Data Mining Assignment 1

Daniele Di Grandi
7035616

Jens Hartkamp
6021271

Alice Hartog
7035608

Classification Trees: Single Tree, Bagging and Random Forest

1 Introduction

In this paper we will analyze the results obtained by three prediction models - Single Tree, Bagging and Random Forest - whose input comes from the bug database of an open-source project, Eclipse, based on the article from Zimmermann et al. [1]. Using these data, we are trying to predict whether or not a bug in the program will happen. A detailed description of this database is provided in Section 1.1.

In fact, the global objective of these models is, after being trained on some data, to be used to predict the value of a target variable. In our case, we will use a binary-type Classification Tree, since, as we will see, our target variable can only assume 2 possible values.

The obtained results are reported, reflected on and discussed. Then, a comparison between the three prediction models will be performed, to understand if there are statistically significant differences between them.

Finally, a trend analysis on how the accuracies of the models are affected by some threshold parameters (namely, the *nmin* and *minleaf* parameters) will be performed.

The code of our program can be found at reference [2].

1.1 Data Description

The authors' aim was to map defects happening in Eclipse, thus the database is a list of packages that could have a bug, observed both during the development and after its release, considering releases 2.0, 2.1 and 3.0. In this analysis only two releases are used: package 2.0 as training set and package 3.0 as test set. In the rows of the database we can find the name of the packages. In the columns we have a number of descriptors. Just to mention a few interesting ones, we have the type of pre-release defects (the ones reported in the last six months before release), type of post-release defects (reported in the first six months after release) and other complexity metrics, which have been computed for each case (row). Examples of these complexity metrics can be aggregators such as average (avg), maximum (max) and accumulation (sum) regarding some aspects of the specific row. All these columns, except for the post-release defects, which is the label to predict, are considered by our models to be features. However, from a total of 210 features, only 41 were chosen by the paper's authors for the model - listed in Table 1 of [1] - and we will do the same in our paper.

As previously said, the main interest is to actually detect and predict whether a case will be defect, thus the post-release situation (column 'post'). In order to do this it is sufficient to classify the packages as defect-prone or defect-free, which translates into defect = 1 and defect = 0, respectively.

By an exploratory analysis of the two databases, it is possible to draw some basic statistics. The training set has a total of 377 packages: 187 cases labeled as 0 (49.6%) and 190 cases as 1 (50.4%). The test set has a total of 661 packages, of which 348 were labeled as 0 (52.65%) and 313 as 1 (47.35%).

If we were to already use a very simple and trivial model to predict the classes, that is always predicting the majority class, we can obtain an initial classification. In the training set the majority class is 1, hence, if we use the training data to understand which is the majority class, we understand that we will always predict 1, and an accuracy of 50.4% on the training set and 47.35% on the test set are obtained. Thus, the accuracy of this trivial model is to be considered 47.35%. Hence, we have a base case to compare with the three more complicated models we will build in this paper.

2 Data Analysis

When it comes to use Classification Trees, several choices should be made. First, in order to compute the impurity of a node, and consequently the best split in that node, the gini-index has been chosen as impurity measure. Second, to prevent overfitting and reduce the computational time, we have included some stopping-rules, by including 3 pa-

rameters: *nmin*, *minleaf* and *nfeat*, for each of the three tested models.

The *nmin* parameter indicates the number of observations that a node must contain at least, for it to be allowed to be split. Hence, if a node contains fewer cases than *nmin*, it becomes a leaf node. In our models we set this parameter to $nmin = 15$.

The *minleaf* parameter is the minimum number of observations required for a leaf node. Hence, a split that creates a node with fewer than *minleaf* observations is not acceptable. In our models we set this parameter to $minleaf = 5$.

The *nfeat* parameter denotes the number of features that should be considered for each split. While for the Single Tree and Bagging models, this value is set to be equal to the total number of available features - 41 in our case - for the Random Forest model it is different. In fact, this is exactly where Random Forest differs from Bagging: in order to de-correlate as much as possible the trees of which the Random Forest is composed of, in this model the *nfeat* parameter is set to be $\sqrt{nfeat}$ rounded to the nearest integer, thus, $round(\sqrt{41}) = 6$. Hence, in this model, each time a split has to be made on a node, 6 different features are chosen among the 41 available, and the best split has to be computed only on these 6 features.

Finally, the m parameter was introduced only for Bagging and Random Forest, and defines how many Single Trees these models must be composed of. In these models, we set this parameter to $m = 100$.

Quality measures. Single Trees, Bagging and Random Forests were all evaluated and compared using five performance indicators: accuracy, precision, recall, f1-score and support. Each model will also generate a *confusion matrix*. In this matrix, the columns represent the model predictions and the rows represent the actual classes that should have been predicted. Thus, the main diagonal tells us the number of correct predictions within each class.

2.1 Single Tree

The first model we are going to discuss is the Single Tree model: a single classification tree that is built and used to predict the value of the target variable. On the Eclipse data, the classification tree resulted in a maximum depth of 10 levels, considering the root node as level 0, or 11 if the root node is considered to be level 1. It has 29 intermediate nodes and 30 leaf nodes. A representation of the first three levels - if the root node is considered to be level 1 - is showed in Figure 1.

For the sake of the representation, the nodes in the third level are to be considered as leaf nodes, as if a pruning operation would have happened to the rest of the tree, although this is not happening in the program we developed. Then, in these new leaf nodes, we predict the majority class.

However, the resulting representation in Figure 1 is not completely appropriate, although it is not that inaccurate as one would expect.

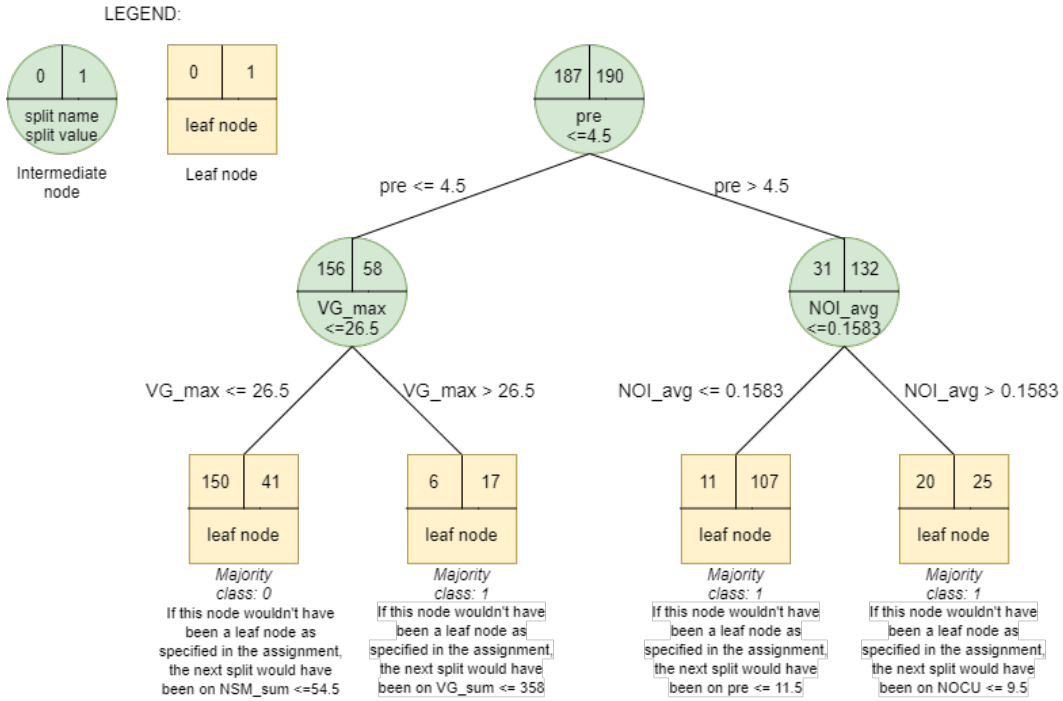

Figure 1: *Classification tree showing the first three levels.*

The main reason is that three attributes can not be a good representation of the model, considering there are 41 attributes used, or at least available. However, a credit has to be given: this over simplified model is actually not as bad as expected. The reason has probably to be searched in the 'pre' feature, which actually is doing a good job in splitting the tree. In a sense, this is fair: as explained in Section 1.1 the column 'pre' are the number of defects reported in the last six months before release. We are trying to predict whether or not the application will be defect-pruned. The logic is clear: our target variable is exactly of the same type as the 'pre' predictor, hence, we are basically using historical data to make predictions. Just as an example, it is like predicting whether or not a type of car would break, provided that we know how many times the same type broke before: this certainly is a more straight-forward indicator rather than using, for example, the average number of kilometers that the considered type of car traveled before breaking down.

Continuing with the complete model, using the test set, we obtain the following results:

		Predicted value		
		0	**1**	**Total**
Actual value	**0**	266	82	348
	1	125	188	313
	Total	391	270	661

Table 1: *Confusion matrix of a Single Tree.*

	0	**1**
Precision	0.68	0.7
Recall	0.76	0.6
f1-score	0.72	0.64
Support	348	313

Table 2: *Precision, recall, f1-score and support for a Single Tree.*

The output shows an accuracy of 68.68%, thus higher than the initial one (47.35%, as explained in Section 1.1) by 21.33%. This shows an already outstanding improvement, proving how the Single Tree model, which is a more complex mathematical structure compared to the simple 'predict the majority class' model, is capable of achieving better results.

2.2 Bagging

When analyzing this model, we have to introduce *bootstrapping*: a procedure that, given the high variance of classification trees as classifiers, guarantees its reduction.

This happens because a new database is created from the original one, with the same size of rows and columns, where rows are instead randomly selected from the original database either zero, one or multiple times. This guarantees that the new database contains different data, but with the same distribution, from the original one and every created tree is different from any other.

Due to the randomness factor included by this procedure, the output of this model changes each time a run is performed. Hence, to overcome this problem and perform a fair comparison, we decided to run 20 times this model and construct a more precise confusion matrix by averaging the results of every confusion matrix obtained in each run, keeping in mind that the total number of classified cases should remain the same. Then, from this new confusion matrix, we can calculate the new quality metrics. Using the test set, we obtain the following results:

		Predicted value		
		0	**1**	**Total**
Actual value	**0**	305	43	348
	1	106	207	313
	Total	411	250	661

Table 3: *Confusion matrix of Bagging.*

	0	**1**
Precision	0.74	0.83
Recall	0.88	0.66
f1-score	0.81	0.74
Support	348	313

Table 4: *Precision, recall, f1-score and support for Bagging.*

The model accuracy is around 77.8%, 30.45% higher than the initial one (47.35%) and, as we will see, the highest of all three models. This result is impressing if compared to the Single Tree model. In fact, only by averaging the predictions of 100 trees and using a sampling with bootstrapping procedure, we obtain an accuracy increase of 9.12%. By comparing the results of the Single Tree model and the results from Bagging we can see that the major increase in precision was obtained by correctly classifying more cases as 1, among all the cases that the algorithm classifies as 1. In fact, for this class, we have obtained an increase of 13% (from 70% to 83%), while for the class 0 only an increase of 6% (from 68% to 74%). On the other hand, among the actual cases that the algorithm should have predict as correct, the recall tells us that this time the class 0 has improved: an increase of 12% was achieved (from 76% to 88%) compared to the increase of 6% (from 60% to 66%) of the class 1.

2.3 Random Forest

As already mentioned, in the Random Forest model, each time we make a split, 6 features are randomly selected to decide the best split. However, this process causes another random factor that is included in the model, other than the bootstrapping procedure. Hence, as in the Bagging model, we have collected the results of 20 run, and obtained a more precise confusion matrix, from which we have calculated the quality metrics.

Using the test set, we obtain the following results:

		Predicted value		
		0	**1**	**Total**
Actual value	**0**	287	61	348
	1	98	215	313
	Total	385	276	661

Table 5: *Confusion matrix of Random Forest.*

	0	**1**
Precision	0.75	0.78
Recall	0.82	0.69
f1-score	0.78	0.73
Support	348	313

Table 6: *Precision, recall, f1-score and support for Random Forest.*

The model accuracy is 76%, higher than the initial accuracy (47.35%) by 28.65% and than the Single Tree one by 7.32%. However, the accuracy obtained here is 1.8% lower than the one obtained with the Bagging model. This result is a bit strange since, normally, the reduced number of features used by the Random Forest model should de-correlate the data, obtaining an improvement from the accuracy point of view on unseen data. However, we should keep in mind that this process is indeed a result of a trade-off analysis, since that it is correct that we have less correlated data, but we also have less features to predict the label value. What it is likely that happened in this particular case, is that the features were not that correlated to justify a reduction in the features used to predict the label, causing a drastic reduction of the search space for the best possible split. Furthermore, as we previously discussed, it is possible that some features are way better than others to predict the label, and if these features are not selected in an iteration of the algorithm, a worse split is instead performed.

2.4 Statistical test between models

In order to establish if there are statistically significant differences between the three models, a McNemar test has been performed [3]. Among all possible statistical tests, the McNemar one has been chosen because it determines if two related groups have significantly different categories proportions.

In this specific test, the H_0 hypothesis states that the two models are similar, hence they do not give statistically significant differences in accuracy, while H_1 states that the accuracies are different. Rejecting H_0 means that the models have statistically significant differences in accuracy, thus H_1 is true. On the other hand, not rejecting H_0 means there is no statistical evidence to state that the models are different, but neither that they are equal.

By setting the probability of type one errors $\alpha = 5\%$, we obtain Table 7 which shows χ^2 and p-value for each model comparison.

	χ^2	**p-value**
Single Tree vs Bagging	29.27	0
Single Tree vs Random Forest	19.72	0.000009
Bagging vs Random Forest	2.36	0.124

Table 7: *Values for χ^2 and p-value obtained in output from the McNemar statistics test.*

If the p-value of a test is less than α, we can reject H_0 and conclude that there is statistical significance between the two accuracies, i.e. the difference between false negatives and false positives is too important and thus not random.

From Table 7, we can conclude with a strong statistical evidence that a Single Tree is worse than the Bagging and Random Forest models in predicting the true label. On the other hand, we have a p-value of 12.4% from the comparison between Bagging and Random Forest, hence, it is not enough to have statistical significance to conclude that Bagging and Random Forest are different.

Moreover, notice that a low p-value in this test means a high value for χ^2. During the test performance we implemented a Yates continuity correction because χ^2 is a continuous distribution while our sampled distribution is discrete [4]. Moreover, for low frequencies, the approximation could be very small.

3 Varying the hyperparameters

In order to gain further insight in the *minleaf* and *nmin* parameter settings, we build the three models while setting the *minleaf* to 0 with an increasing *nmin* and vice versa. The scope of this analysis is indeed to understand how the accuracy of each model changes while one parameter is changing, hence, a fair analysis should not be biased by the other parameter, and this is the reason why we set it to be 0, while the parameter of interest is changing. Please note that we are not trying to perform a parameter tuning for this database through a cross-validation operation, we are only trying to understand how the accuracy is affected by the *nmin* and *minleaf* parameters.

For the *minleaf* parameter we decided to test for the values 1 until 15. The *nmin* setting was set to 3 until 36 with increments of 3. These values were arbitrarily chosen based on the settings given by the assignment. Furthermore, in case of the Random Forest and Bagging, the models were built 5 times and the accuracy was averaged over these runs in an attempt to reduce the fluctuations given by the random component. The models were then evaluated and the accuracy is shown in Tables 8 and 9. We expected to see an output similar to a bell curve where the accuracy should first be low due to overfitting on the training set, after which it would gradually increase until the maximal obtainable accuracy is reached. Once this point is passed we expected to see the accuracy drop back approaching the 47.35% accuracy found by predicting the majority class. This drop in accuracy was expected due to underfitting forced by the high *minleaf* or *nmin* parameter respectively.

***minleaf* Value**	**ST**	**RF**	**BAG**
1	0.70	0.76	0.78
2	0.69	0.76	0.76
3	0.69	0.75	0.78
4	0.71	0.76	0.78
5	0.69	0.76	0.78
6	0.72	0.76	0.78
7	0.74	0.77	0.78
8	0.68	0.76	0.78
9	0.70	0.77	0.78
10	0.74	0.76	0.78
11	0.74	0.76	0.79
12	0.75	0.76	0.79
13	0.75	0.76	0.79
14	0.75	0.76	0.78
15	0.76	0.76	0.79

Table 8: *Accuracy for the Single Tree(ST), Random Forest(RF) and Bagging(BAG) models with varying minleaf settings*

***nmin* Value**	**ST**	**RF**	**BAG**
3	0.70	0.76	0.77
6	0.71	0.76	0.77
9	0.71	0.76	0.78
12	0.69	0.75	0.77
15	0.69	0.77	0.76
18	0.72	0.76	0.77
21	0.71	0.76	0.77
24	0.72	0.76	0.77
27	0.72	0.76	0.76
30	0.72	0.77	0.77
33	0.74	0.77	0.77
36	0.74	0.76	0.76

Table 9: *Accuracy for the Single Tree(ST), Random Forest(RF) and Bagging(BAG) models with varying nmin settings*

When considering the results shown in Table 8 we see there is very little fluctuation in accuracy across the board. When adjusting the *minleaf* parameter the accuracy barely changed. This implies either of two things: the parameter settings we chose to test *minleaf*'s impact were too low and a clearer effect will show once the *minleaf* parameter is set higher, or there are a number of important attributes which allow a big number of nodes to be correctly labeled right away, resulting in very little impact for smaller leaf nodes.

The *nmin* setting shows a similar result. There is very little fluctuation in accuracy leaving two main hypotheses similar to those stated for the *minleaf* parameter. Either the *nmin* settings we chose were too low to make a difference, or there are a small number of attributes which classify a big majority of the nodes after a small number of splits. To further investigate we decided to do a few additional tests with higher increments for both *nmin* and *minleaf*. These results are shown in Tables 10 and 11.

From both these results and some further investigation in the dataset we conclude that the reason of these results is that there is one attribute called 'pre' which has a huge impact on the accuracy of the prediction models. Indeed, we already discussed in Section 2.1 a hint in this direction: the simplified Tree shown in Figure 1 was not that bad despite using only three attributes, among which, 'pre'. Hence, the same reason why this attribute is so good - previously discussed in Section 2.1 - stands even here.

Therefore, the *minleaf* and *nmin* parameters have less of an impact than initially expected. This also explains why even with a very high *nmin* or *minleaf* values, the accuracy only slightly drops instead of approaching the 47.35% accuracy given by predicting the majority class. Apart from this, we do see some small drop in accuracy indicating the parameters still have some effect. However, the ef-

fect is nearly negligible due to the impact of the 'pre' attribute.

***minleaf* Value**	**ST**	**RF**	**BAG**
30	0.79	0.74	0.80
45	0.75	0.73	0.79
60	0.76	0.73	0.81
75	0.80	0.71	0.80
90	0.80	0.70	0.80

Table 10: *Accuracy for the Single Tree(ST), Random Forest(RF) and Bagging(BAG) models with varying minleaf settings*

***nmin* Value**	**ST**	**RF**	**BAG**
50	0.75	0.77	0.78
65	0.77	0.78	0.78
80	0.77	0.77	0.77
95	0.78	0.74	0.78
110	0.78	0.73	0.78

Table 11: *Accuracy for the Single Tree(ST), Random Forest(RF) and Bagging(BAG) models with varying nmin settings*

4 Conclusion

Zimmermann et al. [1] obtained an accuracy of 72.9% using a Logistic Regression model in the analysis of packages, a value in between the accuracy of our Single Tree model and Random Forest. Hence, in the Eclipse database, the Logistic Regression model classifies itself as a less accurate model than Random Forest and Bagging, although a McNemar test should be performed in order to have statistical evidence.

In this paper we have performed an analysis on the results obtained by three models - Single Tree, Bagging and Random Forest - tailored on the Eclipse dataset, where we have tried to predict whether a package of that application will be defect-pruned or not.

Overall, our models' accuracies resulted to be quite high especially for Random Forest and Bagging, obtaining an average values of 76% and 77.8%, respectively. Slightly higher accuracies could have been obtained using a better parameter settings, but as shown in Section 3 these improvements would have been negligible.

References

[1] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*, pp. 9–9, IEEE, 2007.

[2] D. Di Grandi, J. Hartkamp, and A. Hartog, "Python code for the first assignment of the data mining course." `https://github.com/danieledigrandi/DataMining_2021/tree/main/Assignment_1`, 2021.

[3] P. A. Lachenbruch, "Mcnemar test," *Wiley StatsRef: Statistics Reference Online*, 2014.

[4] C. Stefanescu, V. W. Berger, and S. Hershberger, "Yates's continuity correction," *Encyclopaedia of Statistics in Behavioural Science*, vol. 4, pp. 2127–2129, 2005.