Cost-sensitive Strategies for Data Imbalance in Bug Severity Classification: Experimental Results

Nivir Kanti Singha Roy
Ericsson AB
417 56, Gothenburg, Sweden
nivir.kanti.singha.roy@ericsson.com

Bruno Rossi
Faculty of Informatics
Masaryk University, Brno, Czech Republic
brossi@mail.muni.cz

***Abstract*—*Context:* Software Bug Severity Classification can help to improve the software bug triaging process. However, severity levels present a high-level of data imbalance that needs to be taken into account. *Aim:* We investigate cost-sensitive strategies in multi-class bug severity classification to counteract data imbalance. *Method:* We transform datasets from three severity classification papers to a common format, totaling 17 projects. We test different cost sensitive strategies to penalize majority classes. We adopt a Support Vector Machine (SVM) classifier that we also compare to a baseline "majority class" classifier. *Results:* A model weighting classes based on the inverse of instance frequencies yields a statistically significant improvement (low effect size) over the standard unweighted SVM model in the assembled dataset. *Conclusions:* Data imbalance should be taken more into consideration in future severity classification research papers.**

***Keywords*-Software Bug Severity; Supervised Classification; Data Imbalance.**

I. Introduction

A bug (issue) report contains the natural language description of a problem or enhancement for a system under development, plus more structured information, such as the assignee, the reporter, expected time of resolution, severity level, and other set of features useful to characterize the issue. Depending on the development model in use, issue trackers may replace completely requirements documents, so that all requirements pass through the issue tracker. Bug reporting systems are very important in the current software development context, and their importance has risen with the application of more collaborative development methodologies for software development, for example *pull request* mechanisms that are supported by modern issue trackers.

It is not surprising that issue trackers constitute a central point of focus in current software engineering empirical research (e.g., [1], [2]). In this paper, we deal with the classification of severity of bug reports—that is providing evaluation of models to classify issues into severity levels, based on different set of features considered. The whole area derived from the research from Menzies and Marcus published in 2008, when a model based on a rule learner supported by entropy and information gain was used to classify severity levels for NASA projects [3].

There is, however, one pitfall in the classification of bug severity (e.g., [4]): imbalance across different severity levels can bring issues for correctly classifying new instances. It is not unusual in the area to consider only some of the classes available, or to report very different classification performance results for the different classes [4]. Furthermore, data imbalance has already been found to impact the results of software defect prediction performance, and needs to be taken into account [5].

In the current paper, we are looking at ways in which we can improve classification results based on such initial unbalanced situations. The contribution of the current paper is twofold:

- provide an aggregated dataset from some of the most relevant previous papers. Namely, we converted datasets in [1], [6], [3] to a common format helpful for uniform classification and future comparison[1];
- evaluation of different cost-sensitive strategies to hamper data imbalance;

The paper is structured as follows: Section II provides the related research. Section III discusses the proposed empirical approach to deal with data imbalance. Section IV presents the data analysis results and Section V provides the conclusions.

II. Related Research

Severity Classification. Over the years, a large amount of empirical knowledge has been derived in the area of severity classification. What we know so far in the area is that, in general, the Multimodal Naïve Bayes (MNB) and Support Vector Machines (SVM) classification models outperform the Naïve Bayes (NB) model and the k-Nearest Neighbour (k-NN) classification models in most of the studies (e.g. [7]). Most of the papers consider NASA PITS data from PROMISE repository, as it was used in the seminal paper from from Menzies and Marcus (2008) [3]. Among open source projects, majority of articles use Firefox and Mozilla data (see [7], [6]).

The models are generally based on textual analysis of the issues (e.g., [1]), however it is recent evolution to consider more semantically-aware models [4], [8]—suggesting to use topic models to reduce the number of features for the classifier to improve the classification performance. However, there are also studies that consider other aspects for severity, for example stack traces, reports length, attachments, and steps to reproduce [9].

[1]dataset is available at www.unlimited2.com/pages/SEAA2017.html

There is agreement on the usage of feature selection to improve the results, as confirmed in [3], [10], [6]. The traditional measures of precision, recall and f-measure are the mostly used [3], with the introduction of the Area Under the Curve (AUC) and ROC curves in case of binary classification ([1], [6], [11], [9]). Even though severity classification is potentially a multi-class classification problem, many papers consider binary classification, mapping the issues in severe/non-severe (e.g., [1], [7], [6]).

Dataset Imbalance. Severity classification is not only dealing with a multi-class problem but also in many cases with learning from unbalanced data (in most cases the *"normal"* severity level). When a class is over-represented in the dataset, we are usually referring to the problem of data imbalance—the presence of instances from one class several orders of magnitude more numerous than other classes.
Several approaches have been proposed for dealing with data imbalance. However, they go essentially in two directions [12]:

- *Sampling methods*: applying both over-sampling and/or under-sampling to balance the datasets might improve the results for some classifiers [12]. Over-sampling brings the problem of synthetically generating new data. Furthermore, sampling might be completely random or use some supervised classification mechanism to improve retention of valuable information. Examples of such algorithms use the k-Nearest Neighbour to find the nearest samples in the minority classes to create new instances based on the space in-between or use adaptive methods to oversample most difficult classes to learn [13];
- *Cost sensitive methods*: these methods start from the idea of associating different costs to the misclassification of different classes [12]. In a multi-class problem, this means associating a cost-matrix to the classifier that can weight the misclassification in different ways, according to the domain understanding. Intuitively, we might want to set the cost of misclassification of the minority classes (e.g. a *critical issue*) to higher level than a majority class (e.g. a *normal issue*).

Only in recent years, the data imbalance problem has captured the interest of researchers in the area of severity classification. As such, there are not many comparisons available. An oversampling strategy generating synthetic elements from the minority class was combined with an unsupervised method to derive levels for unlabelled data [14]. The approach has been shown to improve severity classification performance on three open source projects, namely Eclipse JDT Core, Eclipse PDE UI and Mylin [14]. Oversampling was also used successfully to determine high-impact bugs on the Ambari, Camel, Derby and Wicket open source software projects [15].

While previous studies focused on oversampling strategies, in the current paper we focus on a cost sensitive method, that is approaching different weights to the majority class in order to penalize it in favour of other classes.

III. Approach

Our approach to data imbalance follows through the following steps:

- all datasets in [1], [6], [3] are transformed to a common format for data analysis. Overall, 17 projects were considered in the analysis: PITS A-F, Eclipse Platform / PDE / JDT CDT, Mozilla Thunderbird / Firefox / Core / Bugzilla and MongoDB, Hibernate and Spring that were mined from Jira repositories;
- run the data pre-processing steps in terms of tokenization, stop-words removal and HTML tags removal and the creation of the tf-idf representation for textual features;
- run the SVM classifier with 10-fold cross-validation, comparing results versus the baseline classifier (always predicting the majority class);
- given a training set $t = \{(x_i, y_i)\}$ where x_i is a vector of all features for sample i, and y_i is the associated label, we define d_m as the number of elements subset of t representing the majority class. We define a class imbalance indicator as $imb = d_m/|t|$, ranging from 0.0 to 1.0: the nearer to 1.0 the greater the imbalance. Since we are using such index to reduce the weight on the majority classes, we define $wt = (1 - imb) * \alpha$ as the weight for the majority class, where α is a multiplier that is tested experimentally;
- we compare three strategies for the SVM classifier: i) SVM(1), SVM with no weighting (standard classifier with a linear kernel), ii) SVM(2), weighting elements inversely proportional to the frequency in the training set, and iii) SVM(3) weighting the majority class according to the level of imbalance, using wt, exploring the best α parameter;

In this paper we adopt the following practices.
Models. We use the SVM model as it has been found among the best models so far. We also compare against the baseline (majority class) model;
Features. We consider only textual features, with no usage of feature selection—the application can consistently improve the results as proved in previous research [6], however in this paper we want to focus on the standard models;
Multi-class classification. We consider multi-class classification for levels and not binarization in severe vs non-severe.
Cross-fold validation. We use standard 10-fold cross-validation;
Performance Measures. We use precision (p), recall (r), f-measure, defined in terms of True Positives (TP), False Positives (FP) and False Negatives (FN).
p=$\frac{TP}{TP+FP}$, agreement of the classification of positive labels within the dataset;
r=$\frac{TP}{TP+FN}$, effectiveness in the identification of positive classes;
f-measure=$\frac{2*p*r}{p+r}$, precision and recall into one indicator.

For multi-class classification problems, there is an additional consideration about averaging of scores. We used a weighted average, in which the average value is calculated by the values

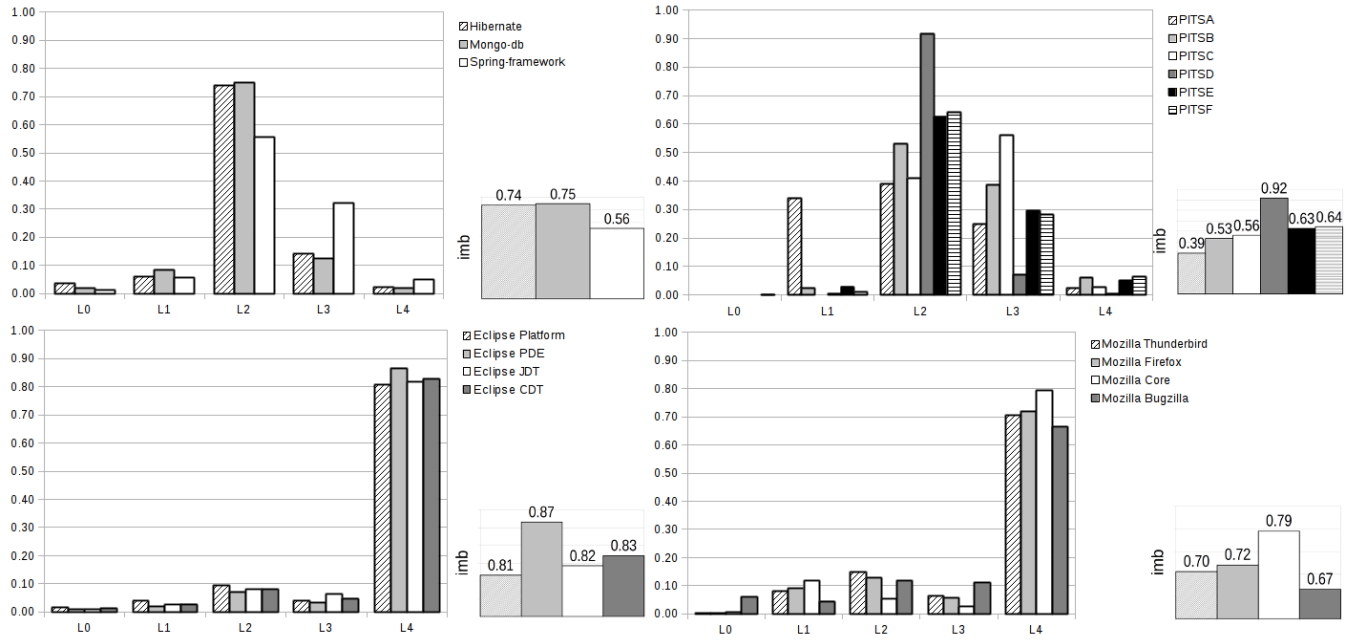

Fig. 1. Data distribution per severity level for part of the projects considered in the analysis. Hibernate, Mongo-db, Spring-framework: L0=blocker, L1=critical, L2=major, L3=minor, L4=trivial. PITS: L0: S1 - highest severity level (prevents execution of operation), L1: S2 - Impacts negatively on the required operation - no work-around available, L2: As L1/S2, but a work-around is available, L3: S4 - inconvenience but no effect on essential capabilities, L4= S5 - any other issue. Eclipse and Mozilla projects: L0=Blocker, L1=critical, L2=major, L3=minor, L4=normal, L5=enhancement.

for each class, computing the average but weighted by the support—the number of true instances for each class. This allows to take class imbalance into account, to a certain degree.

IV. Data Analysis

In the composed dataset, every project (Table I) has a different number of categories for severity, very often dependent on the issue tracking system used (Figure 1). We report also the data imbalance index between different classes, with higher numbers meaning more imbalance of the majority class. For Eclipse and Mozilla, L4 gathers up to 80% of the tickets. We run a *proportion test*[2] of each project against the distribution of tickets in their group and we could reject the *null hypothesis* that the projects come from different data distributions (*p-value $<$ 2.2e-16 two-tailed, 0.05 level of significance*)—with the exception of the PITS D project that shows more imbalance than the other projects in the PITS group.

We followed the approach defined in Section III. For all projects, we run a series of pre-processing steps in terms of stop-word removal and HTML tags removal.

As a first step, we compared all the models with the baseline classifier, that is one classifier that would always suggest the majority class. This comparison showed that in general results are not really considerably better using SVM in the cases of Mozilla, Eclipse, Hibernate, Mongo-db cases. The classifier would be very often prone to suggest the most frequent class based on the unbalanced training set.

Using a classification model in case of highly unbalanced data might bring results very close to the baseline classifier.

[2]https://stat.ethz.ch/R-manual/R-patched/library/stats/html/prop.test.html

TABLE I
Projects information.

Project	Source	Issues	Project	Source	Issues
Hibernate	NEW	4 892	Eclipse Platform	[1,6]	24 115
Mongo-db	NEW	8 800	Eclipse PDE	[1,6]	5 547
Spring	NEW	4 818	Eclipse JDT	[1,6]	10 472
PITS A	[3]	961	Eclipse CDT	[1,6]	5 543
PITS B	[3]	987	Mozilla Thund.	[1,6]	18 750
PITS C	[3]	323	Mozilla Firefox	[1,6]	68 077
PITS D	[3]	182	Mozilla Core	[1,6]	73 373
PITS E	[3]	825	Mozilla Bugzilla	[1,6]	4 324
PITS F	[3]	744			

With the analyzed datasets, for Hibernate and Mongo-db the performance of the standard SVM model, SVM(1), is equivalent to the baseline classifier. Even though results might be useful in final terms of f-measure, it is clear that such classification is not useful to final users, as precision, recall for other classes might be always zeroed. We can think as a final model that always predicts correctly *"normal"* but never *"high"* severity levels. Such model would not be very useful, although reaching good classification performance metrics.

We compared two strategies for cost sensitive balancing of the majority class using the SVM model. The first model, *SVM(1)*, is the unweighted one that is not taking into consideration costs for misclassification. This model is usually outperformed by *SVM(2)*, the model that weights classes inversely proportional to the frequency in the training set.

The *SVM(3)* model looks at the best α parameter that can better optimize wt in terms of f-measure, considering the majority classes. For this reason, we move around the f-

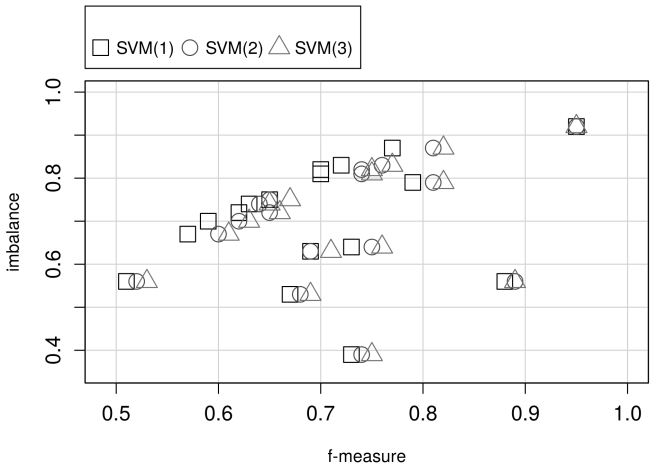

Fig. 2. Imbalance vs f-measure for the three models applied to projects.

measure curve to find the best α value that optimizes the f-measure criteria. In general, results by using *SVM(3)* improve over the other models (Fig. 2). To test the differences, we run t-tests for independent samples (*N=17*):

- SVM(2) provides improvements over the SVM(1) model that are statistically significant (*t-test, p-value 0.0040 $<$ 0.05 significance level, effect size Cohen's d: 0.15*);
- SVM(3) also provides improvements over the SVM(1) model statistically significant (*t-test, p-value 0.0006 $<$ 0.05 significance level, effect size Cohen's d: 0.18*);
- The SVM(3) model - while improving over the SVM(2) model - does not however bring improvements that are statistically significant (*t-test, p-value 0.1022 $>$ 0.05 significance level, effect size Cohen's d: 0.02*);

Summarizing, both SVM(2) and SVM(3) improve over the standard SVM model used. Effect size is small (although slightly lower than the *0.2 cut-off* of Cohen's *d* to define a "small" effect size). Besides these results, the SVM(3) model also requires fine-tuning of parameters that translates in more time needed to find the best models.

V. Conclusions

In this paper, we dealt with bug severity classification, a relevant problem in software engineering to predict the most appropriate severity level of an issue that can be useful for activities such as bug triaging [2]. We focused on data imbalance of classes, typical in this domain, and on different cost-sensitive strategies not considered in previous research. To allow for comparability of next studies, we built a dataset that will be openly available based on previous research papers ([1], [6], [3]). Based on the dataset, we evaluated different cost-sensitive models—with some mechanism to penalize the most frequent class. After the identification of the best model (a linear kernel SVM), we experimentally tested several strategies related to i) weighting of classes depending on the frequencies of classes in training the model, ii) weighting the majority class according to fine-tuning an α parameter. The final conclusion is that the model weighting based on all the class frequencies yields a statistically significant improvement over the standard unweighted SVM model. On the other side, the model weighting the majority class by some form of parameter tuning yields the best results over the standard SVM model, but it does not bring statistically significant improvements over the frequency weighted model. The effect size is small when there are significant results, indicating that the effect is minimal, although present.

We are currently looking into the evaluation of several more advanced cost sensitive methods, to provide an overview of the best approaches to hamper data imbalance.

References

[1] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 1–10.

[2] V. Dedík and B. Rossi, "Automated bug triaging in an industrial context," in *Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on*. IEEE, 2016, pp. 363–367.

[3] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, 2008, pp. 346–355.

[4] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empirical Software Engineering*, pp. 1–30, Aug. 2014.

[5] Z. Mahmood, D. Bowes, P. C. Lane, and T. Hall, "What is the impact of imbalance on software defect prediction performance?" in *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, 2015, p. 4.

[6] N. Singha Roy and B. Rossi, "Towards an Improvement of Bug Severity Classification," in *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, Aug. 2014, pp. 269–276.

[7] A. Lamkanfi, S. Demeyer, Q. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *2011 15th European Conference on Software Maintenance and Reengineering (CSMR)*, 2011, pp. 249–258.

[8] T. Zhang, G. Yang, B. Lee, and A. T. Chan, "Predicting severity of bug report by mining bug repository with concept profile," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 1553–1558.

[9] C.-Z. Yang, K.-Y. Chen, W.-C. Kao, and C.-C. Yang, "Improving severity prediction on software bug reports using quality indicators," in *2014 5th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Jun. 2014, pp. 216–219.

[10] C.-Z. Yang, C.-C. Hou, W.-C. Kao, and I.-X. Chen, "An empirical study on improving severity prediction of defect reports using feature selection," in *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, vol. 1, 2012, pp. 240–249.

[11] R. Jindal, R. Malhotra, and A. Jain, "Analysis of Software Project Reports for Defect Prediction Using KNN," *Lecture Notes in Engineering and Computer Science*, vol. 2211, no. 1, pp. 180–185, Jul. 2014. [Online]. Available: https://doaj.org

[12] H. He, E. Garcia *et al.*, "Learning from imbalanced data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 9, pp. 1263–1284, 2009.

[13] H. He, Y. Bai, E. Garcia, S. Li *et al.*, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE, 2008, pp. 1322–1328.

[14] T. Choeikiwong and P. Vateekul, "Improve accuracy of defect severity categorization using semi-supervised approach on imbalanced data sets," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, 2016.

[15] X.-L. Yang, D. Lo, X. Xia, Q. Huang, and J.-L. Sun, "High-impact bug report identification with imbalanced learning strategies," *J. Comput. Sci. & Technol*, vol. 32, no. 1, 2017.