Stability of Software Defect Prediction in Relation to Levels of Data Imbalance

TIHANA GALINAC GRBAC AND GORAN MAUŠA, University of Rijeka
BOJANA DALBELO–BAŠIĆ, University of Zagreb

Software defect prediction is an important decision support activity in software quality assurance. Its goal is reducing verification costs by predicting the system modules that are more likely to contain defects, thus enabling more efficient allocation of resources in verification process. The problem is that there is no widely applicable well performing prediction method. The main reason is in the very nature of software datasets, their imbalance, complexity and properties dependent on the application domain. In this paper we suggest a research strategy for the study of the performance stability using different machine learning methods over different levels of imbalance for software defect prediction datasets. We also provide a preliminary case study on a dataset from the NASA MDP open repository using multivariate binary logistic regression and forward and backward feature selection. Results indicate that the performance becomes unstable around 80% of imbalance.

Categories and Subject Descriptors: D.2.9 [**Software Engineering**]: Management—*Software quality assurance (SQA)*

Additional Key Words and Phrases: Software Defect Prediction, Data Imbalance, Feature Selection, Stability

1. INTRODUCTION

Software defect prediction is recognized as one of the most important ways to reach software development efficiency. The majority of costs during software development is spent on software defect detection activities, but their ability to guarantee software reliability is still limited. The analyses performed by [Andersson and Runeson 2007; Fenton and Ohlsson 2000; Galinac Grbac et al. 2013], in the environment of a large scale industrial software with high focus on reliability shows that faults are distributed within the system according to the Pareto principle. They prove that the majority of faults are concentrated in just small amount of system modules, and that these modules do not compose a majority of system size. This fact implies that software defect prediction would really bring benefits if a well performing model is applied. The main motivating idea is that if we were able to predict the location of software faults within the system, then we could plan defect detection activities more efficiently. This means that we would be able to concentrate defect detection activities and resources into critical locations within the system and not on the entire system.

Numerous studies have already been performed aiming to find the best general software defect prediction model [Hall et al. 2012]. Unfortunately, a well performing solution is still absent. Data in software defect prediction are very complex, and do not follow in general any particular probability distribution that could provide a mathematical model. Data distributions are highly skewed, which is connected to the popular *data imbalance problem*, thus making standard machine learning approaches inadequate. Therefore, a significant research has recently been devoted to cope with this problem.

Author's address: T. Galinac Grbac, Faculty of Engineering, Vukovarska 58, HR–51000 Rijeka, Croatia; email: tgalinac@riteh.hr; G. Mauša, Faculty of Engineering, Vukovarska 58, HR–51000 Rijeka, Croatia; email: gmausa@riteh.hr; B. Dalbelo–Bašić, Faculty of Electrical Engineering and Computing, Unska 3, HR-10000 Zagreb, Croatia; email: bojana.dalbelo@fer.hr.

In: Z. Budimac (ed.): Proceedings of the 2nd Workshop of Software Quality Analysis, Monitoring, Improvement, and Applications (SQAMIA), Novi Sad, Serbia, 15.-17.9.2013, published at http://ceur-ws.org

Several solutions are offered for the data imbalance problem. However, these solutions are not equally effective in all application domains. Moreover, there is still an open question regarding the extent to which imbalanced learning methods help with learning capabilities. This question should be answered with extensive and rigorous experimentation across all application domains, including software defect prediction, aiming to explore underlaying effects that would lead to fundamental understandings [He and Garcia 2009].

The work presented in this paper is a step in that direction. We present an research strategy that aims to explore performance stability of software defect prediction models in relation to levels of data imbalance. As an illustrative example we present an experiment taken to Stability of Software Defect Prediction in Relation to Levels of Data Imbalance our strategy. We observed how learning performance, with and without stepwise feature selection, in case of logistic regression learner, is changing over a range of imbalances in the context of software defect prediction. The findings are just indicative and are to be explored by exhausting experimenting aligned with proposed strategy.

1.1 Complexity of software defect prediction data

Software defect prediction (SDP) is concerned with early prediction of system modules (file, class, module, method, component, or something else) that are likely to have a critical number of faults (above certain threshold value, THR). In numerous studies it is identified that these modules are not so common. In fact, they are special cases, and that is why they are harder to find. **Dependent variable** in learning models is usually a binary variable with two classes labeled as 'fault–prone' (FP) and 'not–fault–prone' (NFP). The number of FP modules usually is much lower, and represents a *minority class*, than the number of NFP modules which represents a *majority class*. Datasets with significantly unequal distributions of minority over majority class are *imbalanced*. **Independent variables** used in SDP studies are numerous. In this paper we will address SDP based on the static code metrics [McCabe 1976].

In SDP datasets the level of class imbalance varies for various software application domains. We reviewed the software engineering publications dealing with software defect prediction and we noticed that the percentage of the non-fault prone modules (%NFP) in the datasets varies a lot (from 1% in medical record system [Andrews and Stringfellow 2001] to more then 94% in telecom system [Khoshgoftaar and Seliya 2004]) for various software application domains (telecom industry, aeronautics, radar systems, etc.). Since there are SDP initiatives on datasets with a whole range of imbalance percentages, we are motivated to determine the percentage at which data imbalance becomes a problem, i.e., learners become unstable.

As already mentioned above, the random variables measured in software engineering usually do not follow any distribution in general, and the applicability of classical mathematical modeling methods and techniques is limited. Hence, algorithms from the machine learning have been widely adopted. Among various learning methods used in the defect prediction approaches, this paper will explore the capabilities of **multivariate binary logistic regression (LR)**. Our ultimate goal is not to validate different learning algorithms but to explore learning performance stability over different levels of imbalance. The LR has shown very good performance in the past and is known to be a simple but robust method. In [Lessmann et al. 2008] it is the 9th best classifier among 22 examined (9/22) and at the same time it is the 2nd best statistical classifier among 7 of them (2/7). The stepwise regression classifier was the most accurate classifier (1/4) and was outperformed only in cases with many outliers in [Shepperd and Kadoda 2001]. Very good performance of logistic regression was also observed in [Kaur and Kaur 2012] (3/12 it terms of accuracy and AUC), [Banthia and Gupta 2012] (1/5 both with and without preprocessing of 5 raw NASA datasets), [Giger et al. 2011] (1/8 in terms of median AUC from 15 open source projects), [Jiang et al. 2008] (2/6 in terms of AUC and 3/6 according to Nemenyi

post-hoc test), etc. However, neither of the studies has analyzed the performance of logistic regression classifier in relation to data imbalance. The study [Provost 2000] assumes that in majority of published work the performance of logistic learner would be significantly improved, if it is adequately used. We will refer to this issue in more detail in Section 3.

As in the whole software engineering field, an important problem in software defect prediction is the lack of quality industrial data, and therefore generalization ability and further propagation of research results is very limited. The problem is usually that this data are considered as confidential by the industry, or the data are not available at all for industry with low maturity. To overcome these obstacles, there are initiatives for open source repositories of datasets aligned with the goal of improving generalization of research results. However, the problem of generalization still remains, because usually the open repositories contain data from a particular type of software (e.g. NASA MDP repository, open source software repositories, etc.) and/or of questionable quality [Gray et al. 2011].

In this study we used **NASA MDP datasets** and have carefully addressed all the potential issues, i.e. removed duplicates [Gray et al. 2012]. This selection is motivated by simple comparison of results with the related work, so that our contribution can be easily incorporated to the existing knowledge base of imbalance problem in the SDP area.

1.2 Experimental approach

Our goal is to explore stability of evaluation metrics for learning SDP datasets with machine learning techniques across different levels of imbalance. Moreover, we want to evaluate potential sources of bias in study design by constructing number of experiments in which we diverse one parameter per experiment. Parameters that are subject of change are explained briefly in Sect.2.

To integrate conclusions obtained from each experiment a meta–analytic statistical analysis is proposed. These methods are suggested by number of authors as tool for generalizing the results and integrating knowledge across many studies [Brooks 1997]. We propose the following steps:

(1) ***Acquiring data.*** A sample S of independent random variables $X_1, \ldots, X_n$ measuring different features of a system module, and a binary dependent variable Y measuring fault–proneness (with $Y = 1$ for FP modules and $Y = 0$ for NFP modules) is obtained from a repository (e.g. open repository, open source projects, industrial projects).

(2) ***Data preprocessing.***
   (a) *Data cleaning, noise elimination, sampling.*
   (b) *Data multiplication.* From the sample S obtained in step (1) a training set of size $2/3$ the size of S and a validation set of size $1/3$ the size of S are chosen at random k times. In this way k training samples $T_1, \ldots, T_k$ and k validation samples $V_1, \ldots, V_k$ are obtained. These samples are categorized into ℓ categories with respect to the data imbalance defined as the percentage of the NFP modules in T_i and calculated as: $\%NFP_{T_i} = \frac{FP_{T_i}}{FP_{T_i}+NFP_{T_i}}$.
   (c) *Feature selection.* For each training set T_i a feature selection is performed. As a result some of the random variables X_j are excluded from the model. The inclusion/exclusion frequencies of X_j for each of the categories introduced in step (2b) are recorded.

(3) ***Learning.***
   (a) *Building a learning model.* A learning model is built for each training set T_i using the learning techniques under consideration.
   (b) *Evaluating model performance.* Using the validation set V_i, the model built in step (3a) is evaluated using various evaluation metrics. Let M be the random variable measuring the value of one of these metrics.

(4) ***Statistical analysis.***

(a) *Variation analysis.* The differences between ℓ samples of a random variable M obtained from samples T_i and V_i belonging to different categories introduced in step (2b) are analyzed using statistical tests. This step is repeated for each evaluation measure used in step (3b).
(b) *Cross-dataset validation.* The whole process is repeated from step (1) for m datasets from various application domains and sources. The differences between $\ell \cdot m$ samples of a random variable M are analyzed using statistical tests and the results reveal whether general behavior exists.

To summarize, the conclusions are based on the results of statistical tests comparing the mean values of performance evaluation metrics (see Table I) across different data imbalances of a training sample. The stability of performance evaluation metrics obtained with different feature selection procedures is evaluated in the same way.

2. DATA IMBALANCE

Data imbalance has received considerable attention within the data mining community during the last decade. It becomes a central point of this research, since the problem is present in a majority of data mining application areas [Weiss 2004]. In general data imbalance degrades the learning performance. The problem arises with learning accuracy of the minority class, in which we are usually more interested. Usually, we are interested to timely predict rare events represented by the minority class, for which the probability of its occurrence is low, but its occurrence leads to significant costs.

For example, suppose that only very low number of system modules is faulty, which is the case with systems with very low tolerance on failures (e.g. medical systems, aeronautic system, telecommunications, etc.). Suppose that we did not identify faulty module with the help of a software defect prediction algorithm, and due to that have developed defect detection strategy not concentrating on that particular module. Thus, we omit to identify a fault in our defect detection activity, and this fault slips to the customer site. Failure caused by this fault at customer site would then imply significant costs contained of several items: paying penalty to customer, losing customer confidence, causing additional expenses due to corrective maintenance, additional costs in all subsequent system revisions and additional cost during system evolution. This cost would be considered as misclassification cost of wrongly classified positive class (note that positive class in the context of defect prediction algorithm is a faulty module). On the other hand, misclassification cost of wrongly classified negative class would be much lower, because it would involve just more defect detection activities. Obviously, the misclassification costs are unequally weighted and this is the main obstacle in applying standard machine learning algorithms, because they usually assumes the same or similar conditions in learning and application environment [Provost 2000].

The study [Provost 2000] makes a survey of data imbalance problems and methods addressing these problems. Although different methods are recommended for data imbalance problems, it does not give definite answers regarding their applicability in the application context. Some answers are obtained by other researchers in that field afterwards, and a more recent survey is given in [He and Garcia 2009]. Still no definite guideline exists that could guide practitioners.

2.1 Dataset considerations

The most popular approach to the class imbalance problem is the usage of artificially obtained balanced dataset. There are several sampling methods proposed for that purpose. In a recent work [Wang and Yao 2013] an experiment with some of the sampling methods is conducted. However, it is concluded in [Kamei et al. 2007] that sampling did not succeed to improve performance with all the classifiers. In [Hulse et al. 2007] it is identified that classifier performance is improved with sampling, but individual learners respond differently on sampling.

Another problem with datasets is that in practice, the datasets are often very complex, involving a number of issues like overlapping, lack of representative data, within and between class imbalance, and often high dimensionality. The effects of these issues were widely analyzed separately sample size in [Raudys and Jain 1991], dimensionality reduction: [Liu and Yu 2005], noise elimination [Khoshgoftaar et al. 2005], but not in conjunction with the data imbalance. The study performed in [Batista et al. 2004] observes that the problem is related to a combination of absolute imbalance and other complicating factors. Thus, the imbalance problem is just an additional issue in complex datasets such as datasets for software defect prediction.

Different aspects of feature selection in relation to class imbalance has been studied in [Khoshgoftaar et al. 2010; Gao and Khoshgoftaar 2011; Wang et al. 2012]. All these studies were performed on datasets from the NASA MDP repository. In this work we also used a stepwise feature selection as a preprocessing step, because the dataset is high dimensional and we experiment with logistic regression. Hence, we were able to investigate the stability of the performance with and without feature selection procedure over different levels of imbalance.

Besides the methods explained above for obtaining artificially balanced datasets, another approach is to adapt standard machine learning algorithms to operate for imbalance datasets. In that case the learning approach should be adjusted to the imbalanced situation. A complete review of such approaches and methods can be found in [He and Garcia 2009].

2.2 Evaluation metrics

Another problem of standard machine learning algorithms for imbalanced data is in usage of inadequate evaluation metrics during learning procedure or to evaluate final result. Evaluation metrics are usually derived from the confusion matrix and are given in Table I. They are defined in terms of the following score values. A true positive (TP) score is counted for every correctly (true) classified fault-prone module, and a true negative (TN) score for every correctly (true) classified non-fault-prone module. The other two possibilities are related to false prediction. A false positive (FP) score is counted for every false classified or misclassified non-fault-prone module (often referred to as Type II error), and a false negative (FN) score is counted for every false classified or misclassified fault-prone module (often referred to as Type I error) [Runeson et al. 2001; Khoshgoftaar and Seliya 2004]. For example, classification accuracy ACC, the most commonly used evaluation metric in standard machine learning algorithms, is not able to value the minority class appropriately, and leads to poor classification performance of minority class.

In the case of class imbalance, the precision (PR) and recall (TPR) metrics given in Table I are recommended in number of studies [He and Garcia 2009], as well as the F–measure and G–mean which are not used here. The precision and recall in combination give a measure of correctly classified fault–prone modules. Precision measures exactness, i.e., how many fault–prone modules are classified correctly, and recall measures completeness, i.e., how many fault–prone are classified correctly.

Table I. Evaluation metrics

Metrics	Definition	Formula
Accuracy (ACC)	number of correctly classified modules divided by total modules	$\frac{TP+TN}{TP+FP+TN+FN}$
True positive rate (TPR) (sensitivity, recall)	number of correctly classified fault–prone modules divided by total number of fault-prone modules	$\frac{TP}{TP+FN}$
Precision (PR) (positive predicted value)	number of correctly classified fault–free modules divided by total number of fault–free modules	$\frac{TP}{TP+FP}$

The output of a probabilistic machine learning classifier is the probability for a module to be fault-prone. Therefore, a cutoff percentage has to be defined in order to perform classification. Since choosing a cutoff value leaves room to bias and possible inconsistencies in a study [Lessmann et al. 2008], there is another measure that deals with that problem called the area under curve, AUC [Fawcett 2006]. It takes into account the dependence of TPR and a similar metric for false positive proportion on the cutoff value.

All of the aforementioned techniques are not cost sensitive, and in the case of rare cases with very high misclassification cost of type I error the key performance indicator is cost. The most favorable evaluation criteria for imbalanced datasets are cost curves and is also recommended in [Jiang et al. 2008] for SDP domain.

3. PRELIMINARY CASE STUDY

To illustrate the application of the research strategy proposed in Section 1.2, verify strategy, provide evidence for the dependence of the machine learning performance on the level of data imbalance, and indicate our future goals, we have undertaken a preliminary case study.

(1) Dataset KC1 from NASA MDP repository has been acquired. It consists of $n = 29$ features, i.e., independent variables X_j. The dependent variable in this dataset is the number of faults in a system module. From this variable we derived binary dependent variable Y by setting ten different thresholds for fault proneness, from 1 to 19 with step of 2 (1, 3, 5,...). In this way we obtained ten different samples S and we continue the analysis for all of them.

(2) (a) The well known issues with the dataset are eliminated using data cleaning tool [Shepperd et al. 2013].

(b) For each of the ten samples obtained in step (1), we made 50 iterations of the random splitting into training and validation samples. Thus we obtained $k = 500$ samples T_i and V_i with the range of data imbalance from 51% to 96%. The samples are categorized into $\ell = 5$ categories of equal length (each spanning 9%).

(c) In the case study we also consider the influence of a feature selection procedure, as already mentioned in 2. We consider the forward and backward stepwise selection procedure [Han and Kambar 2006]. The decision for inclusion and exclusion of a feature is based on level of statistical significance, the $p - value$. The common significance levels for inclusion and exclusion of features are used as in [Mausa et al. 2012; Briand et al. 2000] with $p\_in = 0.05$ and $p\_out = 0.1$ respectively. The percentage of inclusion of a feature for both procedures and different categories of data imbalance are given in Table II. We conclude that feature selection stability of some features is very tolerant to data imbalance (e.g. Feature 5, 22, 28, 29 is always excluded, for both forward and backward model). Some features are very stable until certain level of balance (for example Feature 2 is always included 100% until category with data imbalance of 78%). It is also interesting to observe that some features have similar feature selection stability in ideal balance case and highly imbalanced case, whereas for moderate imbalance have opposite feature selection decision.

(3) (a) Learning models are built using multivariate binary logistic regression (LR) [Hastie et al. 2009]. The model incorporates more than just one predicting variable and in fault predicting case performs according to the equation

$$\pi\left(X_1, X_2, ...X_n\right) = \frac{e^{C_0+C_1X_1+...+C_nX_n}}{1+e^{C_0+C_1X_1+...+C_nX_n}}, \tag{1}$$

where C_j are the regression coefficients corresponding to X_j, and π is the probability that a fault was found in a class during validation. In order to obtain a binary outgoing variable,

Table II. Percentage of inclusion of a feature

	'51% -60%'		**'60% -69%'**		**'69% -78%'**		**'78% -87%'**		**'87% -96%'**	
	Forw.	Back.	Forw.	Back.	Forw.	Back.	Forw.	Back.	Forw.	Back.
Ft. 1	4,8%	26,2%	1,7%	20,3%	0,0%	9,1%	5,2%	12,2%	14,8%	15,8%
Ft. 2	100,0%	100,0%	100,0%	100,0%	95,5%	100,0%	73,3%	72,7%	44,3%	43,2%
Ft. 3	78,6%	76,2%	91,5%	84,7%	34,1%	77,3%	30,8%	63,4%	16,9%	38,3%
Ft. 4	2,4%	9,5%	10,2%	45,8%	18,2%	52,3%	0,6%	6,4%	0,0%	1,6%
Ft. 5	7,1%	14,3%	5,1%	16,9%	2,3%	22,7%	1,2%	27,3%	2,2%	7,1%
Ft. 6	0,0%	11,9%	5,1%	11,9%	0,0%	4,5%	2,3%	16,9%	3,8%	29,0%
Ft. 7	9,5%	23,8%	42,4%	47,5%	15,9%	50,0%	4,7%	13,4%	0,0%	9,8%
Ft. 8	2,4%	14,3%	8,5%	16,9%	15,9%	20,5%	55,2%	52,9%	84,2%	88,0%
Ft. 9	4,8%	28,6%	10,2%	37,3%	0,0%	22,7%	3,5%	26,7%	0,5%	19,1%
Ft. 10	2,4%	23,8%	6,8%	32,2%	2,3%	20,5%	1,2%	37,2%	2,2%	60,1%
Ft. 11	0,0%	11,9%	0,0%	15,3%	2,3%	43,2%	20,9%	40,7%	13,7%	17,5%
Ft. 12	7,1%	23,8%	1,7%	22,0%	54,5%	86,4%	35,5%	65,1%	9,8%	26,8%
Ft. 13	9,5%	42,9%	20,3%	39,0%	0,0%	27,3%	5,8%	43,6%	2,2%	60,1%
Ft. 14	21,4%	19,0%	16,9%	16,9%	0,0%	6,8%	2,9%	15,1%	0,0%	13,1%
Ft. 15	4,8%	23,8%	5,1%	13,6%	0,0%	13,6%	0,6%	8,1%	6,0%	19,1%
Ft. 16	11,9%	21,4%	1,7%	6,8%	0,0%	25,0%	1,2%	15,7%	2,2%	19,1%
Ft. 17	4,8%	40,5%	1,7%	27,1%	0,0%	11,4%	4,7%	14,0%	17,5%	23,5%
Ft. 18	9,5%	16,7%	27,1%	18,6%	2,3%	38,6%	18,6%	26,2%	12,6%	21,3%
Ft. 19	7,1%	11,9%	25,4%	37,3%	0,0%	9,1%	1,2%	27,3%	0,0%	21,9%
Ft. 20	0,0%	45,2%	1,7%	54,2%	0,0%	45,5%	0,0%	43,6%	2,2%	22,4%
Ft. 21	4,8%	16,7%	0,0%	39,0%	0,0%	27,3%	0,0%	31,4%	0,0%	44,3%
Ft. 22	0,0%	9,5%	3,4%	8,5%	0,0%	0,0%	0,0%	2,3%	0,0%	0,5%
Ft. 23	7,1%	21,4%	16,9%	30,5%	0,0%	6,8%	1,2%	26,2%	0,0%	13,1%
Ft. 24	0,0%	21,4%	0,0%	27,1%	0,0%	20,5%	0,0%	20,9%	0,5%	20,2%
Ft. 25	11,9%	35,7%	10,2%	71,2%	0,0%	9,1%	1,7%	18,0%	10,9%	29,5%
Ft. 26	7,1%	52,4%	1,7%	59,3%	0,0%	20,5%	0,0%	30,8%	1,1%	33,3%
Ft. 27	35,7%	50,0%	8,5%	20,3%	0,0%	4,5%	2,3%	16,9%	11,5%	23,0%
Ft. 28	2,4%	14,3%	11,9%	13,6%	6,8%	4,5%	0,6%	10,5%	1,6%	15,3%
Ft. 29	0,0%	11,9%	1,7%	3,4%	4,5%	0,0%	0,6%	2,3%	0,0%	1,6%

a *cutoff value* splits the results into two categories. Researchers often set the cutoff value to 0.5 [Zimmermann and Nagappan 2008]. However, the logistic regression is also robust to data imbalance and this robustness is achieved with setting of cutoff value to optimal value dependent on misclassification costs [Basili et al. 1996]. Our goal is to explore learning performance over different imbalance levels. However, in this study, due to space limitation, we provide preliminary results exploring learning performance stability of standard learning algorithms. Therefore, we provide results of experiments with cutoff value set to 0.5 (that is how standard learning algorithms equally weight misclassification costs). We considered there three different models (with forward feature selection, backward feature selection and without feature selection) and for each of these models, the coefficients are calculated separately.

(b) For all validation samples from step (2b) we count the TN, TP, FN and FP scores of the corresponding model, and calculate the learning performance evaluation metrics ACC, TPR (Recall), AUC and Precision using formulas in Table I.

(4) We made a statistical analysis of the behavior of evaluation metrics measured in step (3b) between different categories introduced in step (2c). Since the samples are not normally distributed, we used the non-parametric tests. The Kruskal-Wallis test showed for all metrics that the values depend on the category. To explore the differences further, we applied multiple comparison test. It reveals that all considered evaluation metric become unstable at the level of imbalance of 80%. According to the theory explained in section 2, we expect that we will get significantly different mean values for all metrics in category of highest data imbalance (90% - 100%).

4. DICUSSION

Data imbalance problem has been widely investigated and there were numerous approaches studying its effects aiming to propose a general solution to that problem. However, from the experiments in machine learning theory it becomes obvious that this is not only related to proportion of minority over majority class but there are also other influences present in complex datasets. As the datasets in software defect prediction (SDP) research area are usually extremely complex, there is a huge unexplored area of research related to applicability of these techniques in relation to the level of data imbalance. That is exactly our main motivation for this work.

There are many approaches, depending on particular dataset, to SDP and development of the learning model. Since we are interested in the performance stability of machine learners over SDP datasets, we should rigorously explore the strengths and limitations of these approaches in relation to the level of data imbalance. Therefore, we present an exploratory research strategy and an example of a case study performed according to this strategy. Although, we use our experiment to eliminate as much as possible inconsistencies and threats of applying the strategy, there is still place for improvement.

In our case study we present how performance stability is significantly degraded at a higher level of imbalance. This confirms the results obtained by other researchers using different approaches. That conclusion have proved reliability of our strategy. Moreover, with the help of our research strategy we confirmed that feature selection becomes instable with higher data imbalance. We have also observed that the feature selection is consistent across levels of imbalance for some features.

Future work should involve extensive exploration of SDP datasets with the proposed strategy. Our vision is that at the end we can gain deeper knowledge about imbalanced data in SDP and applicability of techniques in different levels of imbalance. Finally, we would like to categorize datasets using the proposed strategy and results of this exhaustive research that would serve as a guideline for practitioners while developing software defect prediction model.

REFERENCES

C. Andersson and P. Runeson. A replicated quantitative analysis of fault distributions in complex software systems. *IEEE Trans. Softw. Eng.*, 33(5):273–286, May 2007.

A. Andrews and C. Stringfellow. Quantitative analysis of development defects to guide testing: A case study. *Software Quality Control*, 9:195–214, November 2001.

D. Banthia and A. Gupta. Investigating fault prediction capabilities of five prediction models for software quality. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1259–1261, New York, NY, USA, 2012. ACM.

V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Software Engineering*, 22(10):751–761, October 1996.

G. E. A. P. A. Batista, R. C. Prati, and M.C. Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explor. Newsl.*, 6(1):20–29, 2004.

L. C. Briand, J. W. Daly, V. Porter, and J. Wüst. A comprehensive empirical validation of product measures for object-oriented systems, 1998.

L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter. Exploring the relationship between design measures and software quality in object-oriented systems. *J. Syst. Softw.*, 51:245–273, May 2000.

A. Brooks. Meta Analysis–A Silver Bullet for Meta-Analysts. *Empirical Softw. Engg.*, 2(4):333–338, 1997.

T. Fawcett. An introduction to ROC analysis. *Pattern Recogn. Lett.*, 27(8):861–874, Aug. 2006.

N. E. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Trans. Softw. Eng.*, 26(8):797–814, Aug. 2000.

K. Gao and T. M. Khoshgoftaar. Software defect prediction for high-dimensional and class-imbalanced data. In *SEKE*, pages 89–94. Knowledge Systems Institute Graduate School, 2011.

E. Giger, M. Pinzger, and H. C. Gall. Comparing fine-grained source code changes and code churn for bug prediction. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 83–92, New York, NY, USA, 2011. ACM.

D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. The misuse of the nasa metrics data program data sets for automated software defect prediction. *Processing*, pages 96–103, 2011.

D. Gray, D. Bowes, N. Davey, Y. Sun and B. Christianson. Reflections on the NASA MDP data sets. *IET Software*, pages 549 5583, 2012.

T. Galinac Grbac, P. Runeson, and D. Huljenic. A second replicated quantitative analysis of fault distributions in complex software systems. *IEEE Transactions on Software Engineering*, 39(4):462–476, 2013.

T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *Software Engineering, IEEE Transactions on*, 38(6):1276–1304, 2012.

J. Han and M. Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

H. He and E. A. Garcia. Learning from Imbalanced Data. *IEEE Trans. Knowledge and Data Engineering*, 21(9):1263-1284, 2009.

J. Hulse, T. Khoshgoftaar, A. Napolitano. Experimental perspectives on learning from imbalanced data. In *in Proc. 24th international conference on Machine learning (ICML '07)*, pages 935–942. 2007.

Y. Jiang, B. Cukic, and Y. Ma. Techniques for evaluating fault prediction models. *Empirical Softw. Engg.*, 13:561–595, October 2008.

Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, K. Matsumoto. The Effects of Over and Under Sampling on Fault-prone Module Detection. In *in Proc. ESEM 2007, First International Symposium on Empirical Software Engineering and Measurement*, pages 196–201. IEEE Computer Society Press, 2007.

I. Kaur and A. Kaur. Empirical study of software quality estimation. In *Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology*, CCSEIT '12, pages 694–700, New York, NY, USA, 2012. ACM.

T. M. Khoshgoftaar, E. B. Allen, R. Halstead, and G. P. Trio. Detection of fault-prone software modules during a spiral life cycle. In *Proceedings of the 1996 International Conference on Software Maintenance*, ICSM '96, pages 69–76, Washington, DC, USA, 1996. IEEE Computer Society.

T. M. Khoshgoftaar and N. Seliya. Comparative assessment of software quality classification techniques: An empirical case study. *Empirical Softw. Engg.*, 9(3):229–257, Sept. 2004.

T. M. Khoshgoftaar, N. Seliya, K. Gao. Detecting noisy instances with the rule-based classification model. *Intell. Data Anal.*, 9(4):347–364, 2005.

T. M. Khoshgoftaar, K. Gao, N. Seliya. Attribute Selection and Imbalanced Data: Problems in Software Defect Prediction *In Proceedings: the 22nd IEEE International Conference on Tools with Artificial Intelligence*, 137-144, 2010.

H. Liu, L. Yu. Toward Integrating Feature Selection Algorithms for Classification and Clustering. *IEEE Trans. on Knowl. and Data Eng.*, 17(4):491–502, 2005.

S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: a proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, 2008.

G. Mausa, T. Galinac Grbac, and B. Basic. Multivariate logistic regression prediction of fault-proneness in software modules. In *MIPRO, 2012 Proceedings of the 35th International Convention*, pages 698–703, 2012.

T.J. McCabe. 1976. A complexity measure. *IEEE Transactions on Software Engineering*, 2:308–320, 1976.

N. Ohlsson, M. Zhao, and M. Helander. Application of multivariate analysis for software fault prediction. *Software Quality Control*, 7:51–66, May 1998.

F. Provost. Machine Learning from Imbalanced Data Sets 101. In *Proc. Learning from Imbalanced Data Sets: Papers from the Am. Assoc. for Artificial Intelligence Workshop*, Technical Report WS-00-05, 2000.

S. J. Raudys, A. K. Jain. Small Sample Size Effects in Statistical Pattern Recognition: Recommendations for Practitioners. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(3):252–264, May 1991.

P. Runeson, M. C. Ohlsson, and C. Wohlin. A classification scheme for studies on fault-prone components. In *Proceedings of the Third International Conference on Product Focused Software Process Improvement*, PROFES '01, pages 341–355, London, UK, 2001. Springer-Verlag.

M. Shepperd and G. Kadoda. Comparing software prediction techniques using simulation. *IEEE Trans. Softw. Eng.*, 27(11):1014–1022, Nov. 2001.

M. Shepperd, Q. Song, Z. Sun, C. Mair Data Quality: Some Comments on the NASA Software Defect Data Sets. *IEEE Trans. Softw. Eng.*, http://doi.ieeecomputersociety.org/10.1109/TSE.2013.11, Nov. 2013.

H. Wang, T. M. Khoshgoftaar, and A. Napolitano. An Empirical Study on the Stability of Feature Selection for Imbalanced Software Engineering Data. *In Proceedings of the 2012 11th International Conference on Machine Learning and Applications - Volume 01*, ICMLA '12, pages 317–323, Washington, DC, USA, 317–323.

S. Wang and X. Yao. Using Class Imbalance Learning for Software Defect Prediction. *IEEE Transactions on Reliability*, 62(2):434 - 443, 2012.

G.M. Weiss. Mining with rarity: a unifying framework. In *SIGKDD Explor. Newsl.*, 6(1):7–19, 2004.

T. Zimmermann and N. Nagappan. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 531–540, New York, NY, USA, 2008. ACM.