Data Mining: Assignment 1

CLASSIFICATION TREES, BAGGING AND RANDOM FORESTS

Ana Borovac
6584446

Argyro (Iro) Sfoungari
6528015

October, 2018

Contents

Abstract

f

1 Short Description of the Data

In the assignment we are asked to use the algorithms we have implemented to analyse the bug dataset of the Eclipse programming environment, before and after the release of each packet. According to the accompanying article [**?**], in order to explain the reason why some programs are more vulnerable to failure than others, we need to take a look at a bug database. Bug databases, list all the errors that occurred during the lifetime of a software. More specifically, the dataset contains information about the number of faults that have been reported in the first six months before and after release. As reported by the authors, metrics are useful for predicting the thickness of the defects. For the assignment we are going to use the 2.0 release as the training set and the 3.0 release as the test set and we are going to monitor if errors have been reported to later versions. We are asked to use the same metrics as the authors and the number of pre-release bugs.

These metrics are:

- FOUT (Number of method calls)

- MLOC (Method lines of code)

- NBD (Nested block depth)

- PAR (Number of parameters)

- VG (McCabe cyclomatic complexity)

- NOF (Number of fields)

- NOM (Number of methods)

- NSF (Number of static fields)

- NSM (Number of static methods)

- ACD (Number of anonymous type declarations)

- NOI (Number of interfaces)

- NOT (Number of classes)

- TLOC (Total lines of code)

- NOCU (Number of files)

And Pre-released defects: the defects that are observed during the development and testing of a program.

The training set consists of 377 samples:

- 187 samples labeled 0

- 190 samples labeled 1

The test set consists of 661 samples:

- 348 samples labeled 0

- 313 samples labeled 1

2 Single Tree

Give a picture of the first few splits of the single tree. Assign to the majority class in the leaf nodes of this tree. Discuss whether the classification rule you get makes sense, given the meaning of the attributes.

Below we have reproduced the first few splits of the single tree. We continue splitting until we get the representation of the left sub-tree. We have to process the given data, and draw to some conclusions in order to decide whether our classification rule makes sense. In our tree, the initial split is based on the number of pre-released defects that are detected during the development and testing of the program. As the split begins, we see that for the better case which is the one with the fewest pre-released defects (≤ 4.5), we have a shorter cyclomatic complexity (a software metric, used to indicate the complexity of a program) `VG_max`. This split seems quite reasonable if we think that with a more complex code we predict more bugs and vice versa, with fewer pre-released defects the complexity is reduced accordingly. Then, we split into the smaller total number of static methods `NSM_sum`, fewer total number of method calls `FOUT_sum`, smaller average of nested block depth `MBD_avg`, fewer maximum number of fields `NOF_max`, smaller average number of methods `NOM_avg` and static methods `NSM_avg` respectively and finally fewer lines of code. Consequently, we could support that the classification rule we created leads to reasonable results, as less complex code, gives less methods, method calls and other programming structures and finally fewer total lines of code.

3 Single Tree, Bagging and Random Forests

3.1 Single Tree

We trained a single classification tree on the training set with `nmin = 15`, `minleaf = 5` and `nfeat = 41`. On the test set we got the following confusion matrix **??**.

True class \ Predicted class	0	1
0	266	82
1	128	185

Table 1: Confusion matrix for a single tree run on the test set.

3.2 Bagging

We used bagging with `nmin = 15`, `minleaf = 5` and `nfeat = 41` and `m = 100`. On the test set we got the following confusion matrix **??**.

True class \ Predicted class	0	1
0	308	40
1	101	212

Table 2: Confusion matrix for a bagging run on the test set.

3.3 Random Forests

We used random forests with `nmin = 15`, `minleaf = 5` and `nfeat = 6`. On the test set we got the following confusion matrix **??**.

True class \ Predicted class	0	1
0	269	79
1	117	196

Table 3: Confusion matrix for a random forests run on the test set.

4 Differences in Accuracy

	Accuracy	Precision	Recall
Single Tree	0,78	0,69	0,59
Bagging	0,92	0,84	0,68
Random Forests	0,79	0,71	0,63