An Experimental Evaluation of Imbalanced Learning and Time-Series Validation in the Context of CI/CD Prediction

Bohan Liu*, He Zhang*, Lanxin Yang*, Liming Dong*, Haifeng Shen**, Kaiwen Song*
*State Key Laboratory of Novel Software Technology, Software Institute, Nanjing University, Nanjing, Jiangsu, China
Email: dg1732003@smail.nju.edu.cn, hezhang@nju.edu.cn, {dg1932007; dg1832001; mf1932152}@smail.nju.edu.cn
**Australian Catholic University, Sydney, New South Wales, Australia
Email: Haifeng.Shen@acu.edu.au

ABSTRACT

Background: Machine Learning (ML) has been widely used as a powerful tool to support Software Engineering (SE). The fundamental assumptions of data characteristics required for specific ML methods have to be carefully considered prior to their applications in SE. Within the context of Continuous Integration (CI) and Continuous Deployment (CD) practices, there are two vital characteristics of data prone to be violated in SE research. First, the logs generated during CI/CD for training are imbalanced data, which is contrary to the principles of common balanced classifiers; second, these logs are also time-series data, which violates the assumption of cross-validation. ***Objective***: We aim to systematically study the two data characteristics and further provide a comprehensive evaluation for predictive CI/CD with the data from real projects. ***Method***: We conduct an experimental study that evaluates 67 CI/CD predictive models using both cross-validation and time-series-validation. ***Results***: Our evaluation shows that cross-validation makes the evaluation of the models optimistic in most cases, there are a few counter-examples as well. The performance of the top 10 imbalanced models are better than the balanced models in the predictions of *failed* builds, even for balanced data. The degree of data imbalance has a negative impact on prediction performance. ***Conclusion***: In research and practice, the assumptions of the various ML methods should be seriously considered for the validity of research. Even if it is used to compare the relative performance of models, cross-validation may not be applicable to the problems with time-series features. The research community need to revisit the evaluation results reported in some existing research.

CCS CONCEPTS

• **Software and its engineering** → *Software verification and validation*; • **Computing methodologies** → *Machine learning algorithms*; **Cross-validation**.

KEYWORDS

continuous integration, continuous deployment, time-series-validation, cross-validation, imbalanced learning

EASE 2020, April 15–17, 2020, Trondheim, Norway

ACM ISBN 978-1-4503-7731-7/20/04…$15.00
https://doi.org/10.1145/3383219.3383222

ACM Reference Format:
Bohan Liu*, He Zhang*, Lanxin Yang*, Liming Dong*, Haifeng Shen**, Kaiwen Song*. 2020. An Experimental Evaluation of Imbalanced Learning and Time-Series Validation in the Context of CI/CD Prediction. In *Evaluation and Assessment in Software Engineering (EASE 2020), April 15–17, 2020, Trondheim, Norway*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3383219.3383222

1 INTRODUCTION

With the development of software, process data is recorded in the software repositories in the form of logs. The mining of repositories is a trending topic and there are plenty of research using machine learning techniques to support the prediction of software products and processes. However, two crucial data characteristics are often ignored. The data in SE is naturally time-series since the product and process will change with the evolution of software. Both software development activities, the people involved, and the complexity of the artifacts change dynamically over time. Another characteristic is that the prediction of defects, code smell, Continuous Integration and Continuous Deployment (CI/CD) results, etc. are all class imbalance classification problems. Therefore, the data for prediction in SE is commonly the imbalanced time-series data, which may not meet the assumptions of commonly used learning algorithms and validation methods.

To the best of the authors' knowledge, there is not a study that systematically compare imbalanced learning models and balanced learning models with time-series-validation as well as provide a comprehensive comparison between cross-validation and time-series-validation in a class imbalance classification problem in SE. In this study, we aim to take predictive CI/CD, where none of the problems have been addressed, as a case to empirically investigate these two coupled problems so as to fill the gap.

CI/CD enables integration and deployment to be completed early so that integration defects can be detected and fixed timely [17, 51]. However, a high frequency of CI/CD would prolong the software lead time as each CI/CD process is potentially time-consuming. There are two possible outcomes of a CI/CD pipeline: *passed* and *failed*. A *passed* one does not detect any defects and can be safely skipped, whereas a *failed* one is critical as the defects need to be detected and fixed at the earliest time possible. A cost-effective way to shorten software lead time is to skip a CI/CD process if it is going to pass and a possible way to know this without actually doing CI/CD is through prediction. A prediction model, which is trained by historical CI/CD data, forecasts the outcome of the next CI/CD right after each commit and the CI/CD process would only be triggered if the prediction outcome was *failed*.

Several studies [20, 24, 25, 41, 54, 55] used machine learning methods to predict CI/CD outcomes. Common to these studies is the evaluation of balanced classifiers (e.g., C4.5, Naive Bayes) using validation methods that shuffle the data sets. However, the amount of *passed* is often several times that of *failed*. Clearly, *failed* is the minority class and as such it is more difficult to predict *failed* than *passed* [20, 24] because common balanced classifiers are based on the assumption that the number of all classes in the sample set is equal and the importance is treated equally. Furthermore, as logs of software development, the input data is naturally time series. Atchison et al. [2] analyzed 1283 projects from TravisTorrent repository and found that CI/CD presents strong seasonal behavior. Consequently, it is reasonable to question the reliability of their prediction results.

The learning of minority class from imbalanced data is an open challenge in machine learning [29, 53]. There are various methods for solving the imbalance problem, e.g., sampling methods, cost-sensitive methods, ensemble methods, etc. [27]. We conduct an experimental study to evaluate 67 models including 4 balanced models and 63 imbalanced models that consists of five types of imbalanced learning methods discussed in the literature review [27]. The contribution of this study is three-fold: 1) We have conducted an experimental study to compare the time-series-validation and cross-validation under a class imbalance classification context. 2) We systematically compare the imbalanced and balanced learning methods using time-series-validation. 3) We provide a more comprehensive and realistic result for predictive CI/CD and reveal the reason of the phenomenon found in existing studies that the performance of prediction on different projects is quite different.

2 CHARACTERISTICS OF CI/CD DATA

This section presents an investigation of the characteristics of CI/CD data and how these characteristics, i.e. class imbalance and time series, impact prediction models.

2.1 Imbalanced time series data

We analyzed the latest TravisTorrent dataset [9], which contains build logs of 1283 GitHub projects that use the Travis CI service. Figure 1 presents the frequencies of ratios of *failed* to *passed* of projects in TravisTorrent. Among the 1283 projects, 52 projects that have no record of *failed* or *passed* were excluded. It shows that there are few projects with a ratio exceeding 0.5 (16%). In 68% projects, the number of *failed* in them is less than 30% of the number of *passed*. Imbalance is a common phenomenon in TravisTorrent.

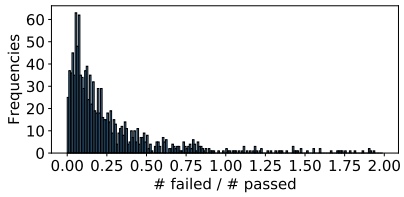


Figure 1: Frequencies of ratios of *failed* vs. *passed* among 1231 projects from TravisTorrent.

Results of the current CI/CD is related to the last one and the time interval between them. This causal relationship was confirmed by existing research that found variables related to the previous CI/CD would affect the predicted results [24, 41]. In addition, new patterns will emerge due to the dynamic changes in complexity and development stages as software evolves. The CI/CD data is clearly time series and the causal relationship has to be preserved when evaluating a prediction model, which means evaluation methods, such as shuffle split or K-fold cross-validation [18], are not applicable. If the causal relationship was not preserved, future patterns would be introduced into the training set, leading to a biased result.

2.2 Imbalance problems in prediction

The imbalance is usually two-fold: 1) the data is imbalanced; 2) we pay more attention to the minority class in practice. This is also true for CI/CD outcome prediction. The purpose of CI/CD is to verify commits and locate the bugs as early as possible. The use of predictive models is to make predictions before executing CI/CD pipeline to reduce some unnecessary builds. The prediction of CI/CD outcome is a binary classification problem and results can be *passed* or *failed*. When a commit is predicted to be *failed*, the developers should execute the CI/CD to validate the result and locate the bug; when a commit is predicted to be *passed*, this build can be skipped. The *failed* result is due to the defects in the commit. The number of *passed* is commonly several times that of *failed* in the CI/CD results of a project. Therefore, it is more difficult to predict *failed*, which is consistent with the results of existing research [20, 24]. On the other hand, if the *failed* results are not effectively identified, the predictive model will cause the fixing of a large number of problematic commits to be postponed. Hence, the recall of *failed* is critical in the evaluation of models.

Class imbalance is a long-standing problem. Studies have shown that some base classifiers (e.g., C4.5, Naive Bayes) perform significantly better on balanced data than on imbalanced data [14, 30, 34]. The reason is that these classifiers treat all categories equally and their learning is aimed at optimizing overall accuracy. When the categories are not balanced, classifiers tend to provide a severely imbalanced degree of the precision, with the majority class having a precision several times than the minority class [27]. However, this is opposite to the purpose of our predictions. Our goal is usually to predict the minority class, for instance, the purpose of defect prediction is to identify the fault-prone modules.

Various imbalanced learning methods were proposed to improve the accuracy of predicting minority class from imbalanced data, which are mainly optimized from two aspects, one is to modify the training set by sampling, and the other is to design the classifier to make it sensitive to the minority class [27].

2.2.1 Sampling algorithms. The use of sampling methods in imbalanced learning is to modify an imbalanced dataset into a balanced dataset. The two common mechanisms of sampling are over-sampling and under-sampling. The former randomly duplicate instances of the minority class and the latter randomly discard instances of the majority class from the original dataset.

With regard to over-sampling, a simple random over-sampling is to duplicate certain instances, which would cause these instances

to become tied, resulting in over-fitting [40]. To overcome this issue, Chawla et al. proposed a Synthetic Minority Oversampling TEchnique (SMOTE), which has been widely adopted at present. The main idea of SMOTE is to generate and interpolate new minority class instances between adjacent pre-existing minority class instances [14]. In addition to this, there are some other improved algorithms designed to mitigate over-fitting.

As to under-sampling, a simple random under-sampling is to randomly remove the samples of majority class until the number of the two categories is balanced. However, the random removing of instances may cause some vital information of majority class to be removed at the same time. In view of this, different improved algorithms were proposed to reduce the loss of valuable information. For instance, EditedNearestNeighbours removes instances that differ from two of its three nearest neighbours.

The combination of under-sampling and over-sampling were also explored. For example, SMOTEENN combines SMOTE and Edited Nearest Neighbours (ENN) [7].

Table 1 presents the sampling algorithms that are considered in this study. We use Imbalanced-learn [35] which is an OSS python toolbox to implement them.

Table 1: Sampling algorithms.

Strategies	Samplers	Tools
Over-sampling	Random Over Sampler (ROS) SMOTE [14] BorderlineSMOTE (BSMOTE) [23] SVMSMOTE [28] ADASYN [26]	Imbalanced-learn
Under-sampling	Random Under Sampler (RUS) One Sided Selection (OSS) [33] Neighbourhood Cleaning Rule (NCR) [34] Near Miss (NM) [56] Instance Hardness Threshold (IHT) [50]	Imbalanced-learn
Combination	SMOTETomek [7] SMOTEENN [8]	Imbalanced-learn

2.2.2 Imbalanced learning classifiers. They are the classifiers that are robust to imbalanced data as well as balanced classifiers assume class balance.

There are various methods, which instead of modifying the original training set, make the classification more sensitive to minority class through special design, so as to improve the accuracy of the classification of minority samples. The four effective mechanisms are cost-sensitive methods, kernel modification methods, one-class learning methods, and ensemble methods.

Cost-sensitive methods use different cost matrices that indicate the costs for misclassifying any particular instances [19]. Kernel modification methods add a trainable post-processing step to map the outputs of predictions into probabilities [42]. One-class learning methods are designed for identifying minority instances by only recognizing one category without distinguishing between different categories [43]. Ensemble methods construct a set of classifiers and then classify instances by taking a vote or weight of the predictions of all the classifiers [16].

Table 2: Learning algorithms.

Strategies	Classifiers	Tools
Cost-sensitive	BayesMinimumRisk [6] Thresholding Optimization (TO) [48] Cost-Sensitive Logistic Regression (CSLR) [3] Cost-Sensitive Decision Tree (CSDT) [5] Cost-Sensitive Random Forest (CSRF) [4] Cost-Sensitive Bagging (CSB) [4] Cost-Sensitive Pasting (CSP) [4] Cost-Sensitive Random Patches (CSRP) [4]	Costcla
Kernel modification	SVC [42]	Scikit-Learn
One-class learning	One Class SVM (OCSVM) [38] Local Outlier Factor (LOF) [13]	Scikit-Learn
Ensemble	Balanced Random Forest (BRF) [15] Easy Ensemble (EE) [36] RUSBoost [46] Balanced Bagging (BB) [37]	Imbalanced-learn
Balanced classifiers	Gaussian Naive Bayes (GNB) CART [12] Gradient Boosting (GB) [21] Random Forest (RF) [11]	Scikit-Learn

We consider 15 imbalanced classifiers as presented in Table 2. We use Costcla[1] to implement cost-sensitive classifiers, Scikit-Learn[2] to implement kernel modification and one-class learning classifiers, Imbalanced-learn to implement ensemble methods.

In addition, four balanced classifiers that performed well in studies [41, 54, 55] are considered, including RF, CART, GNB, and GB. We used Scikit-Learn to implement them. The CART implemented by Scikit-Learn is an optimised version and it is very similar to C4.5. It should be noted that RF and GB are also ensemble methods but they are not imbalanced classifiers without weighting.

2.3 Time series in evaluation

The fundamental assumptions of classical cross-validation techniques such as K-fold and shuffle split are that samples should be independent and identically distributed [1]. However, time series data is characterized by autocorrelation between samples so it does not meet the assumption of using cross-validation.

Figure 2 takes a simple three-fold cross-validation as an example. It compares the results of cross-validation with the time-series-validation. The circle, square, pentagon, and star represent different patterns of samples, respectively. In this time series data, new patterns appear over time. The basic K-fold cross-validation divides the data into K equal folds without disturbing the chronological order, then it iteratively takes one fold as test data each time, and remaining folds as the training data for evaluating the model, and finally average the results of K iterations. In this example, both the recall and precision evaluated using cross-validation are 0.92. In reality, it is impossible to train the model with the data from future to predict what happened in history since the data is generated in a time series order. Time-series-validation simulates the actual validation. Using the test set with the same sample size as in the cross-validation, the model can be iteratively evaluated twice. As a result, the mean values of recall and precision are smaller than the results cross-validation. As to the shuffle split, it completely disrupts the chronological order and we will not repeat it here.

[1] https://github.com/albahnsen/CostSensitiveClassification
[2] https://scikit-learn.org/

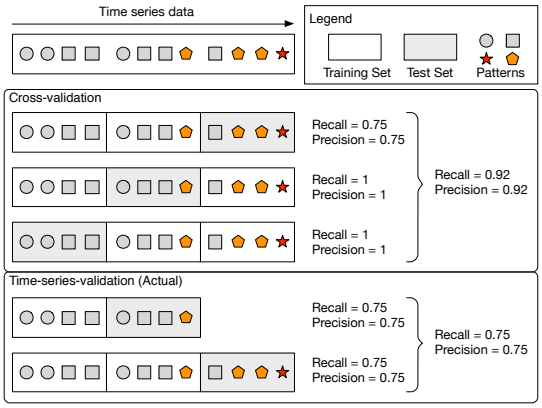


Figure 2: The comparison of prediction performance between cross-validation and actual results.

3 EXPERIMENTAL STUDY

Based on the characteristics of CI/CD data, we explained the problems in training and evaluating predictive models. To study the two problems, i.e. time series and class imbalance, we conduct an experimental study to compare time-series-validation and cross-validation as well as compare imbalance models and balanced models. This study formed a quasi-experiment because we select projects based on a particular criterion to ensure the sample size for model training. We select all the projects that meet the criterion from the two most well-known open-source platform, i.e. Github (TravisTorrent records CI/CD logs of all the projects enabled Travis CI on Github) and Gitlab. From this point, our experimental samples are representative.

3.1 Research questions and hypotheses

We define the following research questions and hypotheses:

RQ1: *Does cross-validation lead to an overly biased evaluation?* RQ1 aims to test our hypothesis that cross-validation would produce biased results since it needs to be based on the assumption that the emergence of new patterns is time-independent. As a more realistic way of validation, time-series-validation cannot be replaced by cross-validation in the evaluation of CI/CD prediction models. We state the null hypothesis as follows:

$H_{0,1}$**:** There is no difference between the results of cross-validation and time-series-validation.

RQ2: *Can imbalanced learning achieve better performance than balanced learning?* In theory, it is more appropriate to use imbalanced learning methods for imbalanced data. RQ2 aims to comprehensively evaluate and compare the performance of various balanced and imbalanced models on predicting CI/CD result. In more detail, RQ2 contains two sub-questions based on the two levels of imbalanced learning. For data level, 1) Can sampling techniques improve the performance of balanced classifiers for imbalanced data? For classifier level, 2) Whether imbalanced classifiers perform better than the balanced classifiers for imbalanced data? The null hypotheses arising from the sub-questions are:

$H_{0,2a}$**:** No sampling algorithm can improve the performance.

$H_{0,2b}$**:** No imbalanced classifiers can achieve better performance than the balanced classifiers.

RQ3: *Does the degree of data imbalance affect the performance of models?* Existing studies have reported that the performance of CI/CD prediction on different projects varies widely. The goal of RQ3 is to investigate whether the degree of imbalance has an impact on model performance, which can be a reference for whether predictions can be applied. We state the null hypothesis as follows:

$H_{0,3}$**:** There is no correlation between the prediction performance and the degree of data imbalance.

3.2 Study design

Driven by the RQs, we conducted a quasi-experiment which is designed as follows.

Data sets. We selected 18 projects from TravisTorrent and Gitlab based on a particular criterion to ensure the sample size (number of CI/CD records) for training. Xia and Li [55] excluded projects with less than 1000 builds for cross-validation. To avoid bias caused by insufficient training data since an n-fold time-series-validation uses only $1/(n+1)$ of the data for training in the first iteration. We excluded projects with less than 4000 builds so that each included project can use at least three-fold time-series-validation with a training set consists of more than 1000 builds. TravisTorrent has 14 projects that meet the criteria. As a supplement, and to increase the universality of the results, we also selected projects from Gitlab. By April 2019, four projects in Gitlab met the criteria. Figure 3 depicts the ratios of *failed* to *passed* for the projects we selected from TravisTorrent and Gitlab respectively. It shows that the number of *passed* is about three times the *failed* on average and only two projects from TravisTorrent have a higher proportion of *failed*. Therefore, most of them are imbalanced data (among the imbalanced data sets in the study [14], the largest proportion is 54%), which is consistent with our findings in Sec. 2.1. We consider data with a proportion less than 0.6 as imbalanced and the two data balanced projects can serve as a reference.

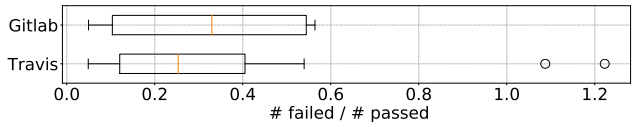


Figure 3: Ratios of *failed* to *passed* of the selected projects.

Treatments. We consider 15 kinds of imbalanced learning algorithms and four balanced classifiers presented in Table 1. As to the four balanced classifiers, we use 12 sampling algorithms (ref. Table 2) combined with each of them, for example, SMOTE+CART denotes the model consists of SMOTE and CART. For comparison, we add a group that does not adopt any sampling algorithm. Therefore, there are a total of 67 models for evaluation, including 4 balanced models, and 63 (15 imbalanced classifiers + 12×4 combinations of sampler+balanced classifier) imbalanced models. We use both time-series-validation and cross-validation to evaluate models. For RQ1, the evaluation using cross-validation is the control group, and the treatment group uses time-series-validation. For RQ2 and RQ3, the control group uses balanced models and the treatment group uses imbalanced models.

3.3 Conduct of the quasi-experiment

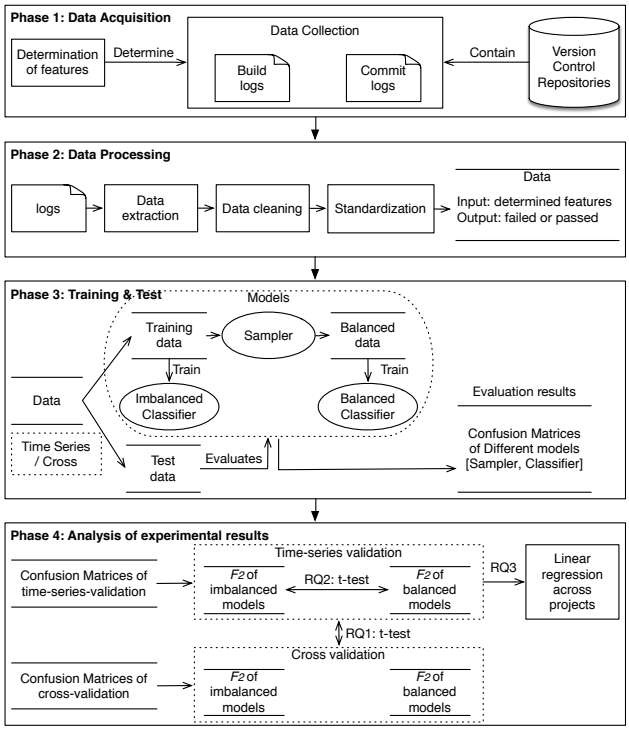


Figure 4: Research framework.

Figure 4 illustrates the research framework of the experimental study, which consists of four phases.

3.3.1 Data acquisition. It follows two steps to collect the raw data. As a supplement, in addition to TravisTorrent, we also considered Gitlab[3]. GitLab is a web-based application for the entire DevOps lifecycle, from project planning and source code management to CI/CD and monitoring. It provides a built-in tool, GitLab CI/CD, which allows users to apply all the continuous methods to their software with no third-party application or integration needed.

Determination of independent variables. We used a total of 24 input variables which are presented and explained in detail in Table 3. Twenty-two of which are borrowed from related studies [24, 41]. We excluded the variables that require the information of subsystem used in the study [24] as it is not available in the selected projects. Besides, we excluded *name of collision developers* since the same developer may use multiple user names, which may introduce a lot of noisy data. To balance the effectiveness of using prediction, we did not consider AST level code change modification data used in the study [25], instead, the *additions_last_build* and *deletions_last_build* that indicate the code changes were used. We conducted pilot experiments and confirmed that eliminating any of the 24 features can make the prediction worse.

[3]https://gitlab.com/

Data collection. We downloaded the latest TravisTorrent dataset[4]. As to Gitlab, we used the API[5] to collect the build logs. The commit logs are also required to generate some variables. We cloned the code and used git log command to obtain the commit logs.

Table 3: Input variables.

Variables	Explanations	Ref.
total_files_pushed	# of files edited in this build.	[24, 41]
avg_file_committed	Avg. # of files edited in this build.	[41]
commit_msg_length	Avg. length of the commit messages of this build.	[41]
time_elapse	The days since the last build sequentially.	[41]
time_last_failed_build	The days since the last failed build.	[24]
additions_last_build	The lines of code added since the last build.	New
deletions_last_build	The lines of code deleted since last build.	New
commit_last_build	# of commits since the last build.	[41]
last_build_result	The result of last build (passed or failed).	[24, 41]
no_src_edited	The XOR between booleans of this and last builds that indicate if there are source code files edited.	[41]
no_config_edited	The XOR between booleans of this and last builds that indicate if there are configuration files edited.	[41]
ame_committer	A boolean to indicate whether the committer is the same as the last build.	[41]
committer_history	The historical fail rate of the pushes by the current committer on this project before this build.	[41]
committer_recent	The result of the last build that triggered by the current committer on this project.	[41]
project_history	The fail rate of all the previous builds in the project.	[41]
project_recent	Similar to project_history but using only last five builds.	[41]
gaussian_threat	Model the build failure of the project as gaussian distribution to measure the threat to current build.	[41]
committer_exp	The number of commits the committer made in the project before this build.	[41]
day_time	Time of Day (0-24).	[24]
weekday	Day of Week (Mon, Tue, Wed, Thu, Fri, Sat, Sun).	[24]
month_day	Day in Month (1-31).	[24]
developer_count	The number of developers who integrated changes to the main branch.	[24]
collision_files	The number of files modified multiple times in parallel since the last certified build.	[24]
collision_developers	The number of developers who modified files in parallel.	[24]

3.3.2 Data processing. It follows three steps to obtain the required data for model input from the collected raw data.

Data extraction. We developed a python program to extract and generate required data of input variables from logs we collected in the previous phase.

Data cleaning. The output variable for training the model is the build result. There are four types of results in TravisTorrent build logs, including *passed, failed, errored, and canceled*, and Gitlab CI/CD contains *passed, failed, skipped, and canceled* instead. We removed logs other than *passed* and *failed* since others mean that the CI/CD was terminated before the scripts were executed, in other words, whether it can pass the test is unknown. There are *passed with warnings* in Gitlab which are caused by the failures of scripts that are allowed to fail. Consistent with [54], we treated them as *passed* since *warnings* are usually not processed further by developers.

Standardization. The values of some numerical variables are fairly large and some even differ by orders of magnitude, for example, there may be more than a hundredfold gap between the total number of commits made by two committers. Therefore, we standardize all the variables to make the mean value of each variable 0 and the standard deviation equals to 1.

[4]https://travistorrent.testroots.org/dumps/travistorrent_8_2_2017.sql.gz
[5]https://docs.gitlab.com.cn/ee/api/pipelines.html

3.3.3 Training & Test. It mainly follows three steps.

Split of Data set. Cross-validation and time-series-validation are used respectively for comparison. It is important to note that the prepared data must be split into the training set and the test set before sampling, otherwise, the test set will be changed by sampling [45]. Time-series-validation simulates the real situation shown in Figure 2, which gradually increases the size of the training set to iteratively evaluate the model as we did in the application scenario. In the first iteration, the first set is the training set and the second set is the test set. In the second iteration, the training set contains both the first and second sets, and the third set is the test set. The rest of the iterations follow the same way. The test set is always the most recent generated data set, which is used to evaluate the model being trained by all the previous data sets. For each iteration, the sample size of the training set is approximately $i * 1000$ samples, where i denotes the ith iteration, and the sample size of the test set is approximately 1000 samples. We did not deliberately make the two validation processes have the same training or testing set as Tantithamthavorn et al. did [52] since whether or not the test set will introduce future data is the major difference between the two methods. In addition, this is more consistent with the validation process in existing studies.

Training. Table 2 presents four categories of imbalanced learning algorithms, i.e. cost-sensitive, kernel modification, one-class learning and ensemble. Besides, four balanced classifiers are considered. These learning algorithms are applied to each training set split in the previous step. For the balanced classifiers, various sampling algorithms presented in Table 1 can be combined with them to form an imbalanced model. Therefore, different sampling algorithms are adopted for the training set, and different training sets will be generated.

Test. The test set is used to test the models trained in the previous step. As a result, the confusion matrices would be generated to indicate the performance of these models.

Evaluation metrics. We take *failed* as positive and *passed* as negative due to the interest of *failed* in practice. Therefore, Precision (P) in this study is the percentage of correct predictions of all the builds that were predicted as *failed* and Recall (R) is the percentage of correctly predicted *failed* to all the true *failed*. P and R are sometimes a pair of contradictory measures. Fβ-measure (F_β) which is the harmonic mean of P and R is commonly used to provide a comprehensive measurement. F_β is measured as follows:

$$F_\beta = \frac{(1+\beta^2)\cdot P \cdot R}{\beta^2 \cdot P + R} \tag{1}$$

where β is a positive parameter to weight the relative importance of the precision over the recall. A greater β value indicates a higher importance of recall over precision. In imbalanced learning, the evaluation of classifiers should focus primarily on the recall of the minority class, and it is a common practice to take β as 2 [32, 39]. In the CI/CD prediction problem, specifically, the cost of missing *failed* is greater than the cost of missing *passed*, since the former means the slip of the defects whilst the latter simply means an extra build. Therefore, we use F_2 to measure the overall performance.

3.3.4 Analysis of experimental results. We evaluate the 67 models using cross-validation and time-series-validation respectively for each project. To answer RQ1, we compare the mean results of cross-validation with those of time-series-validation from both model and project perspectives. To answer RQ2, we only use the results of time-series-validation. We applied different splits to each project respectively so that the size of all test sets can be roughly the same. We used one-tailed t-test to test the hypotheses $H_{0,1}$, $H_{0,2a}$, $H_{0,2b}$ and used linear-regression to test the hypothesis $H_{0,3}$. To compare the two validation methods, we use Relative deviation (R_d) and Absolute deviation (A_d) in terms of models and projects respectively. The R_d and A_d are defined as follows:

$$R_d = \frac{\sum_{i=1}^{n}(c_i - t_i)/t_i}{n}; \qquad A_d = \sum_{i=1}^{n}(c_i - t_i)/n \tag{2}$$

We define R_{dm} and A_{dm} as the R_d and A_d measuring across models, where c_i and t_i denote the average F_2 of model i on all the projects using cross-validation and time-series-validation respectively. To measure the deviation across projects, we define R_{dp} and A_{dp}, where c_i and t_i denote the average F_2 of all the models on project i.

4 RESULTS AND ANALYSIS

This section details and discusses the evaluation results in terms of the RQs. We focus primarily on the predictive performance of *failed* since the overall performance or the performance for majority class would cause the illusion that the model performs well. For example, *failed* accounts for less than 5% in project O, hence, the overall accuracy will reach 0.95 even if all categories are predicted to be *passed*. Due to the space limitation, we share the project list and complete evaluation results at figshare [6].

4.1 Validation methods (RQ1)

Figure 5 depicts the gap of the evaluation results between the two validation methods for each model. The evaluation results are the mean values of F_2 for the 18 projects. Of the 67 models, only for four (6%) models, i.e. OCSVM, NM+GNB, CSLR, and LOF, time-series-validation produced larger F_2. For the remaining 63 models, cross-validation relatively increased F_2 by 14% on average. Specifically, the R_{dm}s ranged from 3% to 40%, of which 18 models have an R_{dm} of less than 10%. Both the high R_{dm}s and A_{dm}s mainly occur in models with poor performance. The R_{dm}s in the four balanced models are about 10%, which is lower than the mean of the imbalanced models. We performed a one-tailed t-test, and the results show that $p < 0.05$, which means F_2 produced by cross-validation is statistically significantly higher than that of time-series-validation (rejecting $H_{0,1}$).

Figure 6 depicts the mean absolute and relative deviations across projects, i.e. A_{dp}s and R_{dp}s. For different projects, the degree of R_{dp}s are obviously different whilst the A_{dp}s are around 4.5%. The reason for the smaller deviation compared with Figure 5 is that there are cases where the cross-validation produced smaller results than time-series-validation. For project H, more than half of models achieved better performance using time-series-validation. For other projects, there are also a few models that get better results using time-series-validation. Each model has situations where it performs better with time-series-validation on some projects, but it has nothing to do with whether it comes from TravisTorrent or

[6] https://doi.org/10.6084/m9.figshare.11317169.v1

Gitlab. However, this is more likely to happen in models such as OCSVM mentioned above. For projects F and Q, there were high R_{dp} due to the fact that several models got a very low F_2 when using time-series-validation, whilst cross-validation did not.

Furthermore, the difference between the results of the two validation strategies are inconsistent across models, which means that cross-validation cannot even be used to compare the performance of the two models relatively.

> For most models, on most CI/CD data, using cross-validation will get more optimistic results. Different from the results found in [31], we found that using future data in training set (cross-validation) also had the potential to make the results worse. In summary, $H_{0,1}$ is rejected and recommend time-series-validation for time-series data.

4.2 Balanced or imbalanced learning (RQ2)

In Figure 5, we highlight balanced models, i.e. GNB, GB, RF, and CART, with red dashed line. There are two main types of imbalanced models, one is the combinations of sampler and balanced classifier, and the other is to directly use imbalanced classifiers. Overall, the performance of balanced models is not even ranked in the top 20% of all models, and the performance of the three models except GNB is at a relatively low level. Among the models we evaluated, the worst 8 models are imbalanced models. Therefore, not all imbalanced models have better prediction performance than balanced models on imbalanced data. However, the proper imbalanced model can obviously achieve better performance.

As to the combinations with samplers, although the F_2 of RF is lower than 0.3, its performance can be improved after combining any sampling algorithms. In all models, eight of the top ten models are the combinations of RF and sampling algorithms, and any sampling algorithm we considered can improve RF. For GB and CART, performance can be improved by combining the appropriate sampling algorithm, otherwise, the performance will be deteriorated. In addition, the combination of SMOTE and RF or GNB achieve a better performance on average, but when the SMOTE is combined with GB or CART, it results in worse performance. We performed a one-tailed t-test to compare the SMOTEENN+RF with the four balanced classifiers for projects, the results show that $p < 0.05$ for each comparison (rejecting $H_{0,2a}$).

For models with imbalanced classifiers, the performance presents polarization as both the best and worst two models are from this type. Among the 15 imbalanced classifiers, nine are superior to all the balanced classifiers, which cover three different learning methods, including all the cost-sensitive algorithms (TO, CSP, CSRP, CSDT, CSB, BMR), a one-class learning algorithms (OCSVM), and an ensemble methods (BB, BRF). We performed a one-tailed t-test to compare the BRF with the four balanced classifiers for projects, the results show that $p < 0.05$ for each comparison (rejecting $H_{0,2b}$).

Considering the different proportions of *failed*/*passed* in different projects, and in order to further compare the performance of balanced and imbalanced learning in different projects, we present the performance of the four balanced models and mean and median F_2 of all imbalanced models in Figure 7. The AVG-TOP10 is better in all the projects except A and F. Even in A and F, the differences from the best-performing model, GNB, were only 0.01. On the other hand,

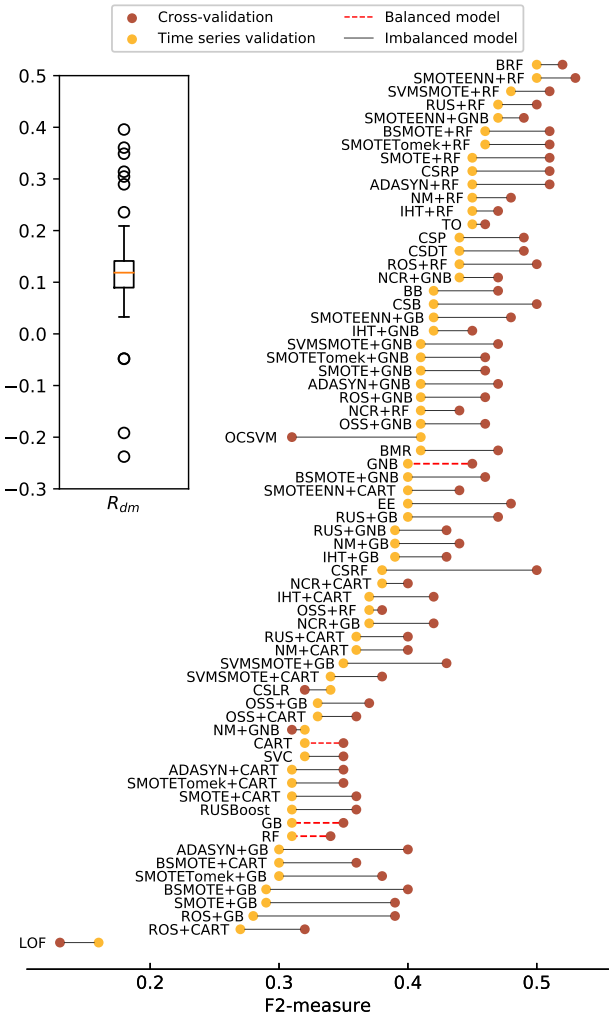


Figure 5: Mean value of F_2 of *failed* for all models validated using cross-validation and time-series-validation respectively (The symbol "+" denotes a combination, e.g., NM+RF denotes the combination of Near Miss and Random Forest); the box-plot presents the distribution of relative deviations of using cross-validation across models (R_{dm}).

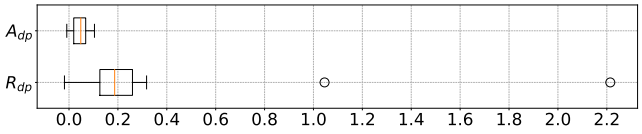


Figure 6: Deviations of all models between cross-validation and time-series-validation across projects.

GNB is not stable as it ranks last in C, G, J and Q. MEDIAN-ALL is better than AVG-ALL as well as AVG-ALL is better than CART, GB, and RF in most projects. In two projects with balanced data, C and E, AVG-TOP10 is still significantly better than the balanced models. However, GB, RF, and CART are better than MEDIAN-ALL

and AVG-ALL in C; RF and GNB are better than MEDIAN-ALL and AVG-ALL in E. It means that most imbalanced models have an advantage for imbalanced data, and even for balanced data, the best imbalanced models still perform better. It is due to the explicit need to focus on minority classes in the imbalance problem.

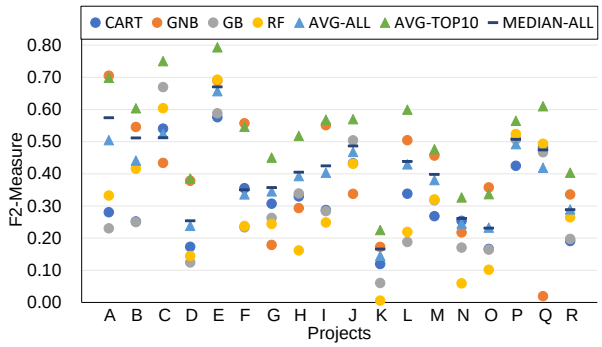


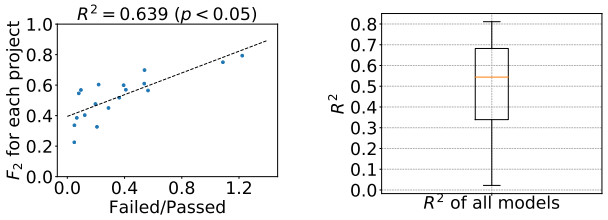
Figure 7: F_2 of models for each project. AVG-ALL and MEDIAN-ALL denote the mean and median value of all the imbalanced models respectively; AVG-TOP10 denotes the mean value of the ten best performing imbalanced models.

For imbalanced data, imbalanced learning is not a panacea, but an appropriate imbalanced model can obviously achieve better performance than the balanced model. Whether it is effective does not lie in the type of imbalance learning methods, but in the specific algorithm. Even with balanced data, imbalanced models perform better when we focus on a specific class (i.e. *failed* in CI/CD). In summary, $H_{0,2a}$ and $H_{0,2b}$ are rejected.

4.3 Impact of imbalance ratio (RQ3)

Figure 7 shows the AVG-TOP10 is better in C and E. In order to further explore whether the degree of imbalance has an impact on model performance, we performed linear regressions between F_2 of each model and imbalance ratio (*failed*/*passed*) for each project. As shown in Figure 8 (a), an R^2 of 0.64 indicates that the lower the degree of imbalance, the better the prediction performance for AVG-TOP10. Figure 8 (b) depicts the distribution of R^2 for each model. It shows the correlation is significant for most models with a mean R^2 of 0.49 (rejecting $H_{0,3}$). For 12 models, we cannot reject $H_{0,3}$ (p>=0.05), one of which is LOF, and the rest are the combinations with GNB. This may be due to the low sensitivity of GNB to data characteristics. NM is the only sampling algorithm that makes GNB sensitive to the degree of imbalance, but it is an algorithm that makes GNB worse as shown in Figure 5.

The degree of data imbalance has a significant negative impact on prediction performance for most models. In summary, for 55 models, $H_{0,3}$ is rejected; and we cannot reject $H_{0,3}$ for 12 models. When the data is excessively imbalanced, that is, when the proportion of *failed* is extremely small, no imbalance model is a silver bullet.



(a) AVG-TOP10 (b) Distribution of all models

Figure 8: Linear regression fitness (R^2) between F_2 and *failed*/*passed* for each project.

4.4 Implications and limitations

Our findings are not only limited to the prediction of CI/CD, but also has enlightenment and reference value for similar problems involving machine learning in SE research. We should be more cautious about the assumptions of the methods used.

For the studies that apply the classical cross-validation to evaluate predictive models training with time series data, we should revisit their results. Our study presents a more realistic evaluation of the CI/CD predictions.

In similar prediction problems, e.g., prediction of code smells, it is necessary to consider the imbalanced learning for better performance. When evaluating the performance of the model on different projects, the degree of class imbalance in the project should be taken into account. The appropriate F_β is recommended to be considered according to the interest of specific category in the specific scene, rather than blindly taking β as 1.

The top ten models for the projects we evaluated were different. Designing a scene-driven model selection algorithm may be more effective than improving a single model. It is also critical to investigate how much more effective it is to use predictive models than the CI/CD without prediction.

One threat to construct validity is that we only used F_2 to measure model performance. Although F-measure is a comprehensive metric that was widely used in class imbalance problems, it is worth studying whether other metrics would produce different results.

Randomization was not possible when we selected projects and algorithms. To mitigate this threat to internal validity, we select a considerable number of algorithms, which cover main types of sampling and imbalanced learning techniques.

We did not discuss the cross-project prediction although there are studies on it in related work. Cross-project prediction is beyond the scope of our study since it is based on different assumptions that the patterns of any project can be obtained from other projects. There are more similar problems in SE that is urgent to explore from an empirical perspective. It is worth reviewing the rationality of the application of machine learning in SE.

5 RELATED WORK

5.1 Imbalanced learning and time-series

Before 2012, many studies were not aware of the data imbalance in defect prediction [22]. Since then this problem was addressed by several studies using different methods such as sampling [10, 47] and weighting [44, 49].

Table 4: The summary of the performance of prediction in existing research.

Study	Data sets	Algorithms	Validation	Results*
[24]	IBM Toronto Labs	C4.5	Shuffle split	$R = 0.69$, $R^{'} = 0.95$
[54]	IBM Jazz^{TM}	Social network, Naive Bayes	Leave-one-out	$R \in [0.55, 0.75]$, $P \in [0.50, 0.76]$
[20]	IBM Jazz^{TM}	Hoeffding Tree	K-fold cross-validation	$R = 0.65$, $A = 0.72$
[25]	Ant, Maven, Gradle	Random forest	K-fold cross-validation	$R = 0.82$, $P = 0.85$, $F_1 = 0.83$, $R^{'} = 0.93$, $P^{'} = 0.92$, $F_1^{'} = 0.92$
[41]	532 OSS projects available on TravisTorrent	Cascaded classifiers, C4.5, Naive Bayes	Shuffle split	$R = 0.69$, $A = 0.74$
[55]	126 OSS projects available on TravisTorrent	CART, Gradient boosting, Logistic regression, Multilayer perception, Multinomial naive bayes, Nearest centroid, Nearest neighbours, Random forest, Ridge regression	K-fold cross-validation; On-line prediction	Cross-validation: 44 projects achieve $F_1 > 0.6$, 21 of which over 0.7; On-line: More than 50% projects achieve $F_1 < 0.3$.

* R, P, and F_1 denote the Recall, Precision, and F1-measure for *failed* respectively; $R^{'}$, $P^{'}$, and $F_1^{'}$ denote the metrics for *passed*; A denotes the overall accuracy.

The work of Seiffert et al. [47] is most similar to the research in this paper. They conducted an empirical study to evaluate the effects of imbalance and noise on predicting fault-prone software modules. Imbalanced methods other than sampling were not considered in their work and the 10-fold cross-validation was used. They also investigated the impacts of degree of imbalance, but only balanced classifiers were evaluated. Their study indicated that the degree of imbalance only has significant impact on RBF network, which is partially consistent with the conclusions of our work. This may be because they evaluated the performance using AUC since the ROC curve itself is not sensitive to the degree of imbalance.

Time-series-validation was rarely discussed in SE. Tantithamthavorn et al. [52] compared different validation methods, but time-series was not considered. Jimenez et al. [31] indicated that introducing future labels in the training set can lead to optimistic results when predicting software vulnerabilities. It reveals the potential problems with cross-validation, but they did not compare validation methods.

5.2 Predictive CI/CD

Table 4 summarizes existing research that applied machine learning techniques to predict the build outcomes, including the data sets used for validation, the evaluated algorithms, and the main validation results. Since a variety of algorithms were evaluated in [41, 55], we only present the best performance.

The concept of predicting build outcome through mining repository was first proposed by Hassan and Zhang [24]. They evaluated the performance of decision trees built using different factors for predicting build outcome. The study showed that using a combination of all kinds of factors performed better, but the prediction of *failed* was not as accurate as *passed*.

Wolf et al. [54] constructed a Bayesian classifier using communication network measures to predict whether integration will fail. However, the model did not achieve satisfactory performance.

Finlay et al. [20] compared Hoeffding Tree classification method with C4.5 and the results showed the former performed better. However, they indicated that it was difficult to predict *failed* accurately.

Hassan and Wang [25] evaluated a model using Random forest algorithm with the metrics available on TravisTorrent as input. Although the model performed well on predicting *failed*, only three large projects, Ant, Maven, and Gradle were used for evaluation.

Using similar metrics available on TravisTorrent, Ni and Li [41] evaluated the performance of cross-project prediction. They compared three algorithms including Cascaded Adaboost, C4.5 and NaiveBayes. Cascaded Adaboost performed better on average, but Naive Bayes was not far behind.

Xia and Li [55] compared nine classifiers using 126 OSS projects from TravisTorrent. They evaluated two scenarios - cross-validation and on-line prediction. All classifiers performed far worse in on-line prediction than in cross-validation, but they did not touch on the root cause of this phenomenon.

It seems that existing research has provided good predictive performance on average using just some common algorithms. However, the performance of the models varies widely on different projects and the cause was not found. In addition, the preference for *failed* in CI/CD was not adequately considered in terms of evaluation metrics and their training sets contain future data, which may lead to optimistic results.

6 CONCLUSIONS

In the application of machine learning, SE researchers should raise their concerns on the assumptions of various methods, or the results may be biased. Our empirical study confirms that the cross-validation would result in an biased result for time series data; the imbalanced learning should be considered for imbalance problems since it would provide better performance than using balanced models. Although these are not novel problems, there are plenty of studies that are not rigorous enough. This paper takes the study of CI/CD prediction as an example to remind the research community to take care of the fundamental assumptions in machine learning. In particular, we should revisit the studies that applied cross-validation on time series data. This study systematically compared time-series-validation and cross-validation in the imbalance classification problem, which fills a gap in the relevant research.

Even with the consideration of imbalanced learning, existing models still have significant limitations. We found no model can achieve an F_2 of 0.6 on average. However, on some data balanced projects, multiple imbalanced models have achieved F_2 values in excess of 0.7. Even for imbalanced models, the degree of data balance largely determines the upper limit of performance. To some extent, it explains the phenomenon that the model performance varies widely between different projects in the existing research, and also provides a reference for whether CI/CD prediction can be applied in a specific project.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (Grant No.61572251).

REFERENCES

[1] Sylvain Arlot and Alain Celisse. 2010. A survey of cross-validation procedures for model selection. *Stat. Surv.* 4, 2010 (2010), 40–79.

[2] Abigail Atchison, Christina Berardi, Natalie Best, Elizabeth Stevens, and Erik Linstead. 2017. A time series analysis of TravisTorrent builds: to everything there is a season. In *Proc. of the 14th Int. Conf. on Mining Softw. Repos. (MSR'17)*. 463–466.

[3] Alejandro Correa Bahnsen, Djamila Aouada, and Björn E. Ottersten. 2014. Example-Dependent Cost-Sensitive Logistic Regression for Credit Scoring. In *2014 13th Int. Conf. on Mach. Learn. and Appl. (ICMLA'14)*. 263–269.

[4] Alejandro Correa Bahnsen, Djamila Aouada, and Björn E. Ottersten. 2015. Ensemble of Example-Dependent Cost-Sensitive Decision Trees. *CoRR* (2015).

[5] Alejandro Correa Bahnsen, Djamila Aouada, and Björn E. Ottersten. 2015. Example-dependent cost-sensitive decision trees. *Expert Syst. With Appl.* 42, 19 (2015), 6609–6619.

[6] Alejandro Correa Bahnsen, Aleksandar Stojanovic, Djamila Aouada, and Björn E. Ottersten. 2014. Improving Credit Card Fraud Detection with Calibrated Probabilities. In *Proc. 2014 SIAM Inter. Conf. on Data Mining (SDM'14)*. 677–685.

[7] Gustavo E. A. P. A. Batista, Ana L. C. Bazzan, and Maria Carolina Monard. 2003. Balancing Training Data for Automated Annotation of Keywords: a Case Study. In *II Brazilian Workshop on Bioinformatics, December 3-5, 2003, Macaé, RJ, Brazil (WOB'03)*. 10–18.

[8] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. 2004. A study of the behavior of several methods for balancing machine learning training data. *Sigkdd Explorations* 6, 1 (2004), 20–29.

[9] Moritz Beller, Georgios Gousios, and Andy Zaidman. 2017. TravisTorrent: synthesizing Travis CI and GitHub for full-stack research on continuous integration. In *Proc. 14th Int. Conf. on Min. Softw. Repos. (MSR'17)*. 447–450.

[10] Kwabena Ebo Bennin, Jacky Keung, Passakorn Phannachitta, Akito Monden, and Solomon Mensah. 2018. MAHAKIL: diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. In *Proc. 40th Int. Conf. on Softw. Eng. (ICSE'18)*. 699.

[11] Leo Breiman. 2001. Random Forests. *Mach. Learn. archive* 45, 1 (2001), 5–32.

[12] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. Wadsworth.

[13] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *Proc. 2000 ACM SIGMOD Int. Conf. on Manage. of Data*, Vol. 29. 93–104.

[14] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *J. Artfi. Intell. Res.* 16, 1 (2002), 321–357.

[15] Chao Chen. 2004. Using Random Forest to Learn Imbalanced Data. (2004).

[16] Thomas G. Dietterich. 2000. Ensemble Methods in Machine Learning. *Multiple Classifier Syst.* (2000), 1–15.

[17] Paul M. Duvall, Steve Matyas, and Andrew Glover. 2007. *Continuous Integration: Improving Software Quality and Reducing Risk*.

[18] Bradley Efron. 1983. Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation. *J. American Statistical Association* 78 (06 1983), 316–331.

[19] Charles Elkan. 2001. The foundations of the cost-sensitive learning. In *Proc. 17th Int. joint Conf. on Artfi. Intell. (IJCAI'01)*. 973–978.

[20] Jacqui Finlay, Russel Pears, and Andy M. Connor. 2014. Data stream mining for predicting software build outcomes using source code metrics. *Inf. & Softw. Technol.* 56, 2 (2014), 183–198.

[21] Jerome H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. *Annals of Stat.* 29, 5 (2001), 1189–1232.

[22] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. 2012. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Trans. Softw. Eng.* 38, 6 (2012), 1276–1304.

[23] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. 2005. Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. In *Int. Conf. on Intell. Comput. (ICIC'05)*. 878–887.

[24] Ahmed E. Hassan and Ken Zhang. 2006. Using Decision Trees to Predict the Certification Result of a Build. In *21st IEEE/ACM Int. Conf. on Autom. Softw. Eng. (ASE'06)*. 189–198.

[25] Foyzul Hassan and Xiaoyin Wang. 2017. Change-aware build prediction model for stall avoidance in continuous integration. In *Proc. 11th ACM/IEEE Int. Sysmp. on Empirical Softw. Eng. and Meas. (ESEM'17)*. 157–162.

[26] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE Int. Joint Conf. on Neural Networks (IJCNN'08)*. 1322–1328.

[27] Haibo He and Edwardo A. Garcia. 2009. Learning from Imbalanced Data. *IEEE Trans. on Knowl. and Data Eng.* 21, 9 (2009), 1263–1284.

[28] Wang He-yong. 2008. Imbalance Data Set Classification Using SMOTE and Biased-SVM. *Comput. Science* (2008).

[29] Robert C. Holte, Liane E. Acker, and Bruce W. Porter. 1989. Concept learning and the problem of small disjuncts. In *Proc. 11th Int. joint Conf. on Artfi. Intell. (IJCAI'89)*. 813–818.

[30] Nathalie Japkowicz and Shaju Stephen. 2002. The class imbalance problem: A systematic study. *Intell. Data Anal.* 6, 5 (2002), 429–449.

[31] Matthieu Jimenez, Renaud Rwemalika, Mike Papadakis, Federica Sarro, Yves Le Traon, and Mark Harman. 2019. The Importance of Accounting for Real-World Labelling When Predicting Software Vulnerabilities. In *Proc. 27th ACM Joint Meeting on European Softw. Eng. Conf. and Sysmp. on the Found. Softw. Eng. (ESEC/FSE'19)*. 695–705.

[32] Xiao-Yuan Jing, Fei Wu, Xiwei Dong, and Baowen Xu. 2017. An Improved SDA Based Defect Prediction Framework for Both Within-Project and Cross-Project Class-Imbalance Problems. *IEEE Trans. Softw. Eng.* 43, 4 (April 2017), 321–339.

[33] Miroslav Kubat and Stan Matwin. 1997. Addressing the Curse of Imbalanced Training Sets: One-Sided Selection. In *Proc. of the Fourteenth Inter. Conf. on Mach. Learn. ICML'17*. 179–186.

[34] Jorma Laurikkala. 2001. Improving Identification of Difficult Small Classes by Balancing Class Distribution. *Artfi. Intell. in Med. in Eur.* (2001), 63–66.

[35] Guillaume Lemaitre, Fernando Nogueira, and Christos K. Aridas. 2017. Imbalanced-learn: a python toolbox to tackle the curse of imbalanced datasets in machine learning. *J. of Mach. Learn. Res.* 18, 1 (2017), 559–563.

[36] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. 2009. Exploratory Undersampling for Class-Imbalance Learning. *IEEE Trans. Systems, Man, and Cybernetics* 39, 2 (2009), 539–550.

[37] Gilles Louppe and Pierre Geurts. 2012. Ensembles on random patches. In *Proc. 2012th Eur.an Conf. on Mach. Learn. and Knowl. Discovery in Databases (ECMLP-KDD'12)*. 346–361.

[38] Larry M. Manevitz and Malik Yousef. 2002. One-class svms for document classification. *J. Mach. Learn. Res.* 2, 2 (2002), 139–154.

[39] Antonio Maratea, Alfredo Petrosino, and Mario Manzo. 2014. Adjusted F-measure and kernel scaling for imbalanced data learning. *Inf. Sci.* 257 (2014), 331–341.

[40] David Mease, Abraham J. Wyner, and Andreas Buja. 2007. Boosted Classification Trees and Class Probability/Quantile Estimation. *J. of Mach. Learn. Res.* 8 (2007), 409–439.

[41] Ansong Ni and Ming Li. 2017. Cost-effective build outcome prediction using cascaded classifiers. In *2017 IEEE/ACM 14th Int. Conf. on Min. Softw. Repos. (MSR'17)*. 455–458.

[42] J. Platt. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers* (1999).

[43] Bhavani Raskutti and Adam Kowalczyk. 2004. Extreme re-balancing for SVMs: a case study. *Sigkdd Explorations* 6, 1 (2004), 60–69.

[44] Duksan Ryu, Jong-In Jang, and Jongmoon Baik. 2017. A transfer cost-sensitive boosting approach for cross-project defect prediction. *Softw. Qual. J.* 25, 1 (2017), 235–272.

[45] Miriam Seoane Santos, Jastin Pompeu Soares, Pedro Henriques Abreu, Helder J. Araujo, and Joao A M Santos. 2018. Cross-Validation for Imbalanced Datasets: Avoiding Overoptimistic and Overfitting Approaches. *IEEE Comput. Intell. Mag.* 13, 4 (2018), 59–76.

[46] C. Seiffert, T.M. Khoshgoftaar, J. Van Hulse, and A. Napolitano. 2010. RUSBoost: A Hybrid Approach to Alleviating Class Imbalance. *IEEE Trans. Systems, Man, and Cybernetics* 40, 1 (2010), 185–197.

[47] Chris Seiffert, Taghi M. Khoshgoftaar, Jason Van Hulse, and Andres Folleco. 2014. An empirical study of the classification performance of learners on imbalanced and noisy software quality data. *Inf. Sci.* 259 (2014), 571–595.

[48] Victor S. Sheng and Charles X. Ling. 2006. Thresholding for making classifiers cost-sensitive. In *Proc. 21st national Conf. on Artfi. Intell. (AAAI'06)*. 476–481.

[49] Michael J. Siers and Md Zahidul Islam. 2016. Addressing Class Imbalance and Cost Sensitivity in Software Defect Prediction by Combining Domain Costs and Balancing Costs. In *Advanced Data Min. and Appl. - 12th Int. Conf., ADMA 2016, Gold Coast, QLD, Australia, December 12-15, 2016, Proc.* 156–171.

[50] Michael R. Smith, Tony R. Martinez, and Christophe G. Giraud-Carrier. 2014. An instance level analysis of data complexity. *Mach. Learn.* 95, 2 (2014), 225–256.

[51] Daniel Ståhl and Jan Bosch. 2014. Modeling continuous integration practice differences in industry software development. *J. Syst. Softw.* 87, 1 (2014), 48–59.

[52] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2017. An Empirical Comparison of Model Validation Techniques for Defect Prediction Models. *IEEE Trans. Softw. Eng.* 43, 1 (2017), 1–18.

[53] Gary M. Weiss. 2004. Mining with rarity: a unifying framework. *Sigkdd Explorations* 6, 1 (2004), 7–19.

[54] Timo Wolf, drian Schröter, Daniela E. Damian, and Thanh H. D. Nguyen. 2009. Predicting build failures using social network analysis on developer communication. In *Proc. 31st Int. Conf. on Softw. Eng. (ICSE'09)*. 1–11.

[55] Jing Xia and Yanhui Li. 2017. Could We Predict the Result of a Continuous Integration Build? An Empirical Study. In *2017 IEEE Int. Conf. on Softw. Qual., Reliab. and Secur. Companion (QRS-C'17)*. 311–315.

[56] Jianping Zhang and Inderjeet Mani. 2003. KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction. In *Icml Workshop on Learning from Imbalanced Datasets*.