have defects.
With respect to the tree construction, if a program has less than 4.5 errors in the pre-released version and a complexity score of less than 26.5, the prediction according to the majority class will be of a non-defective program (node R2), while with a complexity score higher than 26.5 the majority class is the one of defective packages (node L2). The split value agrees with McCabe's interpretation of the correlation between cyclomatic complexity and risk of defects, as shown in Table 2: in fact the best split value 26.5 is near to the lower boundary identifying a complex model, thus more defect-prone.

Procedure	Complexity score	Risk level
Simple procedure	1 - 10	little risk
More complex	11 - 20	moderate risk
Complex	21 - 50	high risk
Untestable	> 50	very high risk

Table 2: McCabe's interpretation of the cyclomatic score as a measure to identify risk of defects.

3 Confusion matrices and quality measures

We have considered three different classifications with the same test set. Firstly, we used a single tree with growing constraints on the leaves, specifically the minimum number of elements needed for a node to be a leaf ($minleaf = 5$) and the minimum number of elements needed for a node to be allowed to split ($nmin = 15$). Secondly, we classified using bagging, taking as prediction for a new instance the majority of the predictions of all the trees built; the same constraints on the leaves are used for each single tree and $m = 100$ bootstrap samples are drawn. Lastly, random forest was used, each time selecting 6 features randomly to perform the best split. In Table 3 the confusion matrices obtained in the three classifications.

		Tree		**Bagging**		**Random Forest**	
		Predicted		**Predicted**		**Predicted**	
		0	1	0	1	0	1
True	0	245	103	307	41	273	75
	1	99	214	102	211	93	220

Table 3: Confusion matrices of the three classification methods.

From this first analysis we see an improvement in the classification as the model increases in complexity: single trees are not the best predictors, but averaging the results from multiple tree classifications reduces the error, resulting in a more accurate result; random forest does not improve over bagging alone: this result is surprising since selecting randomly the features for the best split is expected to reduce the correlation between trees, and thus improve the performance.
To compare the models more precisely, we observed the following quality measures: accuracy, precision and recall. The first relates the number of correct

classifications (true positives and true negatives) to the total number of elements, the second relates the number of true positives to the number of elements predicted as defect-prone, while the last one relates the number of true positives to the number of elements that actually had defects. Ideally, we would like to have all this quantity as close as possible to one, which would result in a perfect classification. The obtained values are listed in Table 4.

	Tree	**Bagging**	**Random Forest**
Accuracy	0.694	0.784	0.746
Precision	0.675	0.837	0.746
Recall	0.684	0.674	0.703

Table 4: Quality metrics for the single tree, tree with bagging and random forest

There is a clear improvements in all three quality measures between the simple tree and the other two models, since averaging the majority vote of multiple classifications reduces the variance of the model and thus the overall error. Comparing bagging and random forest instead, we notice a decrease in accuracy and precision in the latter: random forest is selecting, for each split, a random subset of all the features which could result in a search of the best split on features that are ill suited. The selected *nfeat* features could be the ones that give the worst impurity reduction, thus the best split selected among them does not improve the classification.

4 Differences in accuracy

Comparing the models according to their accuracy gives bagging as the predictor that makes less mistakes; but in order to argue if the differences in accuracies between models are significant we need to perform a statistical test, specifically McNemar's test [2]. The null hypothesis of this test states that the proportion of error in the test set is the same in the two classifications, which basically means that the two models disagree in the same way. If the null hypothesis can not be refused, then as a direct consequence we can argue that the difference in the accuracy is not statistically significant; in fact, having the same proportion of error means having the same proportion of correct predictions (accuracy).

More specifically, the test is based on the *contingency table*, a tabulation displaying the frequencies of the combinations of correct/incorrect predictions in the two models, so essentially counting how many times the two predictions agree and disagree. In Table 5 the three contingency tables for all the comparisons, where we indicated as T_i (resp. F_i) the number of correct (resp. incorrect) predictions made by model i; for example the first entry of Table 5

	T_2	F_2
T_1	492	32
F_1	91	111

(a) Tree vs. Bagging

	T_3	F_3
T_1	416	43
F_1	77	125

(b) Tree vs. Forest

	T_4	F_5
T_2	470	48
F_2	23	120

(c) Bagging vs. Forest

Table 5: Contingency tables

(a), indicated as T_1/T_2, says that tree and bagging both predicted correctly 492 instances, while the second entry, indicated by T_1/F_2, says that 32 elements were correctly classified by the simple tree but classified wrongly by bagging.

From this contingency table the true value and the predicted value of the each two sets of machine learning classifier i and j are compared against each other. The test statistic is S and is computed as follow:

$$S = \frac{(T_i/F_j - F_i/T_j)^2}{T_i/F_j + F_i/T_j}$$

The distribution of S can be approximated with a Chi-Squared (χ^2) distribution with 1 degree of freedom and its p-values of the three tests are summarized in table 6. The results of the Chi-Squared test for the single tree against bagging $\chi^2(1) = 32$, $p < 0.001$, single tree against random forest $\chi^2(1) = 43$, $p < 0.005$ and bagging against random forest $\chi^2(1) = 23$, $p < 0.005$, show that for χ^2. Hence, with a significance level of $\alpha = 0.5\%$, we can refuse the null hypothesis in all the three tests, implying that each model differs significantly from the other in terms of accuracy significantly. Surprisingly it seems that Random Forest does not improve over the Decision Tree with bootstrap aggregation (but vice versa). Although, the random forest walk allows to decorrelate the trees, which is expected to improve the generalization performance on the test set, a small feature set size (e.g. here $\sqrt[2]{\lceil 41 \rceil}$)), when drawing randomly from the uniform feature distribution, might not allow to the best features to be in the feature set for the corresponding tree. In fact, the liklihood that only bad features are included to build the tree are quite high.

Test	Tree vs. Bagging	Tree vs. Random Forest	Bagging vs. Random Forest
p-value	$9.739e^{-8}$	$2.447 * 10^{-3}$	$4.065 * 10^{-3}$

Table 6: P-values of the McNemar's test for comparing the difference in accuracy.

References

[1] Thomas J McCabe. "A complexity measure". In: *IEEE Transactions on software Engineering* 4 (1976), pp. 308–320.

[2] Quinn McNemar. "Note on the sampling error of the difference between correlated proportions or percentages". In: *Psychometrika* 12.2 (June 1947), pp. 153–157. DOI: 10.1007/BF02295996. URL: https://ideas.repec.org/a/spr/psycho/v12y1947i2p153-157.html.

[3] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. "Predicting Defects for Eclipse". In: *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*. 2007, pp. 9–9. DOI: 10.1109/PROMISE.2007.10.