Universität Innsbruck

Department of Computer Science
Research Group Quality Engineering

SEMINAR PAPER

Comparison of Defect Prediction Models

Martin Durcek, Alberte Thegler

Supervisor: Ass.-Prof. Dr. Michael Felderer

Innsbruck, June 1, 2017

Abstract: Many data mining methods have already been introduced into defect predictions. In our work, we tested and compared the results obtained from Naive Bayes and J48 decision tree based defect prediction model. For prediction itself, metrics like .... were used. We also argue about the topic of general defect prediction model and usage of the very same model within several environments.

1 Introduction

Software Fault Prediction(SFP) is one of the most active research areas in software engineering.In spite of diligent planning, documentation, and proper process adherence in software development, occurrences of defects are inevitable. Finding and fixing defects costs companies around the world huge amounts of money and therefore any automated help in reliably predicting where faults are, and focusing the efforts of testers, has a significant impact on the cost of production and maintenance of software. Various regression techniques and recently also machine learning algorithms had been utilized to provide better insight into software repositories and help developers as well as testers to invest their time at work more effectively. Although some voices are critical about the importance and rentability of creating a defect prediction models as a part of software development projects, there is no doubt such model, if implemented correctly, represents additional tool to increase work effectiveness as well as software reliability.

2 Metrics, classifier and dataset selection

In this section we describe the reasons behind choosing specific code metrics, classifiers as well as datasets to work with. Since we don't possess any deep previous knowledge on this particular topic, we mostly refer to other paper and studies - mainly the article *Systematic Review of Machine Learning Techniques for Software Fault Prediction*[2] from Ruchika Malhotra.

2.1 Metrics

There are several metric groups and metrics itself used to describe software repositories and code itself. Besides traditional McCabe and Halstead static metrics, there are also Object-oriented metrics(cohesion, coupling and inheritance count), miscellaneous metrics(change and requirement metrics, code churn) or even hybrid metrics combining various metrics from previous groups. In her extensive research, Malhotra found that in more than 50% of 64 considered papers and articles on fault prediction, traditional procedural metrics had been used. Object-oriented metrics had also been used a lot - particularly coupling between objects(CBO) and response for class(RFC) emerged as highly useful. On the other side, number of children(NOC) and depth of inheritance tree(DIT) didn't seem as good metrics for SFP purposes[2, p. 15].

2.2 Classifier

Choosing the right classifier is also not a straightforward process since there are many statistics and machine learning methods to be used and combined. During last ten years, many papers and articles have been published on the topic of classifier performance in SFP field. According to Malhotra in her paper, most widely used classifiers are decision trees and Bayesian learners followed by neural networks and support vector machines[2, p. 11].

2.3 Datasets

Several dataset options are available for SFP studies, but NASA Metrics Data Program data sets are definitely most widely used. In her research, Malhotra claims that they are used in more than 60% of 64 examined studies. KC1 is the most widely used (both module and class level) which is applied in 43% of studies and has 15.4% faulty modules and 40.6% faulty classes. Similarly, PC1 data set is used in 40% of studies and contains only 6.8% of faulty modules.[2, p. 17].

3 Experiment design

Since we want to gain from our experiment as much knowledge from SFP field as possible, we've decided to carry out rather simple tests of several decision tree and bayesian algorithms to see how they perform on NASA MDP data sets. We carried tests with the whole datasets as well as with data sets restricted just on chosen code metrics. To deal with the lack of data but still avoid overfitting, we're mostly using K-fold cross validation approach. Here is needed to mention, that in WEKA, K-fold validation is executed internally to estimate the generalization error and the output model is the model trained on the whole dataset (partial K-fold models are not shown to the user). We utilize this to also perform our own tests with the datasets where this is possible (due to their sufficient size).

3.1 Evaluation metrics

To compare models(classifiers) and evaluate their performance, there are several metrics which can be used. Of course, each metric carries different information and should be used in different situations. In our case (SFP) it can't be generalised which metrics are important, because it depends on the particular project and policy utilized within the project (should we rather pay attention to the false positives or the false negatives - in other words, do we rather want our test team to concentrate of defect-less but falsely classified class or is it worse when testing team doesn't spend enough time with the defect-rich class because it has been classified as "safe"). Since we can't favor precision over recall or vice versa, F-measure offers itself as good way to consider both of them. We have also again taken look into Malhotra's article[2] to find out that the most used ones are usually recall, precision, area under the curve(AUC - the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example[1]) and F-measure. WEKA outputs the whole confusion matrix plus several more metrics so we utilized it to compare trained classifiers with several of these.

4 Experiment results

4.1 Decision Trees

To compare the decision trees performance, we used the CM1 NASA dataset

CM1 is a spacecraft instrument used for data collection and processing written in C. We obtained from PROMISE repository and it contains 38 metrics including static code metrics like McCabe or Halstead metrics.

After cross-validating 7 different decision trees, we obtained the results which can be seen in table 0.1.

Algorithm	**Correct [%]**	**Recall**	**Precision**	**F-Measure**	**AUC**
Decision Stump	87.156	.872	.760	.812	.713
HoeffDing Tree	87.156	.872	.760	.812	.477
J48	83.792	.838	.822	.829	.622
LMT	86.8502	.869	.815	.821	.714
Random Forest	85.6269	.856	.790	.814	.770
Random Tree	79.5107	.795	.813	.803	.588
REP tree	86.8502	.869	.759	.810	.554

Table 0.1: Decision trees performance using all code metrics of CM1 dataset

At this point, we've restricted the dataset only on chosen metrics and observed the changes in classifiers' performance.

Results for chosen code metric - McCabe's cyclomatic complexity / Lines of code (LOC) can be seen in tables 0.2 and 0.3 respectively.

To check if some potential patterns found in results will be observed again we performed exactly same tests with the JM1 dataset as well. Results can be seen in table 0.4,table 0.5 and table 0.6.

Algorithm	**Correct [%]**	**Recall**	**Precision**	**F-Measure**	**AUC**
Decision Stump	87.156	.872	.760	.812	.513
HoeffDing Tree	86.8502	.869	.804	.816	.490
J48	87.156	.872	.760	.812	.480
LMT	86.8502	.869	.759	.810	.601
Random Forest	85.0153	.850	.793	.814	.471
Random Tree	84.4037	.844	.788	.811	.486
REP tree	87.156	.872	.760	.812	.477

Table 0.2: Decision trees performance using only cyclomatic complexity of CM1 dataset

Algorithm	**Correct [%]**	**Recall**	**Precision**	**F-Measure**	**AUC**
Decision Stump	87.156	.872	.760	.812	.647
HoeffDing Tree	86.5443	.865	.759	.809	.483
J48	87.156	.872	.760	.812	.477
LMT	86.5443	.865	.759	.809	.669
Random Forest	81.6514	.817	.787	.800	.571
Random Tree	78.2875	.783	.785	.784	.527
REP tree	87.156	.872	.760	.812	.477

Table 0.3: Decision trees performance using only LOC of CM1 dataset

Algorithm	**Correct [%]**	**Recall**	**Precision**	**F-Measure**	**AUC**
Decision Stump	81.6637	.817	.667	.734	.641
HoeffDing Tree	81.4761	.815	.760	.764	.565
J48	79.9333	.799	.767	.777	.649
LMT	82.0077	.820	.779	.762	.704
Random Forest	82.6644	.827	.796	.796	.760
Random Tree	76.0555	.761	.762	.761	.591
REP tree	81.4344	.814	.771	.774	.692

Table 0.4: Decision trees performance using all code metrics of JM1 dataset

As we can see from the results, all decision tree classifiers perform very similar. With smaller CM1 dataset, Decision Stump performed best with whole dataset as well as with only restricted metrics. J48 and REP joined Decision Stump in tests with restricted metrics. With much bigger JM1 dataset, Random Forest performed best in casre all metrics were available. If we restricted dataset only on cyclomatic complexity or LOC, J48 had a slight edge. But as already stated before, the differences in results are very small. That's why we decided to perform same test on Bayesian classifiers and compare the results of these 2 groups.

Algorithm	**Correct [%]**	**Recall**	**Precision**	**F-Measure**	**AUC**
Decision Stump	81.6637	.817	.667	.734	.699
HoeffDing Tree	81.6533	.817	.764	.752	.660
J48	81.9869	.820	.824	.743	.604
LMT	81.9243	.819	.804	.743	.660
Random Forest	81.7263	.817	.767	.752	.654
Random Tree	81.6324	.816	.763	.753	.647
REP tree	81.7992	.818	.775	.745	.609

Table 0.5: Decision trees performance using only cyclomatic complexity of JM1 dataset

Algorithm	**Correct [%]**	**Recall**	**Precision**	**F-Measure**	**AUC**
Decision Stump	81.6637	.817	.667	.734	.643
HoeffDing Tree	80.9027	.809	.747	.754	.698
J48	81.9347	.819	.790	.747	.655
LMT	81.8305	.818	.775	.748	.683
Random Forest	81.4865	.815	.765	.763	.681
Random Tree	81.3927	.814	.765	.766	.665
REP tree	81.6533	.817	.767	.761	.658

Table 0.6: Decision trees performance using only LOC of JM1 dataset

Bibliography

[1] Nathalie Japkowicz and Mohak Shah. *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011.

[2] Ruchika Malhotra. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, 27:504–518, 2015.