Simplifying Performance Regression

MEGHA GUPTA, Rochester Institute of Technology
NIKHIL MALKARI, Rochester Institute of Technology
TANNER PETERSON, Rochester Institute of Technology
ADDL TARIQ, Rochester Institute of Technology

ACM Reference Format:
Megha Gupta, Nikhil Malkari, Tanner Peterson, and Addl Tariq. 2022. Simplifying Performance Regression. 1, 1 (April 2022), 7 pages.

1 INTRODUCTION

Performance is the most important aspect in a software project. Due to an increase in the number of changes committed in a project, calculating the performance after every commit is infeasible. Moreover, performance tests are time consuming and expensive in terms of resource utilization. Frequently resources are spent on other software testing, such as working through bugs that lead to failures in software. This is one of the major reasons testing for performance regressions is overlooked. On the other hand, running the test suite infrequently will not identify which of the multiple commits impacted the performance. A software system is considered to have regression when the performance of [1] a certain feature of the system is worse than before. Examples of performance regressions are response time degradation and increased resource utilization. Though in general performance regression does not occur as frequently as bugs in the code, it can sometimes impact a program by needing to revert hundreds of commits to find the correct commit which resulted in the performance hit.

In real-world software applications the percentage of commits having a performance impact is very low, this causes any machine learning implications. Machine learning algorithms work well on balanced, evenly distributed, datasets. Training classifiers on imbalanced datasets would result in a model that would be heavily biased towards the majority classes. Another scalability issue is the type of data availability for adding to the existing dataset. Since software applications are architected differently, it causes the model to be less generalized and adaptable in various environment. These studies have identified metrics regarding actual lines of code and execution, but do not study metrics for automatically detecting performance regression.

In light of the above-mentioned problems, we are using an array of binary classification algorithms, with data level and model level resampling as a solution to predict the need of running performance regression test on a commit change. The training data set contain features engineered through the performance test for each commit.

Authors' addresses: Megha Gupta, Rochester Institute of Technology; Nikhil Malkari, Rochester Institute of Technology; Tanner Peterson, Rochester Institute of Technology; Addl Tariq, Rochester Institute of Technology.

Figure 1 shows the dataset used for this research and a summary of the features below. Original commit A were the prior versions of the software used for this study, which is a Git Java project. The new commit B is the new version that could introduce performance regression. Each record has a benchmark test that assisted identifying performance regression. There are an array of static software metrics such as code counts, deleted, and cyclomatic complexities. The dynamic metrics include % of deletions, % call changes, etc.

- Original Commit A
- New Commit B
- Benchmark test for performance regression
- Static software metrics
- Dynamic software metrics

Target class for trained data are formed by different benchmarking tests that are testing for regression. Any record that has been identified as adding performance regression were labeled as the "hit" and normal commits are flagged as "dismiss". Preprocessing the dataset includes getting data machine learning ready, applying different resampling methods, and model selection. The next step will be to train the model using the k-fold cross-validation like previous studies [5,6,7], in order to prevent overfitting of results to one of the classes. Class imbalance is likely to be seen in the resulting procedure due to the infrequent occurrence that a commit contains regression. To migrate such an issue on some level we will use the Random Under sampling (RUS), Random Over sampling (ROS), and Synthetic Minority Oversampling and Adaptive Synthetic Sampling Techniques (SMOTE) to test out the best fit for our results. A range of machine learning models will be fit to the sampled training data set to attain the optimal results, which will include, but not limited to, Random Forest, SVM, Naive Bayes, and XGBoost. Metrics in form of recall and specificity will be used to evaluate the results for the final selection of model. The model will then be pushed to the application phase, where the real-world dataset will then be processed without the target class and will be passed through the predictive model to obtain the predicted outcomes. The approach will be compared to other research-based baselines to draw final a conclusive decision on scalability and usability of the approach.

Field	Example
Commit A	d9545c7f
Commit B	8f449614
Benchmark	p0000-perf-lib-sanity.sh-perf-report
Del Func >= X	983
New Func >= X	515
Reached Del Func >= X	7
Top Chg by Call >= X%	0.07
Top > X% by Call Chg by >= 10%	0.03
Top Chg by Instr >= X%	FALSE
Top Chg Len >= X%	1
Top Reached Chg Len >= X%	1
AltCountLineBlank	1855
AltCountLineCode	9572
AltCountLineComment	2807
CountLine	14160
CountLineBlank	1845
CountLineCode	8984
CountLineCodeDecl	2480
CountLineCodeExe	6078
CountLineComment	2768
CountLineInactive	168
CountLinePreprocessor	506
CountSemicolon	4652
CountStmt	6548
CountStmtDecl	1676
CountStmtEmpty	29
CountStmtExe	4843
SumCyclomatic	2004
SumCyclomaticModified	1935
SumCyclomaticStrict	2268
SumEssential	968
MaxCyclomatic	1
MaxCyclomaticModified	1
MaxCyclomaticStrict	0
MaxNesting	0
AltAvgLineBlank	0.009604178
AltAvgLineCode	0.002129281
AltAvgLineComment	-0.038383091
AvgCyclomatic	-0.025327237
AvgCyclomaticModified	-0.023220369
AvgCyclomaticStrict	-0.019544558
AvgEssential	-0.048357092
AvgLine	-0.027949614
AvgLineBlank	0.008180922
AvgLineCode	-0.003675811
AvgLineComment	-0.037833961
Cyclomatic	-0.126343446
CyclomaticModified	-0.127949386
CyclomaticStrict	-0.160790747
Hit/Dismiss	0

Fig. 1. Dataset

2 METHODOLOGY

Data used for this research was the same curated dataset that was used in 'Learning to Characterize Performance Regression Introducing Code Changes' by AlShoaibi et al [10]. The data collected was from a Git project that included 6,383 records of commits. 46 features can be noted in which consist of changes to codes and cyclomatic complexities (Figure 1). Also included in the data is a benchmark where each record had a performance test conducted for each commit. One record consisted of null values, and was dropped leaving the study with 6,382 records.

First phase of preprocessing is to analyze the commit dataset used for our study. There are 712 commits that compare Commit A with Commit B and several benchmark tests for each commit totaling the dataset at 6,383 records. 46 features are software metrics to analyze whether or not a commit includes performance regression. The last column in the data is our target variable, 'Hit' for a performance regression and 'Dismiss' for non performance regression commits.

Dataset was read into a Jupyter Notebook[1] using the pandas[2] library. Total count of hits are 401 to the 5,982 dismiss tests, or only 6.3% of the data. Dropped one NA using dropna(), bringing the total to 6382 records. In order to leverage machine learning models, all data types must be converted from object to integers. This was completed using scikit-learn[3] preprocessing package LabelEncoder. Following being encoded, sklearn ensemble has a model feature importance function that describes what columns are either helping the classifier decide hit or dismiss by assigning importance from 0-1. After running this we dropped 6 features from the dataset; 'Reached Del Func >= X,Top Chg by Call >= X%,Top > X% by Call Chg by >= 10%,Top Chg by Instr >= X%, and Top Chg Len >= X%, and Top Reached Chg Len >= X% (figure x). Last preprocessing function used was MinMaxScaler from sklearn preprocessing, which assigns all values between 0-1. This is important for modeling techniques that do not accept negative values.

3 EXPERIMENTS

This section investigates effective resampling methods for classifying commits that introduce performance regression by answering two research questions. Included are results from different machine learning algorithms using a baseline, SMOTE, RUS, and ROS. Various metrics defined below can show the effectiveness of our method and demonstrates what metrics are most impactful for class imbalance problems.

- Classification Accuracy: The total number of correct classifications over the total number tested. This metric does not tell the full story. In a class imbalance problem, baselines can classify no target variables correctly, yet still score a high accuracy.
- Precision: Total of true positives over both the true positives and false positives. This is how correct we are at identifying hit commits that introduce performance regression out of all the hit commits. Precision is not as important in our study as recall, because if we have a false positive that commit may still need to be inspected. This could lead to lost time, and it is decided to still evaluate based on precision.
- Recall: The total true positives over the true positives and false negatives. How accurate our model is predicting performance regression commits throughout the data. Accuracy on identifying hits is more important than false positives and for that reason we rely to recall over precision.
- F1-Score: Balances our precision and recall.
- Specificity: Identifies the true negatives, or accurately classified performance regression commits. This is important to measure if our method can distinguish between the target variable tested.

[1] https://jupyter.org/
[2] https://pandas.pydata.org/docs/
[3] https://scikit-learn.org/stable/

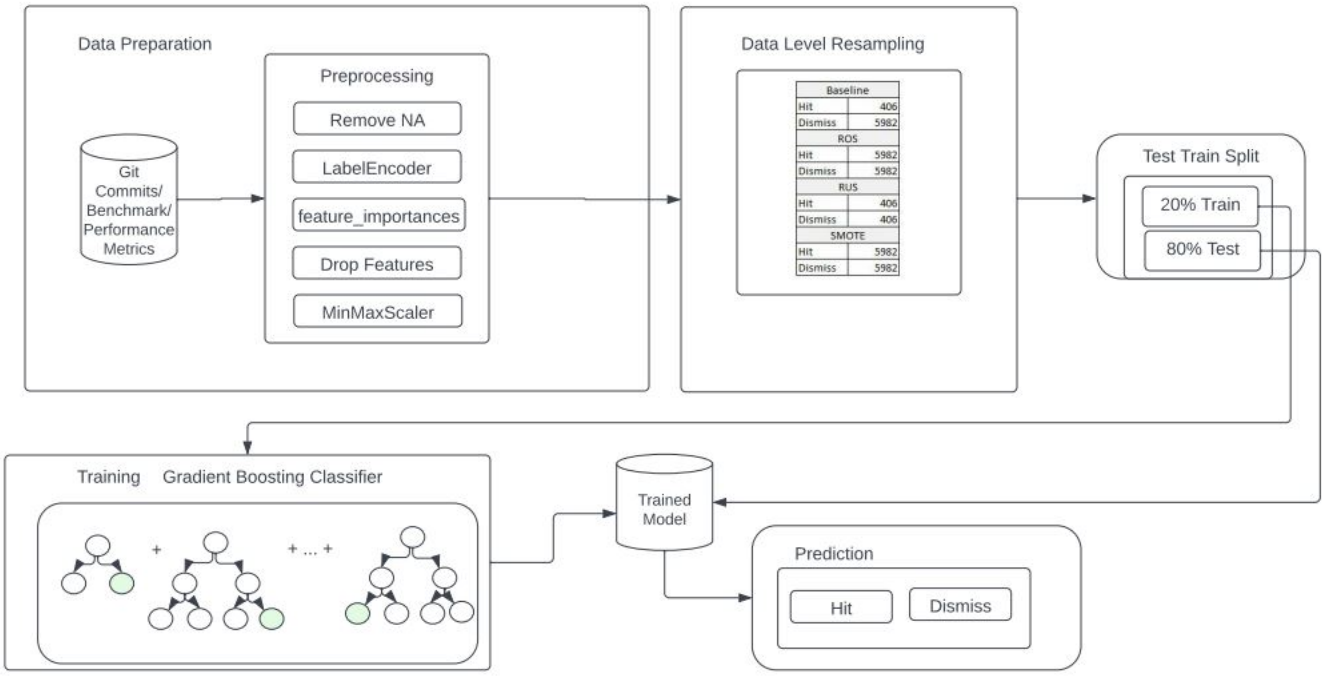

Fig. 2. Approach Overview

- BCR: Balance classification rate, or balanced accuracy, is crucial in identifying how our classification methods are performing considering both sensitivity (recall) and specificity. Important metric given the imbalanced nature of the dataset.

RQ1: Can machine learning classifiers identify performance commits without resampling techniques?

Experiments conducted without resampling methods on the imbalanced dataset are noted as 'Baseline' in figure 2. The Bayes method chosen was ComplementNB[4] that is suited for imbalanced datasets had the highest recall of 0.18 compared to logistic regression, random forest classifier, SVM, and boosted decision tree with 0.0, 0.14, 0.0, and 0.15 respectively. The F1 score that performed the highest was random forest classifier and boosted decision tree, which are both ensemble methods, scoring 0.24 and 0.23 respectively. Nearly twice the model completeness over the three other methods. With regards to RQ1, machine learning classifiers can help predict performance regression hits with some accuracy, however this model completeness will not help developers.

RQ2: Can resampling techniques improve classification to detect performance regression commits?

Experiments conducted using resampling methods were compared to a baseline of the original dataset for each model. All methods were created utilizing Jupyter Notebooks and written in python using scikit-learn libraries. Test and train were all 80 percent and 20% respectively. Three resampling methods were used; SMOTE, RUS, and ROS through the imbalanced-learn[5] python library. Ensemble methods perform best for class imbalance situations, however the study decided to test an array of models explained below.

Logistic regression and SVM both had a baseline recall of 0, meaning that these models do not initially take class imbalance into consideration. Logistic regression is powerful for predicting an outcome such as true or false yet struggles when one of the two is not present in a somewhat equal distribution. Both logistic regression and SVM improved with resampling techniques. Logistic regression top performing with random over sampling and SVM with random under sampling.

The Bayes method chosen was ComplementNB performed at the top of our baseline from RQ1. The baseline method held true to handle class imbalance, but introducing resampling methods caused the lowest results in the experiment in all metrics but one; Bayes performed the top in recall using random under sampling. This is important to show that this method accurately identifies commits that have performance regression, but also introduces the falsest

[4]https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html
[5]https://imbalanced-learn.org/stable/

Classifier	Resampling Technique	Accuracy	Precision	Recall	F1-Score	Specificity	BCR
Logistic Regression	Baseline	0.94	0.00	0.00	0.00	1.00	0.50
	SMOTE	0.62	0.09	0.54	0.15	0.63	0.58
	RUS	0.60	0.08	0.50	0.14	0.61	0.58
	ROS	0.55	0.09	0.66	0.16	0.54	0.59
Random Forest Classifer	Baseline	0.94	0.70	0.14	0.23	0.99	0.57
	SMOTE	0.95	0.30	0.27	0.39	0.99	0.62
	RUS	0.64	0.12	0.76	0.21	0.63	0.70
	ROS	0.95	0.88	0.29	0.44	0.99	0.63
SVM	Baseline	0.94	0.00	0.00	0.00	1.00	0.50
	SMOTE	0.70	0.10	0.49	0.17	0.72	0.60
	RUS	0.58	0.09	0.60	0.16	0.58	0.59
	ROS	0.65	0.09	0.50	0.15	0.66	0.58
Bayes	Baseline	0.82	0.08	0.18	0.11	0.87	0.52
	SMOTE	0.24	0.06	0.79	0.11	0.21	0.50
	RUS	0.23	0.06	0.80	0.12	0.19	0.50
	ROS	0.23	0.06	0.82	0.12	0.19	0.51
Boosted Decision Tree	Baseline	0.94	0.31	0.15	0.24	0.99	0.57
	SMOTE	0.90	0.96	0.50	0.39	0.93	0.70
	RUS	0.75	0.16	0.77	0.27	0.74	0.77
	ROS	0.91	0.34	0.54	0.42	0.93	0.74

Fig. 3. Performance Metrics

positives. Leads to explain how precision on Bayes ranked last in the experiment. Too many false positives will be time wasted investigating performance regression commits that were not actually causing an issue.

Ensemble methods, random forest and boosted decision tree performed the best overall with regards to balance of precision and recall, specificity, and BCR. Both performed a balanced model F1 score with random over sampling methods, which performed the highest recall with under sampling.

Limitations to the methodology used of resampling and model selection can be noted than when this method is used, precision decreases in every model except the boosted decision tree. On the other side, recall benefits. This is part of the trade-off between the two metrics and can be noted in the confusion matrices (Figures 4,5). While we are undersampling the majority class, duplicated the minority class, or creating synthetic examples of the minority class, more false positives are being introduced. Trade off for whether time is spent looking through false positives, or a balanced algorithm. From both sides, developers can utilize these two models to evaluate what fits best with their workload and commitment to fixing performance regression.

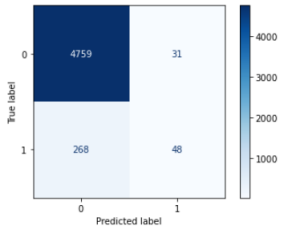

Fig. 4. Boosted Decision Tree Baseline

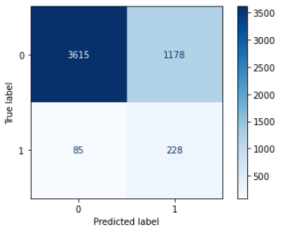

Fig. 5. Boosted Decision Tree RUS

Classifier	Resampling Technique	Stratified K Fold	Accuracy	Precision	Recall	F1-Score	Specificity	BCR
Random Forest Classifer	Baseline	5	0.96	0.93	0.35	0.51	0.99	0.67
	SMOTE	5	0.98	0.98	0.69	0.81	0.99	0.84
	RUS	5	0.80	0.22	0.90	0.35	0.79	0.84
	ROS	5	0.97	0.90	0.59	0.71	0.99	0.79
Boosted Decision Tree	Baseline	5	0.95	0.85	0.21	0.34	0.99	0.60
	SMOTE	5	0.88	0.31	0.68	0.42	0.90	0.79
	RUS	5	0.79	0.21	0.81	0.33	0.79	0.80
	ROS	5	0.88	0.30	0.86	0.47	0.88	0.87

Fig. 6. RFC SMOTE Stratified K-Folds

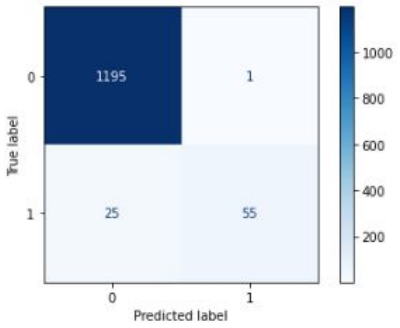

Fig. 7. RFC SMOTE Stratified K-Folds

The other resampling technique used alongside data level resampling, was a stratified k-folds cross-validation. This specific type of fold is sure to distribute the minority class evenly among folds. The results were great when coupled with data level resampling. SMOTE scored an F-1 score of .81 where there were few false positives. What this means in the research problem is that developers will not waste time using a random forest classifier, SMOTE data level resampling paired with k-folds, to identify performance regression commits.

RQ3: How does this technique differ from other performance regression studies?

There are many approaches taken to solve this problem varying from a rule-based approach to use of stochastic evolutionary algorithm. One of the rule-based approach related Perphecy [8] attempts to use combination of features through the tuning process to assign threshold, but it limits the training to a certain search area and does not take in to account the program languages that compiles the code dynamically. Hence, the average specificity for this approach of 0.48 is not that high as compared to 0.85 of our models. Another approach taken by the researchers is the use of NSGA-II which a stochastic evolutionary algorithm, which has been discussed in the PRICE [3]. The result shown by this approach is better than Perphecy but not comparable to our model performance. Our technique handles class imbalance problem on a greater effect when combined with a classifier to improve the results as compared to the regression based technique.

4 ADDITIONAL IMPROVEMENTS

After achieving the best performance using SMOTE based Random Forest classifier with k-folds, we conducted feature engineering to analyze which features were contributing to the performance of RFC. Next, we plotted the tree based feature importance and the permutation importance graph for the most contributing feature set. The permutation importance is calculated on the training set to show how much the model relies on each feature during the training. Visual analysis of the graph clearly showed that the data was highly collinear. When features are collinear, permutating one feature will have little effect on the models performance because it can get the same information from a correlated feature. We handled these multicollinear features by performing hierarchical clustering on the Spearman rank-order correlations picking a threshold of 1, and keeping a single feature from each cluster studying the heatmap and dendrogram of the correlated features. The clustering based feature engineering approach listed out 2 top features for the evaluation of the model and gave the accuracy of 99%. On evaluating SMOTE-based RFC with ‘Commit A’ and ‘AltCountLineBlank’ a significant increase was observed in the F1-score, recall, precision and accuracy.

5 RELATED WORK

Studies have been conducted ranging from a rule-based deterministic approach to traditional and evolutionary machine learning algorithms, to correctly predict the performance impact of code changes. Deterministic approach using Perphecy, which adopts a greedy approach to manually iterate over all the possible combinations of commits and selects a rule which gives the highest detection. Since Perphecy uses a single set of binaries that contains the entire code of application, it was not suitable for programming languages that load the code dynamically [2]. Another approach was PRICE [3] was again a rule-based approach which performed better than Perphecy and NSGA – II. Another approach conducted a study to prioritize or automate the time it takes to perform performance regression testing. Identifying changes in loops and time for execution [4] can help prioritize which commits developers should prioritize to not take time away from other crucial developments. Both of these prior works had left out the model metrics such as precision,

	Classifier	Resampling Technique	Stratified K Fold	FS:HC-SROC	Accuracy	Precision	Recall	F1-Score	Specificity	BCR
0	Random Forest Classifier	Baseline	5	No	0.963166	0.945946	0.4375	0.598291	0.998328	0.717914
1	Random Forest Classifier	SMOTE	5	No	0.977273	0.963636	0.6625	0.785185	0.998328	0.830414
2	Random Forest Classifier	RUS	5	No	0.786050	0.211940	0.8875	0.342169	0.779264	0.833382
3	Random Forest Classifier	ROS	5	No	0.967085	0.913043	0.5250	0.666667	0.996656	0.760828
4	Random Forest Classifier	Baseline	5	Yes	0.991379	0.985915	0.8750	0.927152	0.999164	0.937082
5	Random Forest Classifier	SMOTE	5	Yes	0.992163	0.986111	0.8875	0.934211	0.999164	0.943332
6	Random Forest Classifier	ROS	5	Yes	0.978840	0.907692	0.7375	0.813793	0.994983	0.866242
7	Random Forest Classifier	RUS	5	Yes	0.797022	0.206557	0.7875	0.327273	0.797659	0.792579

Fig. 8. Random Forest Classifier Stratified K-Folds with results

classification accuracy, and F-1 scores. Thus, their research only pointing out the positives. The research conducted in this study took into account the overall model performance, what kind of false positives and negatives are being introduced, and open to developer feedback.

6 THREATS TO VALIDITY

Threats to conclusion validity include stating that adding in resampling methods and false positives to increase recall is acceptable for developers to use for handling performance regression. In the previous study that used this dataset [10] the authors decided to leave off accuracy and F-1 score. This research included these to show that it is known that accuracy decreases and there is a trade-off between precision and recall.

Threats to internal validity consist of assuming that correct features were dropped and correct binary classifiers were chosen. To handle this, the research methodology dropped only the bottom six features that did not have prediction power. Models selected were from common binary classifiers and specifically chosen to handle imbalanced data. When comparing with a baseline this research was able to conclude that classifiers built a relationship between non-impacting commits and detrimental commits.

Threats to construct validity in this research can be validated by the fact this project was from a Git open source project. It can be reflective of a real life scenario because it is. However, the largest threat resides in the metrics chosen and if they are correct in finding performance regression. Software metrics including line counts, deletion of code, added cyclomatic complexity are used as metrics for sound software. The threat includes what other metrics could this research be missing to show a larger relationship and could they increase prediction power of performance regression.

Threats to external validity in this study include; Can this project selected indicate how this methodology would work on another project to identify performance regression. To handle this threat another study would have to be considered to ensure this predicts performance regression. It can be noted that the methodology of resampling has been studied and used for imbalanced datasets. The methodology itself checks out and can be applied to another project in future work.

7 CONCLUSION

Baseline methods that were not utilizing modeling level, or data level, resampling proved to not be as accurate as predicting performance regression than those that did. Using RUS, ROS, and SMOTE proved to be successful, while modeling balancing level packages proved less successful. When using a random forest classifier, paired with SMOTE and stratified k-folds, developers can identify performance regression commits without wasting time on false positives. Limitations to this work include deciding what threshold of false negatives developers are comfortable with handling, as using resampling methods missed some true positives. The threats to validity explained above show that the research is aware that the methodology needs further research to show it can be as successful as with the project tested.

8 REFERENCES

[1] Jinfu Chen and Weiyi Shang. 2017. An exploratory study of performance regression introducing code changes. 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME) (November 2017).
DOI:http://dx.doi.org/10.1109/icsme.2017.13
[2] Augusto Born De Oliveira, Sebastian Fischmeister, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. 2017.
Perphecy: Performance Regression Test selection made simple but effective. 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST) (March 2017).
DOI:http://dx.doi.org/10.1109/icst.2017.17
[3] Deema Alshoaibi, Kevin Hannigan, Hiten Gupta, and Mohamed Wiem Mkaouer. 2019. Price: Detection of performance regression

introducing code changes using static and dynamic metrics. Search-Based Software Engineering (2019), 75–88.
DOI:http://dx.doi.org/10.1007/978-3-030-27455-9-6
[4] U. Sivaji and P. Srinivasa Rao. 2021. Test case minimization for regression testing
by analyzing software performance using the novel method. Materials Today: Proceedings (February 2021).
DOI:http://dx.doi.org/10.1016/j.matpr.2021.01.882
[5] Eman Abdullah AlOmar, Anthony Peruma, Mohamed Wiem Mkaouer,
Christian Newman, Ali Ouni, and Marouane Kessentini. 2021.
How we refactor and how we document it? on the use of supervised machine
learning algorithms to classify refactoring documentation.
Expert Systems with Applications 167 (2021), 114176.
DOI:http://dx.doi.org/10.1016/j.eswa.2020.114176
[6] Fan Fang, John Wu, Yanyan Li, Xin Ye, Wajdi Aljedaani,
and Mohamed Wiem Mkaouer. 2021.
On the classification of bug reports to improve bug localization.
Soft Computing 25, 11 (2021), 7307–7323.
DOI:http://dx.doi.org/10.1007/s00500-021-05689-2
[7] Lobna Ghadhab, Ilyes Jenhani, Mohamed Wiem Mkaouer,
and Montassar Ben Messaoud. 2021.
Augmenting commit classification by using fine-grained source code changes
and a pre-trained deep neural language model.
Information and Software Technology 135 (2021), 106566.
DOI:http://dx.doi.org/10.1016/j.infsof.2021.106566
[9] A. B. De Oliveira, S. Fischmeister, A. Diwan, M. Hauswirth, and P. F. Sweeney,
"Perphecy: Performance Regression Test Selection Made Simple but Effective,"
in 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST), Mar. 2017, pp. 103–113. doi: 10.1109/ICST.2017.17.
[10] Deema ALShoaibi, Hiten Gupta, Max Mendelson, Ilyes Jenhani,
Ali Ben Mrad, and Mohamed Wiem Mkaouer. 2022. Learning to Characterize Per- formance Regression Introducing Code Changes.
In The 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22), April 25–29, 2022, Virtual Event, . ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3477314.3507150