Study and Analysis into the Use of History-Based Dynamic Validation Studies for Demand Properties in Regression Testing

Appari pavan kalyan[1], Dr.Harsh Lohiya[2]

[1]Research Scholar, Dept. of Computer Science and Engineering, Sri Satya Sai University of Technology and Medical Sciences, Sehore Bhopal-Indore Road, Madhya Pradesh, India.
[2]Research Guide, Dept. of Computer Science and Engineering, Sri Satya Sai University of Technology and Medical Sciences, Sehore Bhopal-Indore Road, Madhya Pradesh, India.

Abstract
Regression testing is critical, but it is also expensive and time-consuming to do. It's important to prioritise test cases in order to speed up the testing process because there are only so many resources. Conventional approaches to deciding which test cases should be prioritised emphasise one-time testing and ignore the enormous quantities of historical data that regression testing provides. Using historical data to rank test cases is proposed as part of this work. History-based testing prioritises test cases based on requirements, and the historical data is used to decide the test case priorities in regression testing after the test cases have been established. A real-world application will be used to evaluate the efficacy of our methodology. This will be an empirical study. According to our tests, the technique we've presented using average fault detection rates and fault detection rate improves overall performance in real-world settings.

INTRODUCTION

When making changes to a piece of software, programmers often want to make sure that any code that hasn't been changed hasn't been hurt in any way. If the original code is changed in any way, this is called a regression error. Software developers often use regression testing to look for certain types of regression problems. The easiest way to do regression testing is to run all of the previous tests again. Even though this strategy is easy to put into place, it may be too expensive if it only affects a small part of the system. A different method, called regression test selection, has been proposed (RTS)[1]. Using this method, you can only run a subset of the tests again. Because it's hard to choose the best set of test cases, regression testing research and practise focus on the tradeoffs between cost and utility of RTS techniques (i.e., those that will expose the most faults). But we don't know very much about how these methods work. When studying this topic, researchers often use base and modified versions of a system and the test suites that go with them. Then, they use algorithms to choose tests and compare the size and effectiveness of the tests chosen to those of the original tests. There are two big problems with this method: Regression testing is seen as a one-time event instead of a continuous process, so real-world limitations on time and resources are not taken into account. Regression testing is better described as an ordered series of testing sessions, each of which has limited time and resources and whose performance may depend on how well earlier testing sessions worked. Based on these facts, it seems reasonable to make two guesses. One of them is that the perfect regression tester should be able to do a lot more than the real one can. Our recent research shows that the number of changes made between testing sessions has a big effect on how well different RTS approaches work. When more than two or three changes were made to the subject programmes, one method, called a "safe technique" [2], was often used to choose almost all test cases. So, if this happens in real life, RTS techniques can't be used as they were meant to be (such as when a system is being heavily modified). Instead, the RTS-selected test cases must be cut down even more so that they can be run with the restrictions given. This process is called "test case prioritisation." But there is a second way that past test case performance data could be used to improve the long-term performance of regression testing. At the moment, there is no way to remember how RTS or prioritising turned out. Only information from the current and previous software versions' source code and test case profiling can be used to make these reports. This seems to think that local knowledge can guarantee long-term performance. On the other hand, setting priorities cancels out the guarantees of RTS, making the results of regression testing hard to predict. So, if test case prioritisation is needed, we need to think about how new methods that don't just rely on local knowledge might be helpful.... In this post, we'll start to find out more about these ideas. We care a lot about how well different RTS strategies work when time and resources are very limited. The heuristic we describe and test is a simple one that sorts test cases by how important they are based on data from the past. Our hypothesis is that using these heuristics could make regression testing faster and cheaper in situations where development time is limited. If this hypothesis is true, integration and regression testers could save time and money by better managing and coordinating their testing tasks. So, to see how well this idea works, we made and ran an experiment. In the rest of this publication, we talk about our experiment's design, analysis, and results, as well as possible areas for future research. This comes after our review of the literature.

RELATED WORK

It is imperative that software systems are retested whenever a new platform or environment is introduced to ensure that their quality is not compromised. It has been updated, thus some of the test cases have become obsolete and others are more significant now than they were before. Using regression testing, this issue can be resolved.X. Wang et al[3] In this line of research, a proposal is made to rank the importance of test cases according to historical data. Testing is extremely dependent on requirements; in our history-based method, test case priorities are initially set based on the priorities of the requirements, and during regression testing, these priorities are then dynamically determined based on historical data.

Khatibsyarbini, et al[4] It is possible to make certain that a programme is of excellent quality by putting it through a software testing process. Software testing, on the other hand, is a time-consuming and hence expensive phase in the development process. It is possible to increase software testing efficiency by prioritising the execution order of test cases, especially during regression testing.

Khatibsyarbini, et al[5] The process of testing software allows for the possibility of providing a quality assurance for the product. On the other hand, the period of testing software requires a significant investment of time and money. Performance during software testing can be improved, particularly during regression testing, by prioritising the scheduling of the execution order of test cases. This can be done using a prioritisation technique.

Md. Abdur Rahman et al[6] Regression problems can be found early via test case prioritisation. When a new version is released, both the old and new test cases are run for regression testing. This expensive method adds test cases to regression testing. Test cases must be changed to find the most defects in the quickest time. Most prioritisation strategies use source code coverage, needs grouping, etc. These strategies rely on test case similarities. Similarity-focused approaches can get stuck in a local minimum. This research provides a dissimilar clustering-based approach that uses historical data analysis to identify problematic items. It's usual to execute different test cases before related ones.

PROPOSED METHODOLOGY

A strategy for testing needs to be developed according to a set of parameters that have already been decided upon. The first thing that has to be done is to set up a database that will hold all of the code modules and test cases for the project. The definitions of these modules take into account not only the alterations that occur within a module but also the module's connections to the other modules. Because of this examination of module interaction, the selection of a test case for regression testing must adhere to certain parameters[7]. The next thing that needs to be done is, depending on the test cases that were defined, figure out how crucial the module is. The criticality will be determined after taking into account the amount of defects as well as the particular kinds of flaws. Together, these two parameters will be used to determine the relative cost of the development of test cases. The importance of the test and the significance of its code would determine how high it would be on the priority list. In the end, a method known as dynamic programming will be applied in order to generate the test sequence. For the purpose of determining the appropriate sequence, testing will consume the smallest possible sum of money. The quality of any project should be maintained[8] or improved as it is the single most critical thing that can be done to ensure a project's success. It is important that something be completed within the allotted amount of time and money. Testing the software is the phase that consumes the most time and results in the most financial investment. Even if there isn't enough time or money, it has to be done repeatedly after it has already been done. Testing the system is the final stage that must be completed before a software product can be handed over to a customer. In order to determine whether or not the software satisfies the criteria of the customer, testing is performed on the finished, integrated, and operational product. Therefore, this is the final opportunity to test whether or not the system[9] that was built functions in the manner that the customer desired. Testing a full integrated system, which consists of hardware, an operating system, and features that need to be coordinated with one another, takes time and costs money. This is because testing an integrated system involves all of these components. The greater the number of components or settings involved in this procedure, the more complicated it will become. When modifications are made to the software system, those modifications should not have an effect on the code that has not been modified. When something fails to work as it should, this is referred to as a "regression error." Finding and correcting faults that were introduced as a result of previously implemented changes is referred to as "regression testing." In order to test for regression in the most natural way possible, you should run all of the existing test cases once more. This is both expensive and unnecessary, particularly considering that only a small portion of the software has been modified. The number of test cases (TCs) that are necessary[10] to cover all of the configuration options for a piece of software could be anywhere from hundreds to millions. It is impossible to conduct tests on all of these different TCs since doing so would take too much time and cost too much money. Software firms were concerned about the lack of time and resources, which made it difficult to do system testing effectively and maintain a high level of quality and reliability in their products. This caused the companies to worry. As a result of this, there needs to be a method that maintains the quality of the product while also reducing the cost of testing. In order to cut costs associated with testing, various strategies such as test case selection, test case minimization, and test case prioritisation (TCP) were developed. Test case selection methodology is a method that reduces the expenses associated with testing by selecting a few good subsets of test cases from the test suite based on information regarding the programme, changes performed, and the test suite [11]. The reduction of a test suite to a more manageable size while maintaining the same level of coverage as the full test suite is the goal of test suite minimization, which results in cost savings associated with testing. But there are a few issues with both of these approaches. For instance, some empirical evidence demonstrates that test suite minimization has the same rate of detecting flaws as the original test suite, whilst other empirical information demonstrates that test suite minimization makes it more difficult to find bugs than the original test suite did. Test case selection strategies have the same problem. There are methods for selecting test cases that are both effective and secure, and these methods can detect defects just as effectively as the original test suite. However, this approach is not applicable to all types of software. Because these two methods of decreasing costs get rid of their TCs, it is possible that certain TCs will not have their potentially serious problems discovered. Because of this, the overall quality of the programme that is developed suffers. The TCP approach sorts the TCs in descending[12] order of priority, with the ones having the highest priority in terms of their PGs being executed first, followed by the ones having the lowest priority. The challenges associated with test case selection and minimization are circumvented by TCP's refusal to dispose of test cases (TCs). In addition, if the testing procedure had been terminated earlier, the time that was spent testing would have been time well spent because the test cases that had successfully fulfilled their PGs would have been run first, followed by the test cases that had a lesser priority. The TCP approach can be utilised for both testing at the beginning of the process as well as regression testing.

History-based Approach

A matrix analysis is used to establish relationship clusters between software artefacts, and singular value decomposition is a technique for prioritisation that is based on these clusters. In order to decide which test cases should be prioritised, it is necessary to take into consideration three different factors: association clusters, a modification vector, and the connection that exists between the test cases and the files. A change matrix is utilised in the application of the SVD technique that is used to generate association clusters[13]. When two files are regularly modified together to correct a fault, a phenomenon known as association clustering might arise. Additionally, the file is connected to tests that either have an effect on it or are carried out by it. A recent change to the system is now reflected by a vector, and within that vector, each value indicates whether or not a particular file has been modified by indicating whether or not it has been changed. This was done for the sake of thoroughness. As a consequence of this, a priority can be assigned to each file based on how closely the newly introduced change matches each test case, and this is made possible by the association clusters and the modification vector. To accomplish this objective, you can consider use the modification vector in conjunction with the association clusters. The concept that any software artefact[14] can be taken into consideration for priority is one of the innovative aspects that distinguish this method from others. According to Sherriff and his colleagues, faults in non-source files, such as those found in media or documentation, can be just as detrimental as those found in source code. Source code errors are the most common type of file defect.

Coverage-based Techniques

This measurement is referred to more frequently by its colloquial moniker, "test coverage analysis," than by its more accurate name, "code coverage analysis," among software testing professionals. This metric is what puts the code of a programme through its paces while the testing part of the validation is being performed, and it measures that. In the context of software testing, the absence of any third-party applications or resources is referred to as "white boxing." The following is a list of tools and methods that have been mentioned in the coverage: checking to see if the application has been tried out on a particular set of test scenarios and finding that it hasn't creating extra test cases with the goal of expanding the scope of the investigation (d) The process of deriving a quantitative metric of code coverage by locating test cases that are redundant and do not contribute to increased coverage, which is an indirect sign of quality; The term "white-box testing" refers to testing methodologies that place an emphasis on the underlying structural components. A comparison is made between the behaviour of the test programme and the apparent intent of the source code using structural testing. Functional testing, which is sometimes known as "black-box" testing, is concerned with comparing the behaviour of a test programme to a set of criteria. Functional testing is also often called as "white-box" testing. In the course of its analysis, it takes into consideration any structural or logical problems that may exist within the software. During functional testing, the only thing that matters is what a piece of software is capable of achieving, not how it operates in the background. The "coverage based methodology" group is responsible for assigning priority to test cases based on criteria such as requirement coverage, total requirement coverage, additional requirement coverage, and statement coverage.

Cost effective-based techniques
consist of techniques for prioritizing[15] test cases according to expenses, such as the cost of analysis and the cost of prioritisation. This subject has been investigated by a great number of researchers. Existing strategies for cost-effective and based test case prioritising will be discussed in the following paragraphs. The amount of resources needed to execute and validate a test case is directly proportional to the case's associated cost. In addition to this, cost-cognizant prioritising calls for an evaluation of the gravity of each flaw that a test case is able to uncover. The total function coverage prioritisation (fn-total), the additional function coverage prioritisation (fn-addtl), the total function difference-based prioritisation (fn-diff-total), and the additional function difference-based prioritisation are the four practical code coverage-based heuristic techniques (fn-diff-addtl). models of costs for prioritisation that take into account these various costs. In order to prioritise test cases, they developed the following variables: cost of analysis (abbreviated as Ca(T)), and cost of the prioritising algorithm (abbreviated as Cp) (T). WP equals Ca(T) plus Cp (T) WP is a weight prioritisation value that is assigned to each test case. • The cost of an examination of the source code, an analysis of the differences between the old and new versions, and the collecting of execution traces are all included in Ca(T). • Cp(T) is the real cost of operating a prioritisation tool, and depending on the prioritisation algorithm that is being used, this cost can be calculated during either the preliminary phase or the critical phase.

This paper presents a case study of the implementation of history-based regression testing in a big software development organisation. The goal of this work is to increase test efficiency and transparency at the function level. An automated tool has been developed as a direct result of the findings of this study. This tool combines the concepts of historical regression[16] analysing principles from literature with best practises from the current process and the opinions of practitioners. The currently implemented experience-based strategy, along with two systematic approaches, are subjected to a post hoc quasi-experiment in order to facilitate comparison. In order to further evaluate the tool's performance, practitioners do manual assessments of the output it generates. The purpose of using history-based prioritisation is to improve the effectiveness of a test suite by taking into consideration the characteristics of earlier iterations of test case executions. Practitioners first identified historical efficacy [17] and execution history as essential components to include in a tool. Our strategy was put into practise and then assessed, but the practitioners stopped using the execution history as a basis for prioritisation. The non-historical aspects that are being stressed are working toward achieving confidence rather than efficiency. A history-based strategy to prioritising test cases was demonstrated to be beneficial, but it did not account for all of the essential characteristics that needed to be considered. The test management system is intuitive to operate and stores all of the relevant historical data within its database. We provide the findings of a case study that was conducted on the implementation of history-based regression testing in a big software development company with the intention of improving test transparency and efficiency at the level of function tests [18]. A semi-automated tool is being built based on a combination of historical conceptions from the literature regarding regression testing, good practises from the procedure that is currently being used, and practitioner opinions. In this experiment, a comparison is made between the currently used experience-based strategy, two systematic approaches, and a post hoc quasi experiment. Practitioners undertake their own in-depth reviews of the output of the instrument, in addition to the automated evaluations that are performed. The purpose of using history-based prioritisation is to improve the effectiveness of a test suite by taking into consideration the characteristics of earlier iterations of test case executions. The practitioners came to the conclusion that it is essential for a tool to possess the following characteristics: historical effectiveness and execution history. After the implementation and evaluation of our tool[19], practitioners no longer used execution history as a basis for prioritisation. The non-historical aspects that are being stressed are working toward achieving confidence rather than efficiency. Even though it did not take into consideration all of the important parameters, the history-based method [20] for prioritising test cases was found to be effective in achieving the desired results. The test management system is intuitive to operate and stores all of the relevant historical data within its database.

DISCUSSION

For the objective of increasing test efficiency and transparency, we conducted a case study on the implementation of history-based regression testing within a large software development organisation. As a result of the investigation into current practises, a semi-automated tool has been developed, which incorporates ideas from the literature about history-based regression testing with best practises discovered during the current process and the opinions of practitioners. Post hoc quasi experiments are used to investigate three distinct strategies: the existing strategy, which is based on experience; two systematic procedures; and the way now employed. Practitioners do manual evaluations of the instrument's results in order to verify their accuracy. The features of previous runs of particular test cases are taken into consideration by history-based prioritisation in order to increase the overall effectiveness of a test suite's testing. Practitioners initially identified historical efficacy and execution history as two such qualities that must be integrated into a tool. Execution history was no longer used as a basis for prioritisation following the implementation and evaluation of our solution by practitioners. The non-historical traits that were discovered are intended to increase trust rather than to improve the efficiency of the testing procedure. Even though [21] had created a history-based technique for prioritising test cases, it was deemed to be a valid basis for prioritising test cases. Easy-to-use test management software provides access to previous data necessary for analysis.

In light of the two predetermined research areas, we have arrived at the following conclusion:
RQ1. How do you determine which test cases to prioritise and pick for regression testing? Prioritizing and selecting regression test cases is determined by a number of criteria, including the following: The session's Scope and Focus, in addition to the session's Time and Resource Constraints, serve as the basis for selecting test cases that are historically effective, have a long history of execution, and have a high static priority. This selection process also takes into account the session's Time and Resource Constraints. Test cases are prioritised according to the function areas that they are a part of as a part of the effort to cover as many functionalities as is humanly practical. The present status (such as blocked or postponed) of a test case is used to

filter out invalid test cases, while the status of test cases from previous sessions is used to determine the current priority of test cases.

RQ2. Does picking test cases and putting them in order of importance based on the project's history make regression testing better? By automating as much of the selection process as possible, we can make it more open and clear. This kind of automation should combine things that are specific to the situation with good ideas that are well-known in the research literature. But the details of the proposed methods are less important because they will have to be changed to fit the current process, the tools available, and the way people act. It's not possible to include every part of the selection process in a tool, so there also need to be rules about how to use the tool. This is because you can't put every part of the selection process into a single tool. It's possible that the tool will help some regression test sessions more than others (e.g. if for example the focus is on problem areas). Even if there is a tool, the chosen test suites still need to be looked at by hand to find any gaps in coverage and get familiar with any important test scenarios. In the quantitative analysis we did, we found that putting a history-based priority on a test suite that had already been chosen made it easier to find flaws early on. This is helpful in case a test session was cut short before it was done. Much more important is how well a method of choosing is based on putting things in order of importance. Our data doesn't give us enough information to say for sure, but it looks like the efficiency of finding faults is getting better, and in the worst case, it's getting a little worse. This is because we don't have enough information to make a precise assessment. The history-based method was improved by adding parts that took into account the age of test cases and their static priority, two more factors for which data was available in the test management system. The history of test cases and static prioritisation were the two extra things. Because of these changes, the testers had more faith in the test suites. However, the extensions had no effect on how well the prioritising or selection processes worked. In a nutshell, testers liked the big ideas of history-based prioritisation and putting best practises into a tool, but they didn't like the details of this particular implementation. Many of the problems that have come up can be fixed by making changes to how the implementation is done. If there was a link to the error reporting system, at least some of the most important things that haven't been done yet could be done. On the other hand, some problems with the tool can't be easily fixed because the test management database may have information that is out of date or doesn't exist.

CONCLUSION

This study looks at test case prioritisation and other strategies for regression testing. In this study, we will talk about how to prioritise test cases and what important contributions they have made. Prioritization strategies are also argued over in terms of how they should be put into groups. Three of the most common ways to set priorities are based on the type of code coverage data, how feedback is used, and how code has changed. Some of the most common ways to order tasks are by coverage, fault, code change, code analysis, tester input, and programme data. There are also greedy, cost-aware, and history-based approaches. In these methods, coverage and finding flaws are often done at the same time. Since finding bugs is the main proxy for all prioritisation strategies, the focus moves from code analysis to prioritisation based on history. It is clear that the industry prefers to use artificial prioritisation techniques over coverage-based ones, and research in this area shows that coverage-based techniques produce less positive results than artificial ones, especially when it comes to smaller-scale programmes and artificial seeds for them. On a larger scale, more research is needed to compare both artificial and coverage-based methods so that the differences between them can be seen more clearly. We've seen that some artificial procedures lose their effectiveness when the size of the programme goes over a certain threshold, and that the effectiveness goes down as the size of the programme goes up.

REFERENCE

[1]. Li, K., Wu, G. Randomized approximate class-specific kernel spectral regression analysis for large-scale face verification. Mach Learn 111, 2037–2091 (2022). https://doi.org/10.1007/s10994-022-06140-9

[2]. Abolghasemi, M., Hyndman, R.J., Spiliotis, E. et al. Model selection in reconciling hierarchical time series. Mach Learn 111, 739–789 (2022). https://doi.org/10.1007/s10994-021-06126-z

[3]. X. Wang and H. Zeng, "History-Based Dynamic Test Case Prioritization for Requirement Properties in Regression Testing," 2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED), 2016, pp. 41-47, doi: 10.1145/2896941.2896949.

[4]. Khatibsyarbini, Muhammad; Isa, Mohd Adham; Jawawi, Dayang N.A.; Tumeng, Rooster (2017). Test case prioritization approaches in regression testing: A systematic literature review. Information and Software Technology, (), S0950584916304888–. doi:10.1016/j.infsof.2017.08.014

[5]. Khatibsyarbini, Muhammad, et al. “Test Case Prioritization Approaches in Regression Testing: A Systematic Literature Review.” Information and Software Technology, vol. 93, Jan. 2018, pp. 74–93, 10.1016/j.infsof.2017.08.014.

[6]. Md. Abdur Rahman, Md. Abu Hasan and Md. Saeed Siddik. Prioritizing Dissimilar Test Cases in Regression Testing using Historical Failure Data. International Journal of Computer Applications 180(14):1-8, January 2018.

[7]. X. Zhao, Z. Wang, X. Fan, and Z. Wang, “A clusteringbayesian network based approach for test case prioritization,” in Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual, vol. 3, pp. 542–547, IEEE, 2015.

[8]. L. W. Yee, N. A. A. Bakar, N. H. Hassan, N. M. M. Zainuddin, R. C. M. Yusoff and N. Z. A. Rahim, "Using Machine Learning to Forecast Residential Property Prices in Overcoming the Property Overhang Issue," 2021 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET), 2021, pp. 1-6, doi: 10.1109/IICAIET51634.2021.9573830.

[9]. C. Guici and L. Yiyun, "The empirical analysis on demand of property insurance in Hubei Province based on panel data," 2017 29th Chinese Control And Decision Conference (CCDC), 2017, pp. 6378-6383, doi: 10.1109/CCDC.2017.7978319.

[10]. M. Bai and R. Zhao, "The Test and Calculation of the Underwriting Cycle in Property-Liability Insurance of China," 2007 International Conference on Computational Intelligence and Security Workshops (CISW 2007), 2007, pp. 600-603, doi: 10.1109/CISW.2007.4425567.

[11]. F. S. Ismail, Noor Elaiza Abd Khalid, N. A. Bakar and R. Mamat, "Optimizing oil palm fiberboard properties using neural network," 2011 3rd Conference on Data Mining and Optimization (DMO), 2011, pp. 271-275, doi: 10.1109/DMO.2011.5976540.

[12]. G. Chen and L. Ge, "Comparison of Forecasting Methods for Aspect Ratio of Wood Tracheid," 2010 2nd International Workshop on Database Technology and Applications, 2010, pp. 1-4, doi: 10.1109/DBTA.2010.5658959.

[13]. “A Dissimilarity with Dice-Jaro-Winkler Test Case Prioritization Approach for Model-Based Testing in Software Product Line.” KSII Transactions on Internet and Information Systems, vol. 15, no. 3, 31 Mar. 2021, 10.3837/tiis.2021.03.007. Accessed 26 Mar. 2022.

[14]. G. Chen and L. Ge, "Comparison of Forecasting Methods for Aspect Ratio of Wood Tracheid," 2010 2nd International Workshop on Database Technology and Applications, 2010, pp. 1-4, doi: 10.1109/DBTA.2010.5658959.

[15]. Jianxin Zhou, Yinxin Ji, Zongliang Qiao, Fengqi Si and Zhigao Xu, "Nitrogen oxide emission modeling for boiler combustion using accurate online support vector regression," 2013 10th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), 2013, pp. 989-993, doi: 10.1109/FSKD.2013.6816339.

[16]. Žliobaitė, I., Hollmén, J. Optimizing regression models for data streams with missing values. Mach Learn 99, 47–73 (2015). https://doi.org/10.1007/s10994-014-5450-3

[17]. Tsamardinos, I., Borboudakis, G., Katsogridakis, P. et al. A greedy feature selection algorithm for Big Data of high dimensionality. Mach Learn 108, 149–202 (2019). https://doi.org/10.1007/s10994-018-5748-7

[18]. Mukhoty, B., Dutta, S. & Kar, P. Robust non-parametric regression via incoherent subspace projections. Mach Learn 110, 2941–2989 (2021). https://doi.org/10.1007/s10994-021-06045-z

[19]. Osojnik, A., Panov, P. & Džeroski, S. Incremental predictive clustering trees for online semi-supervised multi-target regression. Mach Learn 109, 2121–2139 (2020). https://doi.org/10.1007/s10994-020-05918-z

[20]. Benavoli, A., Azzimonti, D. & Piga, D. A unified framework for closed-form nonparametric regression, classification, preference and mixed problems with Skew Gaussian Processes. Mach Learn 110, 3095–3133 (2021). https://doi.org/10.1007/s10994-021-06039-x

[21]. Ribeiro, R.P., Moniz, N. Imbalanced regression and extreme value prediction. Mach Learn 109, 1803–1835 (2020). https://doi.org/10.1007/s10994-020-05900-9