Comparing Multilayer Perceptron and Echo State Network in Software Risk Analysis

Markus Mettälä & Piyush Paliwal

January 21, 2013

1 Abstract

In this project we experimented with a dataset containing initially 17 features to predict the binary outcome of bugs or no bugs. We experimented with baseline methods and multilayer perceptron. As an addition we build a echo state network and compare the results from it to the baseline methods and multilayer perceptron. The results from the experiments were quite clear that echo state network does not compare in the performance in this particular dataset.

Keywords: Supervised Learning, Data Preprocessing, Feature Selection/Extraction, Logistic Regression, Multilayer Perceptron Neural Network, Echo State Network

2 Introduction

Software risk analysis is an important factor when developing software that is to be sold and used by others. By performing a risk analysis on software the developers can get an reasonably good overview of the state of the program. This state can tell to developers whether it is mature enough to be delivered to users or whether the risk of finding a bug in the system is too large.

Bugs in a software can be of many forms, they can be severe or just minors. Depending on the classification of the bug the developers can make a decision of whether the bug is acceptable or if it is vital that it is fixed before shipping to users.

Being able to identify the bugs in the system even without using the software would be an valuable asset to the person making decisions about the systems release. One of the methods used to identify bugs is using software metrics to analyze the source code.

Software metrics describe the source code in the sense of its structure. For example in object oriented programming the number of private and public methods are one metric. There are many other metrics developed

and to use these to predict the number of bugs in a system even when the system is under development is an interesting approach.

There are other approaches to predicting the bugs in a software, or doing the software risk analysis, for exmaple one method is to explore the development history of the project and identify files which contain bugs the most often. With this approach there is some delay in the prediction of bugs as only the history is known and the current state is not that much taken into account.

By using software metrics to predict bugs the current state of the software is only interesting and we can therefore keep on running the analysis and get a new metrics for the quality of the software.

In this report we will first describe the methods used to predict the software quality in section 3. Next we will describe the experiments we did and the dataset we used in the experiments in section 4. After the experiments we will present our results in section 5 and before concluding we discuss the results of the project in section 6.

3 Methods

3.1 Baseline Methods

We have proposed logistic regression as baseline method(can be seen as low complexity classifier) before trying MLP or Echo state network. And observed the performance of model.

3.1.1 Logistic regression

Logistic regression measures the relationship between a categorical class variable and several continuous input variables (features) by converting the class to probability score. In other way, it measures the estimation of the probability of something occur, (estimating the instances to relate with corresponding class) by fitting the features to logistic function.

3.2 Multilayer Perceptron

Multilayer perceptron network is a feedforward neural network. There can be multiple layers of nodes in the network and each of the layers are fully connected to the next layer. All nodes except the inputd nodes in the network use nonlinear activation function, such as hyperbolic tangent. Multilayer perceptrons are commonly trained with backpropagation algorithms.

As our implementation of multilayer perceptron is quite standard we will not go into too deep detail of its inner workings. The only difference to the techniques learned in the course is the resilient backpropagation algorithm used to train the network.

Martin Riedmille [10] describes the resilient backpropagation algorithm in his technical report in the following way "Resilient backpropagation is a local adaptive learning scheme, performing supervised batch learning in multi-layer perceptrons" [10]. The basic principle is to only consider the sign of the partial derivative in weight step to indicate the direction of the weight update [10].

3.3 Reservoir Computing

Reservoir computing is a methodology using recurrent neural networks with backpropagation-decorrelation learning rule, more commonly now referred to as reservoir computing [9].

The main idea behind reservoir computing is to build a random reservoir of hundreds of sparsely connected neurons. The motivation for this is to avoid the shortcomings of gradient-descent recurrent neural network training such as non-convergence, parameter update can be computationally costly, long-term memory is hard to learn and advanced training algorithms are mathematically involved [9]. In addition to random reservoir the output of the reservoir is a linear combination of neuron signals from reservoir and can be obtained with linear regression.

There are basically two main approaches in reservoir computing for the reservoirs, liquid state machine and echo state networks. In the next section we will describe echo state network in more a bit detail as it is the one which we used in our project.

3.3.1 Echo State Network

Echo state network is based on the observation that a random recurrent neural network possesses certain algebraic properties and training a linear output from reservoir achieves excellent results in practical applications [9]. The name comes from the dynamic reservoir which states are the echoes of its input history [9]. The neurons in reservoir typically use sigmoid functions such as hyperbolic tangent.

Another important feature in echo state networks is the use of leaky integrator neurons. These neurons allow a leaky integration of its activation function from previous inputs [9].

There are many other important features of echo state networks which can be found for example in paper by Lukosevicius et al. [9]. Other features described in the paper was not usefull for us since we used the echo state network in time-independent classification and followed the descriptions of Luis A. Alexandre et al. [6] more closely.

4 Experiments

This section describes the experiments we did on the dataset with different approaches. We user R for all the computations and Weka for some data preprocessing. Next we will describe the dataset we used and how we experimented with it.

To evaluate the methods we used accuracy and F-measure. We mainly focus on the F-measure because the classes are not equally distributed thus accuracy does not give perfect results. Also all experiments was done by 10-fold cross validation and the results were the averages of the 10 folds.

4.1 Dataset

The dataset we chose for our problem is a freely available collection of software metrics from several open source projects [1]. From these projects we chose Eclipse JDT Core dataset and specifically churn dataset.

This dataset contains 989 samples and many software metrics values for them. There is bug value indicated in a number from 0 upwards. Because we only want to do classification of bug or no bug we convert the dataset so that all zero bug samples remain as zero bugs and all other values are converted to one. This way we can make the bug value a bit more evenly distributed, but it still is majority of zero bugs with a ratio of 4 zeros to 1.

4.1.1 Data preprocessing

We checked the summary of data and looked for if it contains some impossible values or missing values. And we didn't found any. However, the features did not have the same unit, thus we standardized them such that they have zero mean and unit standard deviation.

4.2 Feature Selection

In this section we describe the feature selection methods we used and the resulting selected features.

4.2.1 CFS Subset Feature Selection

CFS method gives subset of feature that gives best set of feature based on correlation with classification that evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them.

Using this method features were reduced from original 17 to 10 following by: Cbo, fanOut, lcom, numberOfAttributes, numberOfAttributesInherited, numberOfLinesOfCode,numberOfMethodsInherited,rfc,wmc ,numberOfPrivateMethods

4.2.2 Logistic Regression as Wrapper (Based on AIC Criteria)

The AIC criteria is the deviance of the fitted model considering regularization in the sense to penalize the number of attributes. It evaluates attribute sets by using a learning scheme. Cross validation is used to estimate the accuracy of the learning scheme for a set of attributes. Off course results would vary depending on the threshold set to discard the attributes.

Feature were reduced from original 17 to 10 following by: fanIn, wmc, cbo, numberOfPrivateAttributes, dit, fanOut, numberOfMethods, rfc, numberOfPrivateMethods, numberOfAttributes

4.2.3 PCA Feature Extraction

PCA(principal component analysis) is used for feature extraction task. Further more we can have reduced feature that are not the original features but the transformed feature of PCA. Each of the transformed features is linear combination of all of the original ones. We preserve 95% of original information that the original data have. 100% information indicates we have as many number of transformed features as the original ones, consequences NO dimensionality reduction. So in order to reduced dimensionality we preserve at least 95 % of original information. (Of course we may wish to preserve even lesser than 95%, then we will have to lose more information of the original features. That might not give too good results).

10 transformed feature names PC1,PC2. . . PC10 were enough to preserve this 95

4.3 Baseline Methods

WIth baseline methods we experimented by running the algorihtms with all different feature sets. As these methods were not our main focus of experiments we did not do much with them.

4.4 Multilayer Perceptron

In experiments with multilayer perceptron we experimented with the effect of different number of nodes in hidden layer to the performance of the prediction, so it was a single hidden layer network. We also changed the stopping threshold for error in training phase to see how much it effects the results. Also as in other experiments we used four different features sets for results.

4.5 Echo State Network

In our experiments with echo state network we experimented with the effects of the size of the reservoir, leaking rate and sparsity of the reservoir. As we are more or less performing the same type of task as Luis A. Alexandre et

al. [6] we will try their recommendations of reservoir size 200 and sparsity of 20 %.

Also to verify that our echo state network is performing correctly we compare the result of Luis A. Alexandre et al. [6] in one dataset. For this test we chose the Synthetic dataset from Ripley [2] which is similar to our problem with two binary classes.

The main idea of using echo state networks in time-independent tasks is to keep the same input to the reservoir until the output of the nodes in the reservoir do not change significantly [6].

5 Results

In this section we describe the results we obtained from the experiments. We begin with baseline methods then present results from multilayer perceptron and finally the results of echo state network.

5.1 Baseline Methods

Table 1 shows the results for different feature sets in logistic regression. when compared to other results from neural network based methods the results are not so good. Even echo state network performs better than these..

Table 1: Results from logistic regression

Feature set	F-measure	Accuracy
All	0.2433	0.8304
PCA	0.2558	0.8314
AIC	0.2435	0.8334
CFS	0.2470	0.8294

5.2 Multilayer Perceptron

We experimented with AIC features and the number of hidden neurons. Figures 1 and 2 shows the results from these experiments. It can be seen from the results that actually the less there are neurons the better the F-measure is. For the accuracy there is no significant effect from the number of hidden neurons.

Experiment results with CFS features on multilayer perceptron can be seen in figures 3 and 4. The results are similar to the ones with AIC features. The difference is that with CFS features the results are generally at a beter level while with AIC feature the F-measures were better with less neurons.

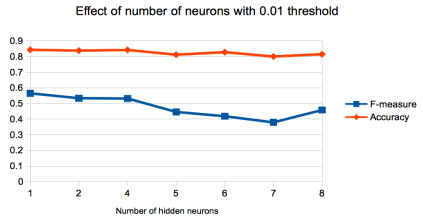

Figure 1: Effect of neurons with threshold 0.01 and AIC features.

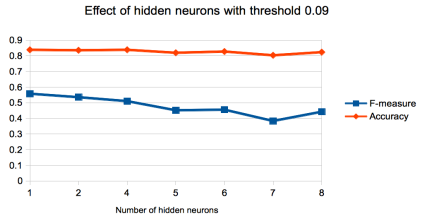

Figure 2: Effect of neurons with threshold 0.09 and AIC features.

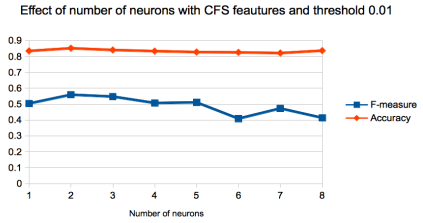

Figure 3: Effect of neurons with threshold 0.01 and CFS features.

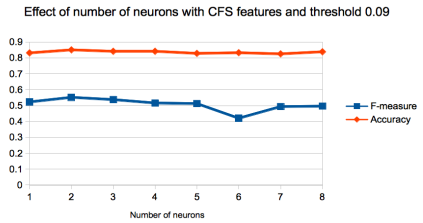

Figure 4: Effect of neurons with threshold 0.09 and CFS features.

Figures 5 and 6 show the results of using all features with multilayer perceptron. When we compare these results to other results for multilayer perceptron they are clearly not as good.

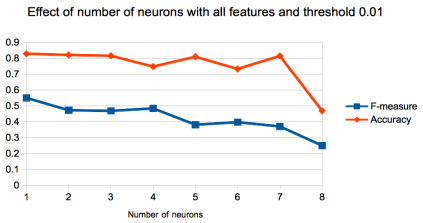

Figure 5: Effect of neurons with threshold 0.01 and all features.

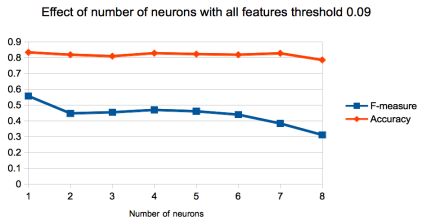

Figure 6: Effect of neurons with threshold 0.09 and all features.

Figures 7 and 8 show the results of using PCA features with multilayer perceptron. The results indicate that the less neurons with PCA features yields better results. This is consistent with other multilayer perceptron experiments.

From the results, we can observe best model for logistic regression was the one with PCA features. For MLP, best F-measure could be considered either with PCA features(F-measure=0.5658 with optimal hidden node=1) or with AIC feature(0.5644 and optimal hidden node=1)

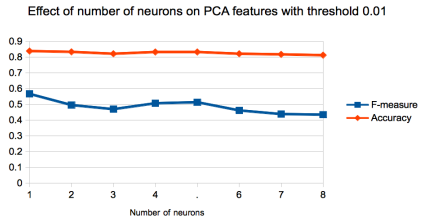

Figure 7: Effect of neurons with threshold 0.01 and PCA features.

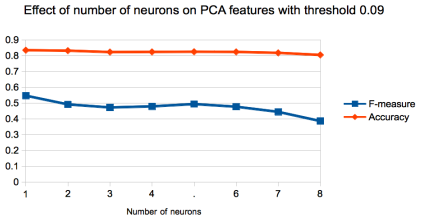

Figure 8: Effect of neurons with threshold 0.09 and PCA features.

5.3 Echo State Network

To begin with we started the development from the basic echo state network script for R developed by Mantas Lukoševičius [3]. This script was developed for time-series prediction so we had to modify it according to the approaches described in paper by Luis A. Alexandre et al. [6].

The main modification we had to do for the code was that inputs are not changed to the reservoir until the node outputs do not change significantly. The resulting outputs from some of the nodes can be seen in figure 9. It can be seen that the outputs of the reservoir nodes actually do converge within a few hundred iterations.

Next we tested our echo state network against the synthetic dataset from Ripley [2]. The results in Alexandre et al. [6] were that the results were 90.725 % correct. In our experiments we got a accuracy of 92 % and F-measure of 0.917 so we are confident that our implementation functions correctly.

Next we experimented with the effect of reservoir size on all selected features by the feature selection process. We kept the leaking rate at a constant 0.2 and sparsity at 90 % so that there were 90 % of all possible connections connected in the reservoir.

Figures 10 through 13 shows the results of the experiments. We can see from the results that depending on the features used the reservoir size might affect quite much to the resulting accuracy or not. For example in PCA features there is not that much variation in the results. But in the other three feature sets there can be found some values which are clearly worse or better than others.

Next experiment was to see the result of leaking rate to the performance of the echo state network. The results can be seen in figures 14 through 17. The results are in this case more constant than compared to thre results from reservoir size experiment. There is much less variation in the results.

Our final experiment with echo state networks was with the sparsity. Figures 18 through 21 shows the results from the experiments. In this experiment we found the most effect in the results. There are a lot of

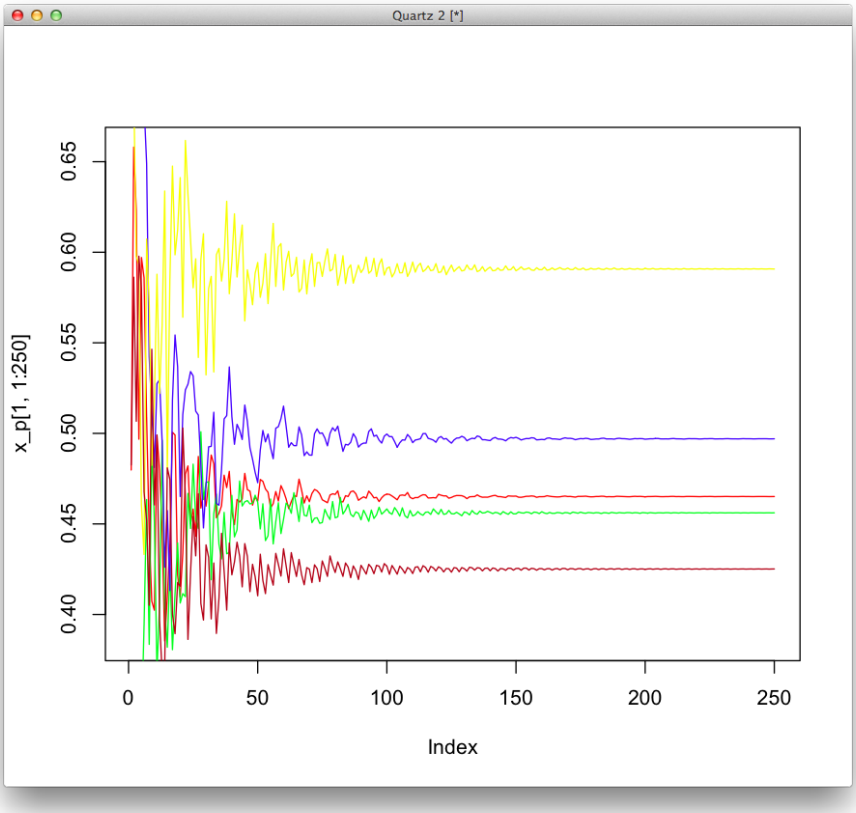

Figure 9: Stabilization of some of the output signals from nodes in reservoir.

effect of sparsity in all feature sets except in AIC features. With all features in can be seen clearly that 0.8 sparsity yields best results and similarly for CFS features 0.9 yields best results. For PCA experiment it is more difficult to find the optimum value but it appears to be in the lower rance of 0.2 sparsity.

All in all the results we got for this problem with echo state networks were not even close to the performance presented in Alexandre et al. [6] paper or even to the performance we got for the same dataset. Even if we would combine the optimal parameters for each feature set we would not achieve much better results.

The best combination of the results was with reservoir size of 200 (F-measure 0.376206) in CFS features. For sparsity the best value was 0.9 with CFS features again (F-measure 0.376206). And for leaking rate best value was 0.4 with CFS features (F-measure 0.39954).

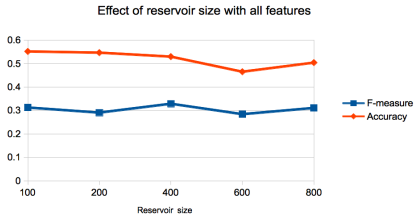

Figure 10: Reservoir size on all features.

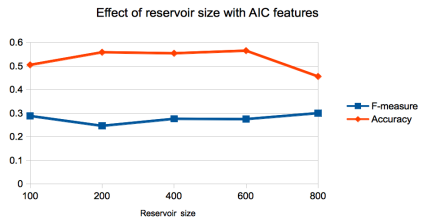

Figure 11: Reservoir size on AIC features.

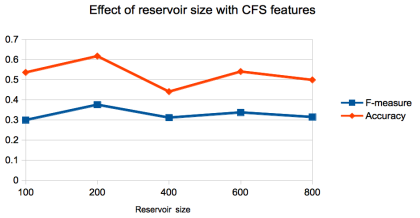

Figure 12: Reservoir size on CFS features.

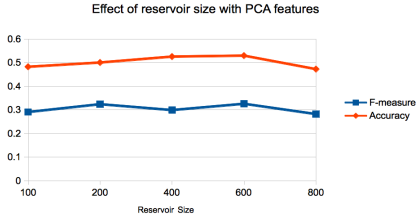

Figure 13: Reservoir size on PCA features.

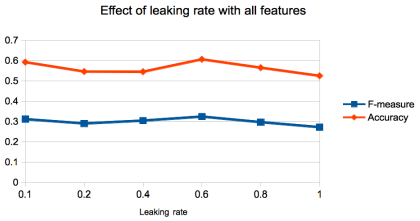

Figure 14: Leaking rate on all features

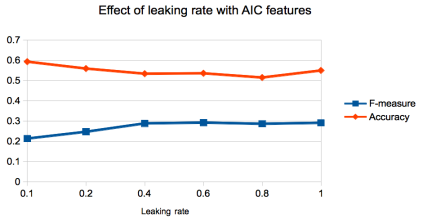

Figure 15: Leaking rate on AIC features

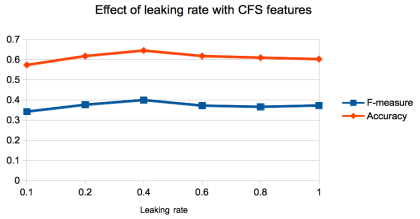

Figure 16: Leaking rate on CFS features

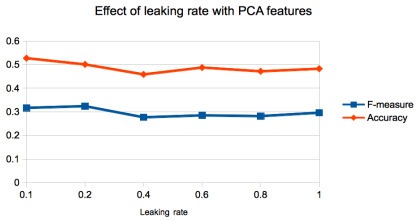

Figure 17: Leaking rate on PCA features

6 Discussions

We can clearly see from the results that echo state network is not comparable in the performance at this particular problem. The results are almos half

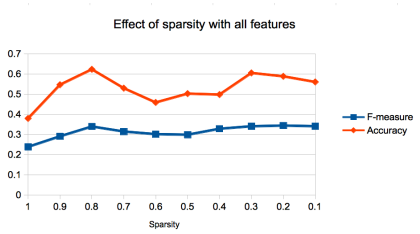

Figure 18: Sparsity on all features

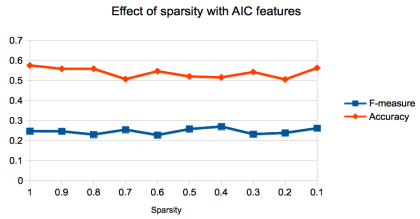

Figure 19: Sparsity on AIC features

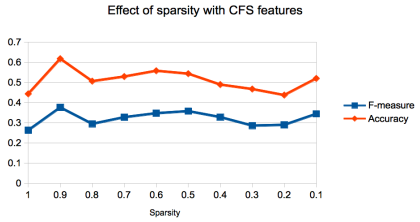

Figure 20: Sparsity on CFS features

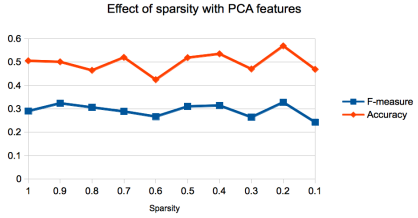

Figure 21: Sparsity on PCA features

of the results achieved by other methods. We did not find any clear reason why echo state networks behave so badly in this dataset. Against baseline methods echo state network provides a slightly better performance.

Perhaps one reason might be in the number of features why echo state networks did not do better. In the dataset which we used from the paper of Luis A. Alexandre et al. [6] the synthetic dataset [2] contains only two inputs. In our dataset there were ten inputs in the reduced datasets and 17 in the full dataset.

Also one important factor that affected the results of all experiments was the uneven distribution of classes. There were many more of class zero than there were of class one. We did a small experiment with echo state network with a subset of the data so that the classes were evenly distributed. With this dataset we could get an 10 % improvement to the accuracy and F-measure. But even with this improvement the results are quite much worse than the ones achivede from other methods even with unevenly distributed classes.

7 Conclusions

We can conclude that the echo state networks do not compare to other neural network approaches or to baseline methods. The echo state itself seems to perform well on specific situations where the complexity of the dataset is smaller.

We also learned a lot of echo state networks by building one itself and also the behavior of multilayer perceptron became more clear.

In multilayer perceptron, we only tried hidden nodes to optimize in the range of 1 to 8. However, we can not say this to be the best strategy as optimal solution could be out of this range that we have not tried anymore. Perhaps we should have used even more large incremental ranges and after that tiny ranges only around the number of hidden node which was found best from larger range. However, we may observe that increasing the nodes (in our case more than 6 or 7) decreases the performance which might suggest us to stop more tuning of nodes.

We can also observe that feature selection affected the result at significant scale. Multilayer perceptron and Echo state network took too much time compare to the logistic regression. Its because of their complex computation phenomenon. Still MLP performed better than logistic regression.

References

[1] Marco D'Ambros, Michele Lanza, Romain Robbes, *Bug prediction dataset*, http://bug.inf.usi.ch/, accessed 19.1.2013.

[2] B.D. Ripley, *Pattern Recognition and Neural Networks by B.D. Ripley*, http://www.stats.ox.ac.uk/pub/PRNN/, accessed 19.1.2013.

[3] Mantas Lukoševičius, *Sample ESN source codes*, http://minds.jacobs-university.de/mantas/code, accessed 19.1.2013.

[4] Donald E. Neumann, *An Enhanced Neural Network Technique for Software Risk Analysis*, IEEE transactions on software engineering, vol. 28, no. 9, September 2002.

[5] Sunghun Kim, E. James Whitehead Jr. and Yi Zhang, *Classifying Software Changes: Clean or Buggy?*, IEEE transactions on software engineering, vol. 34, no. 2, March/April 2008.

[6] Luís A. Alexandre, Mark J. Embrechts and Jonathan Linton, *Benchmarking Reservoir Computing on Time-Independent Classification Tasks.*

[7] Tibor Gyimo'thy, Rudolf Ferenc, and Istva'n Siket, *Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction*, IEEE transactions on software engineering, vol. 31, no. 10, October 2005.

[8] Renu Kumar, Suresh Rai and Jerry L. Trahan, *Neural-Network Techniques for Software-Quality Evaluation*, Proceedings Annual Reliability and Maintainability Symposium, 1998.

[9] Mantas Lukoševičius, Herbert Jaeger, *Reservoir computing approaches to recurrent neural network training*, Computer Science Review 3, pp. 127–149, 2009.

[10] Martin Riedmiller, *Rprop - Description and Implementation Details*, Technical Report, Institut Für Logik, Komplexität und Deduktionssyteme, University of Karlsruhe, January 1994.