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 16, 2017

Abstract: Fault prediction models are becoming incresingly more popular by companies that wish to decrease the amount of errors in their system and the work of the developers in finding these errors. Many different prediction models have been introdced and tested in many difference scientific experiments. In our project we have tested Decision trees against Bayesian learners in order to see if there is a difference in prediction quality. We have also looked into the number of metrics used for a prediction model, and if increasing the number of metrics, also increases the performance of the prediction model. For the prediction model we used metrics like Lines of Code, Cyclomatic complexity and number of unique operands. We used the evaluation metrics % correctness, F-measure and AUC to evaluate our results. In order to learn how the models performed in general we did t-testing on the data and learned that decision trees seem to perform better than bayesian learners, however not by a big percentage.
If we create a hypothesis test for the number of metrics then this part below should be changed.We also wanted to look into the number of metrics and if the number affected the performance of the prediction model. However, the outcome of our experiment did not show any precise pattern in performance and therefore we cannot say that the number of metrics make a difference in how precise the performance model is. However is should also be mentioned that this experiment was in a small scale, and in order to complety rule out that the number of metrics have an influence on how precise the prediction model is, a number of larger experiments should be created.

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. Therefore, any automated help in reliably predicting where faults are, and thereby 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, have been utilized to provide better insight into software repositories and help developers as well as testers to invest their time at work more effectively. Some professionels voices a critical concern about the importance and rentability of creating a defect prediction models, as part of software development projects. However, there is no doubt that such a model, if implemented correctly, represents an additional tool to increase work effectiveness as well as software reliability.

2 Experiment design

2.1 Experiment description

The field of Software Fault Prediction is big, and the number of possible and interesting experiments is very large. One could for example compare groups of classifiers (decision trees vs. Bayesian learners) or particular classifiers within one group. Comparison can also be to analyse performance on different datasets or compare particular software metrics. Another option could also be just changing the parameters of one classifier in the stable environment.

To build the models, we used all decision tree as well as Bayesian network algorithms available in WEKA[1]. Reason behind our decision to use these two groups is that in most papers we worked with, decision trees and Bayesian networks have always been ranked among the top performing algorithms(e.g. [4] and [5]). Therefore the first question we addressed in our work was:

We tried to create an interested experiment which is as similar as possible to real-world problems and experiments in the field. We found the topic of number of used metrics in a fault prediction discussed in Wang's paper[6] interesting and therefore decided to also investigate in this area. We decided to try and answer the following question:

Does the number of metrics in a fault prediction model, change the performance of the predictor?

2.2 Experimental Setup

We designed our experiment so that we compare two different groups of classifiers which we choose to be decision trees and bayesian algorithms and we decided to use NASAs MDP PC1 dataset, which have been used in many cases. For each of these classifiers we are experimenting with 1,3,5 and 7 software repository metrics (numbers in [6] were in the same range), which we will describe below. To deal with the lack of data but still avoid overfitting, we will mostly be using 10-fold cross validation approach. Here it is important 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).

[1]More about weka: `http://www.cs.waikato.ac.nz/ml/weka/`

2.3 Evaluation metrics

To compare models 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 is important to notice which metrics are important and which are not, because it depends on the particular project and policy utilized within the project. For example is it important to decide if a team should focus on false positives or false negatives. In other words, is it worse for the test team to concentrate on defect-less but falsely classified classes or is it worse when testing teams does not spend enough time with the defect-rich classes because it was classified as "safe". Because of this we can not favor *precision* over *recall* or vice versa. In such case, *F-measure* offers itself as a good way to consider both of them. Additionaly, in Malhotra's article[4], one of the most used evaluation metrics was *area under the curve* so we're using that one as well. At last, we've also decided to compare the correctness of the classifiers. Despite it's not a right metric to use with skewed datasets, it's intuitive and very easy to understand.

F-measure is the harmonic mean of precision and recall.[4]

Area under the curve or AUC is the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example.[3]. It can be regarded as a measure of aggregated classification performance as it in some sense averages performance over all possible thresholds[1]

For each metric we have run a t-test to evaluate the significance of results. This process is described in more detail in section 5.

3 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. The article *Systematic Review of Machine Learning Techniques for Software Fault Prediction*[4] from Ruchika Malhotra explains a lot about the metrics and classifiers and we have based most of our decisions on this article.

3.1 Metrics

A typical software defect prediction model is trained by using previously collected fault data. Metrics characteristics or count influences the performance of the model.[6]. To decide the ideal number of metrics for the experiment, we looked at the results from Wang's paper on this topic[6]. Here it can be seen that models perform very good with just 3 metrics. The paper also demonstrates that models generally perform better with restricted number of metrics then with many(all) of them. There are several metric groups 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. Malhotre did extrensive research in her paper and found that in more than 50% of the 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[4, p. 15].

For our experiment, as mentioned above we decided to experiment with the number of metrics used in the fault prediction model. The metrics we decided to use are:

- Cyclomatic complexity: Is a count of linearly independent paths through a program's source code.
- Lines of code: Measure the size of a program by counting the number of lines in the text of the source code.
- Branch count: Number of branches in project
- Number of unique operands: The number of unique operands such as numerical, text and boolean values.
- Number of unique operators: The number of unique operators that evaluate the operands. Such could be plus, minus, multiply and divide.
- Halstead content: Rather complex metric representing algorithms complexity. It's affected by several other metrics[2]
- Maintenance severity: Describes difficulty of maintaining a module

It's been shown that models performing well seem to be using sets/combinations of various metrics (e.g. combinations of process, product and people-based metrics)[2]. Based on this knowledge, we tried to use both McCabe and Halstead groups and generally tried to use more complex metrics so as many as possible repository features could make an impact. But despite this, metrics have been chosen randomly and there remains a research to be done in this field as well (to see which exact metrics lead to better performance).

[2]More on Halstead content: `https://maisqual.squoring.com/wiki/index.php/Halstead_Intelligent_Content`

3.2 Classifier

Choosing the right classifier is also not a straightforward process since there are many statistics and machine learning methods to be used and to combine. During the last ten years, many papers and articles have been published on the topic of classifier performance in the SFP field. According to Malhotra[4],the most widely used classifiers are decision trees and Bayesian learners followed by neural networks and support vector machines[4, p. 11].
It is because of this, that we decided to use decision trees and bayesian learners, since we wish to get the best possible results in order to properly evaluate the difference between the number of metrics.

3.3 Datasets

Several dataset options are available for SFP studies, but NASAs Metrics Data Program data sets are definitely the most widely used. In her research, Malhotra[4] claims that they are used in more than 60% of the 64 examined studies. The data set called 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, the PC1 data set is used in 40% of studies and contains only 6.8% of faulty modules.[4, p. 17]. For our experiment we decided to use the PC1 dataset due to its bigger size and therefore more training data.

4 Experiment results

4.1 Decision Trees

The results of the decision tree models, can be seen in the following tables. Tables 0.1,0.2, 0.3 and 0.4 document the performance of all the decision tree models.

Algorithm	**Correct [%]**	**F-Measure**	**AUC**
Decision Stump	91.347	.872	.551
HoeffDing Tree	91.205	.871	.620
J48	91.347	.872	.491
LMT	91.489	.876	.573
Random Forest	91.489	.885	.601
Random Tree	92.198	.899	.595
REP tree	91.347	.872	.522
Mean	91.488	.878	.565

Table 0.1: Decision trees performance using 1 repository metric

Algorithm	Correct [%]	F-Measure	AUC
Decision Stump	91.347	.872	.714
HoeffDing Tree	91.205	.872	.491
J48	90.780	.874	.605
LMT	91.347	.888	.816
Random Forest	87.943	.874	.748
Random Tree	84.539	.851	.568
REP tree	90.922	.875	.701
Mean	89.726	.872	.663

Table 0.2: Decision trees performance using 3 repository metric

Algorithm	Correct [%]	F-Measure	AUC
Decision Stump	91.347	.872	.714
HoeffDing Tree	91.347	.872	.491
J48	90.354	.872	.616
LMT	91.347	.890	840
Random Forest	91.063	.898	.845
Random Tree	88.227	.882	.624
REP tree	91.347	.880	.636
Mean	90.719	.880	.681

Table 0.3: Decision trees performance using 5 repository metric

Algorithm	Correct [%]	F-Measure	AUC
Decision Stump	91.347	.872	.729
HoeffDing Tree	91.347	.872	.491
J48	90.070	.880	.679
LMT	91.489	.889	.827
Random Forest	91.773	.903	.851
Random Tree	89.503	.895	.668
REP tree	90.638	.873	.704
Mean	90.881	.883	.707

Table 0.4: Decision trees performance using 7 repository metric

4.2 Bayesian Learners

The results of the decision tree models, can be seen in the following tables. Tables 0.5,0.6, 0.7 and 0.8 document the performance of all the decision tree models.

Algorithm	Correct [%]	F-Measure	AUC
Bayes Net	91.347	.872	.491
Naive Bayes	88.794	.863	.524
Naive Bayes Multinomial	91.347	.872	.491
Naive Bayes Multinomial Text	91.347	.872	.491
Naive Bayes Multinomial Updatable	91.347	.872	.375
Naive Bayes Updatable	88.794	.863	.524
Mean	90.496	.869	.482

Table 0.5: Bayesian Learners performance using 1 repository metric

Algorithm	Correct [%]	F-Measure	AUC
Bayes Net	91.347	.872	.688
Naive Bayes	88.510	.878	.592
Naive Bayes Multinomial	88.227	.882	.749
Naive Bayes Multinomial Text	91.347	.872	.491
Naive Bayes Multinomial Updatable	88.227	.882	.748
Naive Bayes Updatable	88.510	.878	.592
Mean	89.361	.877	.643

Table 0.6: Bayesian Learners performance using 3 repository metric

Algorithm	Correct [%]	F-Measure	AUC
Bayes Net	89.645	.877	.754
Naive Bayes	88.085	.878	.624
Naive Bayes Multinomial	87.375	.883	.779
Naive Bayes Multinomial Text	91.347	.872	.491
Naive Bayes Multinomial Updatable	87.375	.883	.779
Naive Bayes Updatable	88.085	.878	.624
Mean	88.652	.878	.675

Table 0.7: Bayesian Learners performance using 5 repository metric

Algorithm	Correct [%]	F-Measure	AUC
Bayes Net	89.787	.876	.732
Naive Bayes	87.375	.872	.656
Naive Bayes Multinomial	85.815	.865	.638
Naive Bayes Multinomial Text	91.347	.872	.491
Naive Bayes Multinomial Updatable	87.375	.872	.643
Naive Bayes Updatable	87.375	.872	.656
Mean	88.179	.871	.636

Table 0.8: Bayesian Learners performance using 7 repository metric

5 Result analysis

5.1 Hypothesis Testing

After conducting the experiment, we obtained specific values for each algorithm in the two classifier groups that we decided to use. To compare those groups as a whole, we will work with the mean values of each group. We computed the mean value from all evaluation metrics and we also left out the maximal and minimal values of each group in order to prevent outliers from affecting our results. In both scenarios, the mean values have been almost the same and because of the limited scope of our report, we decided to do the hypothesis testing for the mean value computed from all evaluation metrics, which can be seen in the tables 0.9, 0.10 and 0.11.

Means of Correct [%]		
# of metrics	**Decision tree**	**Bayesian Learners**
1	91.488	90.496
3	89.726	89.361
5	90.719	88.652
7	90.881	88.179

Table 0.9: Means of Correct [%] evaluation metric

Means of F-measure		
# of metrics	**Decision tree**	**Bayesian Learners**
1	.878	.869
3	.872	.877
5	.880	.878
7	.883	.871

Table 0.10: Means of F-measure evaluation metric

Means of AUC		
# of metrics	**Decision tree**	**Bayesian Learners**
1	.565	.482
3	.663	.643
5	.681	.675
7	.707	.636

Table 0.11: Means of AUC evaluation metric

At first sight, it looks like that decision trees perform better than the Bayesian learners. To confirm this, we executed a paired t-tests for each metric (t-test is a statistical method to confirm or deny the null/alternative hypothesis). If there are clear hints

that one group performs better we're going to perform one-sided t-test whereas if the data don't display any supremacy of one group, two-sided t-test was used. For the F-measure evaluation metric, we executed a two-tailed t-test since mean values do not clearly display performance supremacy of any group. On the other side, correctness and AUC have always showed slightly better performance of decision trees over bayesian learners so in their case we can perform one-tailed t-test.
We measured performance for 4 different scenarios which results into 6 degrees of freedom ($n1+n2-2$). With confidence level set to standard default value 95% the critical t-value found in statistical table for t-distribution is 2.447 and 2.015 (for 2-tailed and 1-tailed t-test respectively). As can be seen in table 0.12, according to t and p-values we can

Metric	**T-test tails**	**Degrees of freedom**	**Confidence level [%]**	**t-value**	**t-value threshold**	**p-value**
Correctness	1	6	95	2.8945	2.015	0.06278
F-measure	2	6	95	1.1852	2.447	0.3213
AUC area	1	6	95	2.3921	2.015	0.09656

Table 0.12: Hypothetis testing results

reject null hypothesis in favor of an alternative hypothesis in both cases of the one-tailed tests (correctness, AUC). This can, in other words, be interpreted that when speaking in terms of correctness and AUC, decision trees really performed better than bayesian learners. On the other side, experiment with F-measure as evaluation metric didn't show performance advantage in any of 2 groups.

Another area we concentrated on in our experiment was the effect of number of software metrics used to train the prediction models. Results can be seen in tables...

6 Conclusion

For this project we can conlude.. and so on..

Bibliography

[1] Karel Dejaeger, Thomas Verbraken, and Bart Baesens. Toward comprehensible software fault prediction models using bayesian network classifiers. *IEEE Transactions on Software Engineering*, 39(2):237–257, 2013.

[2] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2012.

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

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

[5] Shivkumar Shivaji, Jr E James Whitehead, Ram Akella, and Sunghun Kim. Reducing features to improve bug prediction. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 600–604. IEEE Computer Society, 2009.

[6] Huanjing Wang, Taghi M Khoshgoftaar, and Naeem Seliya. How many software metrics should be selected for defect prediction? In *Twenty-Fourth International FLAIRS Conference*, 2011.