Using Threshold Derivation of Software Metrics for Building Classifiers in Defect Prediction

MARINO MOHOVIĆ, GORAN MAUŠA and TIHANA GALINAC GRBAC, University of Rijeka

The knowledge about the software metrics, which serve as quality indicators, is vital for the efficient allocation of resources in quality assurance activities. Recent studies showed that some software metrics exhibit threshold effects and can be used for software defect prediction. Our goal was to analyze if the threshold derivation process could be used to improve a standard classification models for software defect prediction, rather than to search for universal threshold values. We proposed two classification models based on Bender method for threshold derivation to test this idea, named Threshold Naive Bayes and Threshold Voting. Threshold Naive Bayes is a probabilistic classifier based on Naive Bayes and improved by threshold derivation. Threshold Voting is a simple type of ensemble classifier which is based solely on threshold derivation. The proposed models were tested in a case study based on datasets from subsequent releases of large open source projects and compared against the standard Naive Bayes classifier in terms of geometric mean (GM) between true positive and true negative rate. The results of our case study showed that the Threshold Naive Bayes classifier performs better than the other two when compared in terms of GM. Hence, this study has shown that threshold derivation process for software metrics may be used to improve the performance of standard classifiers in software defect prediction. Future research will analyze its effectiveness in general classification purposes and test on other types of data.

1. INTRODUCTION

Software defect prediction (SDP) aids the process of software quality assurance by identifying the fault prone code in an early stage, thus reducing the number of defects that need to be fixed after the implementation phase [Mauša and Galinac Grbac, 2017]. Therefore, having a successful prediction model can greatly reduce the cost of the product development cycle [Boucher and Badri, 2016]. Studies carried out to analyze the distribution of defects in large and complex systems encourage this field of research [Galinac Grbac et al., 2013; Galinac Grbac and Huljenic, 2015]. Software metrics are often used for SDP and the defect prediction models built on the basis of product metrics are already well known. Different types of classifiers have been used and analyzed for the purpose of SDP [Lessmann et al., 2008]. Novel paradigms like the usage of multi-objective genetic algorithms empowered by the concepts of colonization show that there is still more for progress [Mauša and Galinac Grbac, 2017].

However, some studies focused more on the software metrics and the calculation of metrics thresholds for SDP purpose [Arar and Ayan, 2016]. It encouraged us to consider both the threshold calculation and the known prediction models and combine these methods for SDP. Consequently, this paper continues our research conducted on the stability of software metrics threshold values for software defect prediction [Mauša and Galinac Grbac, 2017]. The threshold values are generally not stable across dif-

This work has been supported in part by Croatian Science Foundation's funding of the project UIP-2014-09-7945 and by the University of Rijeka Research Grant 13.09.2.2.16.
Author's address: M. Mohovicć; email:mmohovic@riteh.hr; G. Mauša, Faculty of engineering, Vukovarska 58, 51000 Rijeka, Croatia; email: gmausa@riteh.hr; T. Galinac Grbac, Faculty of engineering, Vukovarska 58, 51000 Rijeka, Croatia; email: tgalinac@riteh.hr.

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.
In: Z. Budimac (ed.): Proceedings of the SQAMIA 2018: 7th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Novi Sad, Serbia, 27–30.8.2018. Also published online by CEUR Workshop Proceedings (http://ceur-ws.org, ISSN 1613-0073)

CEUR-WS.org/Vol-2217/paper-moh.pdf

ferent projects and do not represent a reliable predictor for software defects when used alone. However, within the same project and project release, a stable threshold level can be found. Thus, we concluded that rather than searching for universal threshold values, threshold values should be used in a different way to aid software quality assurance. In this paper we investigate that idea and our main research question (RQ) is:

—RQ: Does the threshold derivation process have the potential to improve a standard classifier for the defect prediction purpose?

We proposed two novel algorithms based on the threshold derivation process, namely Threshold Naive Bayes (TNB) and Threshold Voting (TV). The proposed TNB algorithm enhances the standard naive Bayes classifier using the calculated levels of thresholds for SDP, while TV is based on the percentage of metrics which exceeded the threshold level. We used the naive Bayes model as a base for our TNB as it is one of the most common classification techniques successfully used in various application domains [Zhang, 2004].

In this paper, we present a case study which compares our two proposed algorithms, TNB, TV with the standard Naive Bayes (NB) in terms of geometric mean (GM) between true positive rate and true negative rate. Our algorithms use the ensemble of many metrics predictions for prediction of the fault prone code, rather than using single metric prediction, therefore they provide more accurate results for SDP. The algorithms are tested on 10 different dataset versions using 10 fold cross-validation to overcome the sampling bias [Alpaydin, 2004]. The results we have obtained show some improvement when using the TV over NB (1-14%) and a much greater improvement when using the TNB over NB, in range from 5% up to 22% in terms of GM. The TNB also performed better in most cases than the TV algorithm (0.5-11%) and we concluded that the proposed TNB algorithm gives best results for SDP overall.

The rest of the paper is organized in the following way: related work that motivated this study is presented in Section 2; the research methodology is described in Section 3; the details of our case study are given in Section 4; the results are presented and discussed in Section 5; and the paper is concluded in Section 6.

2. BACKGROUND

Many different machine learning models have been used for fault proneness prediction. A study that benchmarked different classification models for SDP showed that the logistical regression model and naive Bayes based model acquired good results in SDP [Lessmann et al., 2008]. Some studies also analyzed the threshold rate for naive Bayes classifier which is used for SDP [Blankenburg et al., 2014; Tosun and Bener, 2009]. An algorithm for efficient computation of a rejection threshold was proposed and its authors had come to conclusion that using such a model reduces the classification errors [Blankenburg et al., 2014]. Another study that worked on ways to improve the Naive Bayes classifier used the decision threshold optimization and analyzed the changes in prediction performance measures [Tosun and Bener, 2009]. They managed to reduce the probability of false alarms by changing the naive Bayes default threshold rate of 0.5. According to their work, naive Bayes classifier can be used for software defect prediction but it needs some adjustments so that it can be more accurate.

Moreover, recent studies focused more on software metrics threshold levels using ROC curves and Alaves ranking[Boucher and Badri, 2016] and the threshold derivation process [Arar and Ayan, 2016] for the SDP purpose. They showed that for some metrics an efficient threshold level could be calculated and used for SDP. It motivated us to continue our research on software threshold levels and analyze whether they can be used to improve the performance of naive Bayes model for SDP. Unlike some studies [Blankenburg et al., 2014; Tosun and Bener, 2009], our study focused on the calculation of

metrics threshold levels rather than changing the default naive Bayes threshold decision rate for SDP. These calculated metrics thresholds would be used for the fitting of the naive Bayes model for SDP purpose without changing the NB decision threshold.

3. METHODOLOGY

The methodology for the construction of threshold derivation process based classifiers is schematically presented in Fig. 1. The threshold derivation is presented in the upper part, while the lower part presents our two proposed algorithms. The initial step of presented methodology is to divide the data in 10 parts of equal size and class distribution for 10-fold cross-validation using stratified sampling method. These parts are then used as follows: 7 parts are used for training, 2 for validation and 1 part for testing purposes. Two classifiers based on the threshold derivation method are proposed by this methodology, named Threshold Voting (TV) and Threshold naive Bayes (TNB). The confusion matrices are calculated for each algorithm for performance comparison. Further details about the proposed methodology are described in the next few subsections.

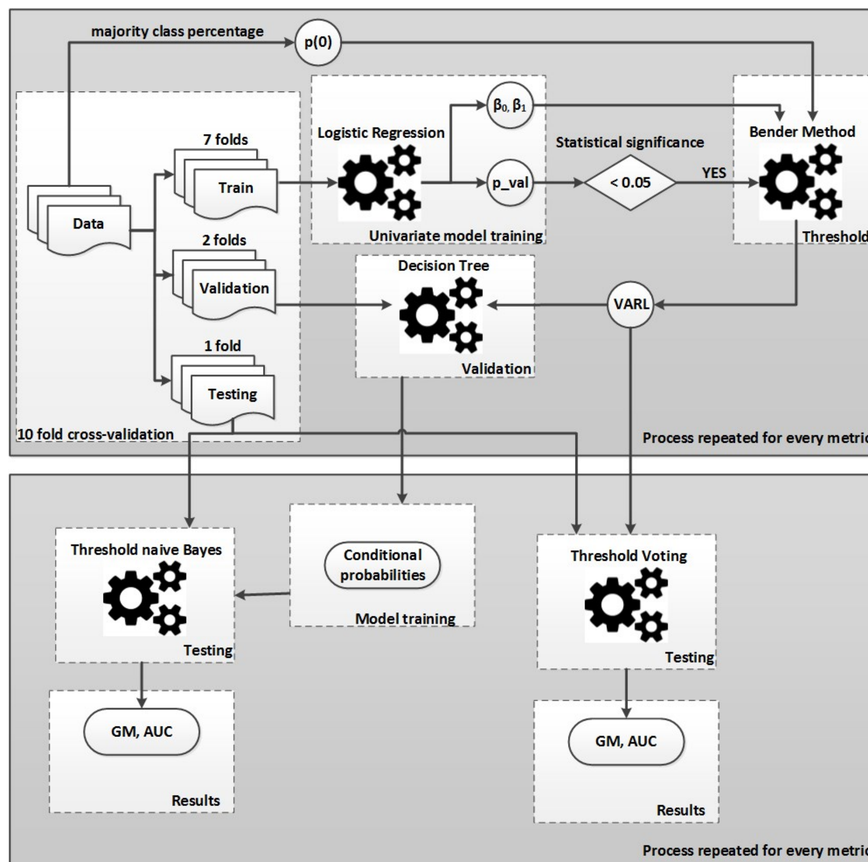

Fig. 1. Methodology of this case study

3.1 Threshold Derivation

The threshold is derived by using the univariate binary logistic regression model, a classification technique in which one dependent variable can assume only one of two possible cases. The probability for the occurrence of each case is defined by the logistical regression equation:

$$P(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}} \tag{1}$$

where P(X) is the case probability, X represents the metric value, and β_0 and β_1 are the logistic regression coefficients. The coefficients are calculated for every metric separately. The logistic regression coefficients along with the percentage of the majority class p_0 are used by the Bender's method for threshold derivation [Bender, 1999]. Bender defined a method for calculation of the Value of an Acceptable Risk Level (VARL) based on a logistic regression equation (1). The method computes the VARL using the following equation:

$$VARL = \frac{1}{\beta_1}\,(ln(\frac{p_0}{1-p_0}\,) - \beta_0) \tag{2}$$

where β_0 and β_1 are logistical regression coefficients and p_0 is the base probability. The ratio of the faulty modules is used as the base probability as in [Arar and Ayan, 2016], since it yields best results. Threshold level is calculated only for the metrics which are statistically significant for univariate logistic regression model. Therefore, a one-tailed significance test with a 95% confidence level (0.05) is used to determine whether the corresponding coefficient is statistically significant or not. If the p-value is greater than 0.05, the algorithm skips that metric and calculates threshold levels only for the significant ones. This process is the standard way of calculating the threshold levels [Mauša and Galinac Grbac, 2017; Arar and Ayan, 2016].

Afterwards, the algorithm iterates through the validation data part and compares the calculated threshold levels with real metric values. If the metric value exceeds the threshold level for a given instance, the individual metric prediction classifies that instance as fault-prone and otherwise as non fault-prone. This step is repeated for every metric and the results are many individual software defect predictions based on metric values. Our proposed classification algorithms, TNB and TV, are trained based on the mentioned individual metrics predictions.

3.2 Classification Algorithms

3.2.1 *Standard naive Bayes Classifier* . The Naive Bayes Classifier is a probabilistic multivariate classifier based on the Bayes theorem. In our case, the multiple attributes correspond to different software metrics and software modules, respectively. Using the conditional probability *P(E|H)*, we can calculate the probability of an event *E* using its prior knowledge *P(H)*:

$$P(H\,|E) = \frac{P(E\,|H) * P(H)}{P(E)} \tag{3}$$

Naive Bayes classifier calculates these probabilities for every attribute. This step trains the naive Bayes model which is afterward used for prediction. The mentioned model is trained on the raw data which is a combination of the training and validation data part. After the conditional probabilities are calculated they are used in Naive Bayesian equation:

$$\begin{aligned} h_{bayes} &= argmax P(H)P(E|H) \\ &= argmax P(H)\prod_{i=1}^{n} P(a_i|H) \end{aligned} \tag{4}$$

to calculate the posterior probability (MAP estimation) for each class. The class with the highest posterior probability is the outcome of prediction which in our case predicts the fault-prone code or non fault-prone code. These predictions are being made on the testing data part which is also used for other two algorithms.

3.2.2 *Threshold naive Bayes.* Our proposed classification algorithm TNB is based on the naive Bayes Classifier described in the previous subsection. The difference is that it is being trained on individual metric predictions mentioned in the subsection 3.1, rather than on the raw metrics data as explained in previous subsection 3.2.1

The training of the TNB model is based on the individual predictions, i.e. calculated threshold levels from the validation data part. These individual metric predictions are evaluated using the evidences of the fault prone code in validation data part which derives 4 different conditional probabilities for every metric (attribute):

—probability that the metric predicted that there will not be a defect in code under condition that the defect actually did not happen $P(X_i = 0 \mid Y = 0)$,

—probability that the metric predicted that there will not be a defect in code under condition that the defect actually happened $P(X_i = 0 \mid Y = 1)$,

—probability that the metric predicted that there will be a defect in code under condition that the defect actually happened $P(X_i = 1 \mid Y = 1)$ and

—probability that the metric predicted that there will be a defect in code under condition that the defect actually did not happen $P(X_i = 1 \mid Y = 0)$,

where X_i represents the individual metric prediction of a fault prone code and Y represents the evidence of a fault prone code which is stored in the last column of the validation dataset. These probabilities correspond to the number of cases in which a certain condition (eg. X_i=1, Y=1) happened. Therefore, by calculating the percentage for each of the four mentioned conditions and repeating this step for every significant metric, a model similar to NB is trained. Consequently, the mentioned validation data part is actually used as the second training part for our model.

Finally, the TNB predicts every instance of testing data part using these calculated conditional probabilities and the calculated threshold levels as fault-prone or non fault-prone. The prediction process starts by comparing the real metric values with the calculated metrics threshold levels which generates the individual metric predictions for the testing data part. This step is similar to the step where conditional probabilities are being calculated. The difference is that these individual metrics predictions are now being used in next step along with calculated conditional probabilities to finally predict the class of an instance. It is done by multiplying trained conditional probabilities for every metric which correspond to the $P(a_i|H)$ from the naive Bayes equation (4), iteratively through metrics in one row. The individual metrics predictions correspond to the a_i and the percentage of a class correspond to P(H). The result of these steps are two posterior probabilities (MAP estimations) for every file (posterior probability that the instance is class 1 and posterior probability that the instance is class 2). These probabilities are then compared mutually and the decision is made based on which of the two is more likely.

3.2.3 *Threshold voting.* Our second proposed classification algorithm, named Threshold Voting also relies on individual metrics predictions based on the calculated threshold levels. Threshold levels are calculated based on training data part and the validation data part combined into one dataset as it does not need the validation part. This step trains the TV classifier model. Afterwards, these calculated threshold levels are being compared with the real metric values from the testing data part which

generates the individual metric predictions. Finally, it calculates the percentage of the individual metrics predictions which predicted class 1 and the percentage of the individual metrics predictions which predicted class 2. The final classification is based on majority vote for these two percentages, i.e. the higher percentage predicts the instance as fault-prone or non fault-prone.

4. CASE STUDY

Matlab 2016 environment was used to develop the algorithms and to perform this case study. The details of the study will be explained in the next few subsections.

4.1 Datasets

The case study is based on empirical SDP datasets from 5 subsequent releases of two open source projects of Eclipse community, JDT and PDE. The chosen datasets are carefully collected, structured in a matrix form and formated as csv (comma-separated values) files and open to public use [Mauša et al., 2016]. The matrix contains the description of software metrics and the paths of java files within a project's release. There are 48 product metrics, later used as independent attributes X for prediction, and the number of defects in the last column, later transformed into binary dependent attribute Y. Attribute Y is fault-prone (FP) if there the number of defects is greater than 0 and non-fault-prone (NFP) otherwise. The defects are faults (bugs) that caused a loss of functionalities in software, i.e. whose severity is classified as minor or above [Mauša et al., 2016]. To visualize the number and size of datasets and the distribution of FP and NFP files, we presented Table I.

The case study is based on 10 fold cross-validation, so the datasets are divided into 10 parts using the stratified sampling technique. We combined 7 parts of the dataset as the training set, 2 parts as the validation set, and 1 part as the testing part. This process is repeated in 10 iterations, changing the order of parts used for training, validation and testing in every iteration.

Table I. Description of datasets

PROJECT RELEASE	NUMBER OF FILES	FP	NFP
JDT R2.0	2397	45.97%	54.03%
JDT R2.1	2743	31.94%	68.06%
JDT R3.0	3420	38.6%	61.4%
JDT R3.1	3883	32.76%	67.24%
JDT R3.2	2233	36.54%	63.46%
PDE R2.0	576	19.27%	80.73%
PDE R2.1	761	16.29%	83.71%
PDE R3.0	881	31.21%	68.79%
PDE R3.1	1108	32.22%	67.78%
PDE R3.2	1351	46.19%	53.81%

4.2 Evaluation

After the algorithms have been trained and tested, we evaluated their performance of defect prediction. Since the stratified sampling has been used to divide the data, the percentage of the two classes is kept within all the data parts, therefore all the classifiers are fairly compared with the base class probability (p0) being provided for all three classifiers. Comparing the predicted class with actual class value for each java file within the testing part, we computed the elements of confusion matrix. Confusion matrix consists of four possible outcomes: true positive (TP) and true negative (TN) for correct classification and false positive(FP) and false negative(FN) for incorrect classification.

Using the four possible outcomes, we computed evaluation measures standard for SDP: true positive rate (TPR), true negative rate (TNR) and geometric mean (GM) using Equations 5, 6 and 7. TPR and

TNR represent the accuracy of the positive and the negative class of data, respectively. GM represent the geometric mean between TPR and TNR and it usually used to evaluate the performance of binary classification in presence of unbalanced datasets [Japkowicz and Shah, 2011].

$$TPR = \frac{TP}{TP + FN} \tag{5}$$

$$TNR = \frac{TN}{TN + FP} \tag{6}$$

$$GM = \sqrt{TPR * TNR} \tag{7}$$

5. RESULTS

We analyzed the software defect prediction performance of the three classifiers using the real data and we computed the confusion matrices. The arithmetic mean of TPR, TNR and GM values for every dataset release is presented in Table II. Boxplots of the GM results is given in Figure 2 for a more thorough representation. TNB results are colored green, TV results are colored red and Standard NB results are colored black and dataset releases are separated with the vertical line as every release was tested separately.

Table II. Average values of GM results for TNB, TV and NB classifiers in JDT and PDE project releases

Classifier:	TNB	SV	NB	TNB	SV	NB	TNB	SV	NB	TNB	SV	NB	TNB	SV	NB
Release	R2.0			R2.1			R3.0			R3.1			R3.2		
JDT project															
TPR	0.53	0.39	0.26	0.57	0.41	0.24	0.59	0.41	0.28	0.56	0.37	0.29	0.55	0.36	0.32
TNR	0.77	0.86	0.92	0.81	0.89	0.93	0.81	0.90	0.93	0.81	0.90	0.94	0.81	0.90	0.94
GM	0.64	0.58	0.48	0.68	0.60	0.47	0.69	0.61	0.51	0.67	0.58	0.52	0.67	0.56	0.55
PDE project															
TPR	0.68	0.66	0.57	0.58	0.51	0.45	0.57	0.49	0.41	0.61	0.55	0.40	0.48	0.39	0.32
TNR	0.85	0.87	0.90	0.81	0.83	0.89	0.79	0.84	0.88	0.79	0.83	0.90	0.72	0.79	0.88
GM	0.76	0.75	0.70	0.68	0.64	0.62	0.67	0.64	0.60	0.69	0.67	0.60	0.59	0.55	0.53

The GM results have shown that the TNB classifier performs better than the NB classifier for all dataset releases. For JDT project releases the prediction improvement over the NB is in range from 11% to 22%, while the improvement for PDE releases is in range from 5% to 10%. The TV also performed better than the NB for most releases, between 1% - 14% for JDT and between 2% - 8% for PDE. The overall improvement in terms of GM is the result of a higher TPR for our two proposed algorithms, which is very important for the good SDP. The NB has very high TNR because it predicts most instances as non fault prone but in the same time it gives very low prediction accuracy of the fault prone instances, which is not very useful for the quality insurance. Our proposed algorithms have slightly lower TNR than the NB but substantially higher TPR which gives better overall results. Finally, our TNB classifier also performs better than our second proposed algorithm, TV. When compared mutually the improvement of TNB over TV is between 1% - 11%.

6. CONCLUSION

The goal of this study was to analyze if the threshold derivation process could be used to improve the standard classification techniques for software defect prediction. We proposed two novel algorithms

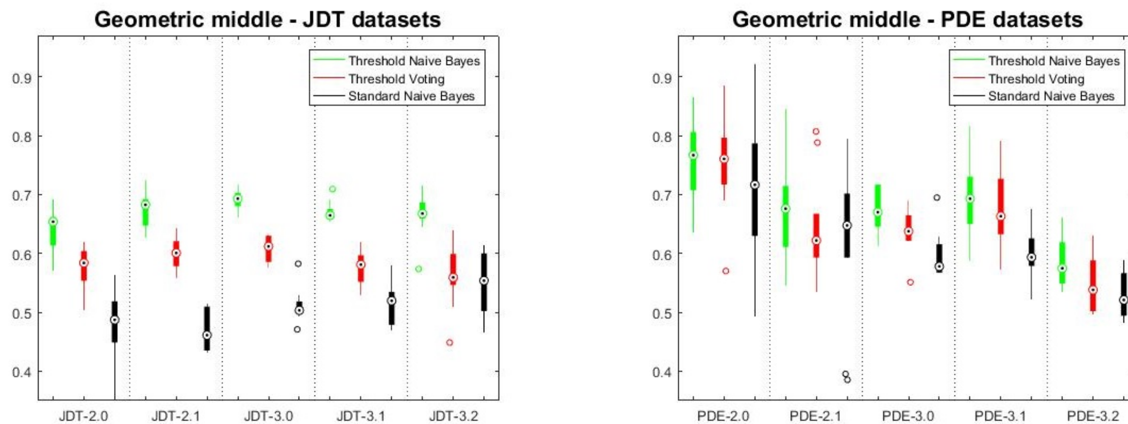

Fig. 2. Box and whisker plot for TNB, TV and NB classifiers

based on threshold derivation of software metrics, namely Threshold Naive Bayes - TNB and Threshold Voting - TV. The proposed algorithms were compared against the standard naive Bayes classifier, which inspired us to develop the TNB classifier.

The main contribution of this study is the new approach and methodology for improvement of the standard classification techniques using the threshold derivation. We hope that our approach will serve as guidelines for future related research in the SDP research area. The conclusions we obtained with this study are:

—Threshold derivation method may be used for improving the performance of standard classification models for SDP.

—The improvement of the proposed TNB classifier over the standard NB in terms of GM ranges from 3% to 22%, and it gives the best results for SDP overall.

—The improvement of the proposed TV classifier over the standard NB in terms of GM ranges from 1% to 14%.

—The standard, unmodified naive Bayes algorithm does not give very good result for SDP, because it has a very low TPR.

Conclusion validity of this research is strong because we used standard techniques for sampling and statistical analysis. The used methodology is explained in details and the case study is based on publicly available datasets so the whole process is easy to replicate. However, our conclusions are based on a small scale case study and this limits their external validity. This study should be extended on a larger scale for more general conclusions and our future work will replicate it on additional SDP datasets.

REFERENCES

G. Mauša and T. Galinac-Grbac ”The Stability of Threshold Values for Software Metrics in Software Defect Prediction” in *Proc. of MEDI*, Lecture Notes in Computer Science, vol 10563, 2017, pp. 81-95.

A. Boucher and M. Badri ”Using Software Metrics Thresholds to Predict Fault-Prone Classes in Object-Oriented Software” in *Proc. 4th Intl. Conf. on Applied Computing and Information Technology*, 2016, pp. 169-176

T. Galinac Grbac, P. Runeson and D. Huljenic. ”A second replicated quantitative analysis of fault distributions in complex software systems”. *IEEE Trans. Softw. Eng.*, vol. 39, pp. 462-476, Apr. 2013.

T. G. Grbac and D. Huljenic. "On the probability distribution of faults in complex software systems". *Information and Software Technology*, vol. 58, pp. 250-258, Feb. 2015.

S. Lessmann, B. Baesens, C. Mues and Swantje Pietsch. "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings", *IEEE Trans. Softw. Eng.*, vol. 34, pp. 485-496, Jul./Aug. 2008.

G. Mauša and T. Galinac Grbac. "Co-evolutionary Multi-Population Genetic Programming for Classification in Software Defect Prediction: an Empirical Case Study". *Applied soft computing* , vol. 55, pp. 331-351, Jun. 2017.

O. F. Arar and K. Ayan. "Deriving Thresholds of Software Metrics to Predict Faults on Open Source Software". *Expert Systems With Applications*, vol. 61, pp. 106-121, Nov. 2016

H. Zhang. "The Optimality of Naive Bayes" in *Proc. FLAIRS Conference*, 2004

E. Alpaydin. *Introduction to Machine Learning* MIT Press, 2004.

M. Blankenburg, C. Bloch and J. Krger . "Computation of a Rejection Threshold Used for the naive Bayes Classifier" in *Proc. 13th International Conference on Machine Learning and Applications*, 2014, pp. 342 - 349.

A. Tosun and A. Bener, "Reducing False Alarms in Software Defect Prediction by Decision Threshold" in *Proc. Third International Symposiumm on Empirical Software Engineering and Measurement Optimization*, pp. 477 - 480, 2009.

R. Bender. "Quantitative Risk Assessment in Epidemiological Studies Investigating Threshold Effects". *Biometrical Journal*, vol. 41, pp. 305-319, 1999.

G. Mauša, T. Galinac Grbac and B. Dalbelo Basic. "A systematic data collection procedure for software defect prediction". *Computer Science and Information Systems*, vol. 13, pp. 173-197, 2016.

N. Japkowicz and M. Shah. *Evaluating Learning Algorithms: A Classification Perspective.* New York, NY: Cambridge University Press, 2011, pp. 100-101

T. Fawcett. "An introduction to ROC analysis". *Pattern Recognition Letters*, vol. 27, pp. 861-874, Jun. 2006.

V. B. Kampenes, T. Dyb, J. E. Hannay and Dag I. K. Sjberg. "Systematic Review: A systematic review of Effect Size in Software Engineering Experiments". *Information and Software Technology*, vol.49, pp. 1073-1086, Nov. 2007.

W. Fu and T. Menzies, "Revisiting Unsupervised Learning for Defect Prediction" in *Proc. ESEC/FSE17, 11th Joint Meeting on Foundations of Software Engineering Paderborn*, 2017, pp. 72-83

H. He and E. A. Garcia. "Learning from Imbalanced Data". *IEEE Trans. on knowledge and data engineering*, vol. 21, pp. 1263-1284, Sep. 2009.