A Survey on Machine Learning Techniques for Software Engineering

J. Asaad, E. Y. Avksentieva

ITMO University, St. Petersburg, Russia

Abstract. Machine learning (ML) environments offer a variety of methods and tools that help to solve problems in different areas, including software engineering (SE). Currently, a large number of researchers are interested in the possibilities of using various machine learning techniques in software engineering. This paper provides an overview of machine learning techniques used in each stage of the software development life cycle (SDLC). The contribution of this review is significant. Firstly, by analyzing sources from bibliographic and abstract databases, it was found that the topic of integrating machine learning techniques into software engineering is relevant. Secondly, the article poses questions and reviews the methodology of this research. In addition, machine learning methods are systematized according to their application at each stage of software development. Despite the vast amount of research work on the use of machine learning techniques in software engineering, further research is required to achieve comprehensive comparisons and synergies of the approaches used, meaningful evaluations based on detailed practical implementations that could be adopted by the industry. Thus, future efforts should be directed towards reproducible research rather than isolated new ideas. Otherwise, most of these applications will remain poorly realized in practice.

Keywords: machine learning, software engineering, software development life cycle.

DOI 10.14357/20718632230408 **EDN** ADARZK

Introduction

Nowadays, it's difficult to pass a day without seeing or reading an article on machine learning (ML), data mining, big data analytics, artificial intelligence (AI), and the profound changes they are bringing to society, particularly after releasing ChatGPT in November 2022.

Machine learning deals with the issue of how to build programs that improve their performance through experience. Machine learning algorithms have proven to be of great practical value in a variety of application domains. Machine learning has been successfully applied in many areas of software engineering, ranging from features extraction to testing to bug fixing. If software developers had a better grasp of machine learning approaches, their assumptions, and guarantees, they might adopt and select the best techniques for their intended applications. To meet the needs of changing approaches to software development, future software engineering (SE) techniques and tools will need to be much more automated, lightweight, adaptable, and scalable to keep pace with increased developer productivity. The increasing reliance on applications with machine learning (ML) components calls for mature engineering techniques that ensure these are built in a robust and future-proof manner.

Furthermore, software is an indispensable component of the majority of systems and is integrated into the daily lives of society. With the advancement of technologies such as open systems and highly automated or networked devices, software systems are becoming very complex [1]. Additionally, several people from different areas of expertise are usually required to be involved in a software project, which

also increases its complexity. Since software is developed by humans, it is usual that people make mistakes; thus, in every commercial piece of software, some errors always occur [2], and as the level of complexity increases, these error ratios become even higher [3]. Automating the SDLC process through machine learning may solve all these problems. Consequently, we will try to detail several software engineering cases where machine learning has already been attempted effectively.

1. The research question and methodology

The aim of this study is to find out which machine learning methods have been used in the software development life cycle and their performance. This research will enable us to recognize the shortcomings that should be considered to enhance the efficiency of these methods.

The following questions inspire this research:

RQ1: What categories of software applications were found or reported in the current phase?

RQ2: Which ML algorithms have been utilized in this phase?

RQ3: How did ML-based techniques perform? Do ML-based techniques perform better than non-ML-based ones?

Structuring the literature review involved breaking down the overall task into several smaller steps so as to enable us to explore the literature for systematically extract relevant information. Firstly, key search strings were utilized: 'Machine learning for software engineering', 'Machine learning + software engineering', 'Machine learning for SDLC', 'Machine learning + (and | for | +) + software requirement', 'Machine learning + (and | for | +) + software design', 'Machine learning + (and | for | +) + software testing', 'Machine learning + (and | for | +) + software construction' and 'Machine learning + (and | for | +) + software maintenance' so as to identify a baseline set of research papers. Google, Google Scholar, and digital libraries of publications from ACM and IEEE were used to find these publications.

Following the completion of the initial phase of the literature review, the shortlists for each search term were further evaluated. The relevance of publications was examined by reading each abstract and conclusion. Each publication was sorted according to the number of citations it had and the year it was published. The next step of the process involved reading the publications in detail and making further evaluations in relation to their relevance. Overall, the results of this systematic approach are present in Section 3.

2. Background and related works

The interaction between software engineering (SE) and machine learning (ML) has been studied by researchers for a long time [1-3]. The first Symposium on Software Engineering for Machine Learning Applications (SEMLA) at Polytechnique Montréal was organized on 12 and 13 June 2018, with the support of Polytechnique Montréal's Department of Computer Engineering and Software Engineering, the Institute for Data Valorization (IVADO), SAP, and Red Hat. Around 160 participants from 160 different countries attended the event, including students, professors, and professionals from the business sector.

On the other hand, other studies draw attention to the gap between the communities of SE and ML. The focus of these groups may be one factor in this separation, the ML community is concerned with algorithms and their performance, while the SE community is concerned with developing and deploying software-intensive systems [4].

Two areas of synergy are revealed when the expertise and experience of these two communities are combined:

The phrase "SE for ML" refers to tackling numerous SE responsibilities for engineering ML systems, such as designing, creating, and maintaining software systems that support ML. Researchers are attempting to pinpoint the distinctions between designing ML systems and conventional software in order to create new strategies and tools to address these disparities.

In contrast, the phrase "ML for SE" refers to applying or adapting AI technologies to address various SE tasks, such as software fault prediction, code smell detection, reusability metrics prediction, and cost estimation , etc. Researchers utilize ML models obtained from SE data (source code, requirement specifications, test cases, etc.) to engineer software more efficiently and effectively.

Machine learning (ML) is a branch of research that offers computers the capacity to learn without being explicitly programmed. It was first described by Arthur Samuel in 1959.

The term “software engineering” was created in 1972 by David Parnas. Software engineering is defined by the IEEE as “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software”. Software development methodology is the process that uses organizational processes to carry out the required procedures for the analysis, design, implementation, and maintenance of information systems. Software projects must be well prepared and planned for in order to succeed and produce high quality software on schedule.

The software development life cycle (SDLC) is the main element in software development. It is the entire process of building any software product. There are various types of SDLC methodologies, for instance, Agile, Waterfall, DevOps, V-Model, Iterative, Dynamic System Development Model, Extreme Programming, Feature Driven Development, Joint Application Development, Spiral, Rapid Application Development, and Lean methodology.

Generally, the phases that constitute the software development life cycle include requirements analysis, design, implementation, testing, and maintenance [5].

At the present time, software engineering has transitioned from traditional waterfall models to agile software development. A waterfall model is a sequential process where the success of each stage depends on the success of the previous stages. All requirements are thought to be clearly established at project inception and essentially stable after that. Agile processes are iterative software development techniques that offer adaptability and flexibility in response to changing conditions while producing high-quality software. It emphasizes informal, adaptable project management that will improve communication and transparency.

This paper focuses on machine learning for software engineering by systematically reviewing the machine learning literature for software development tasks.

3. Machine learning for software engineering

As we mentioned, the software development life cycle (SDLC) consists of five phases [5]. In this section, we will discuss the research questions for each phase of the SDLC.

3.1. ML for requirement engineering

Throughout the software development process, requirement engineering (RE) is essential. Prioritization and requirement identification are the key stages of the RE process [6].

RQ1: The ML-based techniques were used for the identification of different types of software requirements: functional requirements (FRs) [7-11] and nonfunctional requirements (NFRs) [12-18].

RQ2: Although there are a multitude of different machine learning algorithms and approaches available for text processing, they may be broadly divided into two groups: supervised learning algorithms (SL) and unsupervised learning algorithms (USL). In addition, between supervised and unsupervised learning algorithms, there is another form termed semi-supervised learning (SSL).

The outcome of this RQ revealed that there were mainly two different machine learning algorithms identified in the selected primary research studies. These machine learning algorithms basically fall into two types: SL and USL ML algorithms.

Besides, we found that there are primary studies that used thematic analysis or qualitative coding techniques. As well, the selected studies depict that USL algorithms, specifically Latent Dirichlet Allocation (LDA), are the most popular type of machine learning algorithm. Support Vector Machine (SVM) was the second-most popular ML algorithm category. It was quite interesting to observe that a few of the selected primary studies did not use any algorithm and used thematic coding or qualitative coding techniques for identifying and classifying the different types of software requirements.

The three primary parts of the process might be categorized as follows:

- text preprocessing, which involves removing any unnecessary text from the data.
- applying the various ML algorithms is basically what the learning step entails.
- analyzing or evaluating an ML algorithm's methodology.

The chosen research articles revealed a total of six alternative Natural Language Processing (NLP) preprocessing methods. The following is a quick explanation of the many preprocessing methods found.

Stop words removal is the act of eliminating specific auxiliary verbs from the text, such as "be," "do," and "have," as well as distinct articles such as

"the," "a," and "an" [19]. Splitting a statement into words is a technique known as tokenization [20].

The act of making the text into a standard style, such as lowercase or uppercase, is known as case unification. Stemming is the process of taking a term and reducing it to its base or origin. For instance, terms like "goes," "gone," and "going" will all be shortened to "go." [19]. Punctuation removal is the process of removing different punctuations like commas, semicolons, question marks, and exclamation marks. On the other hand, some studies averaging lack reporting about using any of the preprocessing step's majorities of the chosen primary research studies regard machine learning-based techniques as 'black boxes,' providing no specific description of how these techniques truly operate.

RQ3: Not every study that was chosen has tested its performance evaluation criterion. Although the LDA and SVM were utilized in several different research articles, it was quite significant to observe that their performance results considerably varied from one another. For instance, the LDA algorithm performs well in one study [16], nevertheless, it does not perform that good in another study [17].

3.2. Software Design

In the software development life cycle, it is the most inventive phase. This phase's objective is to arrange or plan the required definition. It is the planning and issue-solving process for a software solution. It involves software designers and developers specifying the strategy for a fix. This phase results in a software design document (SDD).

Software design is a highly complex and challenging activity. Nevertheless, using software design patterns makes this phase more organized. A software design pattern can be defined as a presupposed structure of classes organized and interacting in a particular manner to solve a recurring design problem.

RQ1: The studies show that ML is able to be used to avoid some problems in this phase, for instance, detection of the bad smells earlier [5], meaning detecting symptoms that the system's design or programming may be flawed [21]. As well, ML-based techniques are able to be used in design pattern recognition (adapter, strategy) [22]. Furthermore, some studies experimented with five design patterns (Singleton, Adapter, Composite, Decorator, and Factory Method) [23].

Some types of SDLC, for instance, Agile, divide the architecture of a system into components. Consequently, the selection of software components is part of the design phase. Some studies suggest a novel approach to machine learning [24], which can assist in the selection of reusable software components.

RQ2: The following machine learning models have been used for experiments in the selected studies: logistic regression, random forest, IBk [5], neural network and decision tree [22], zero, one, Nave Bayes, JRip, C4.5, SVMs (with different kernel functions), simple KMeans, and CLOPE [23].

The suggested machine learning approach to selecting reusable components combines the Decision Tree and Neural Network modules to determine the more accurate and suitable object of the software design pattern, which may help with efficient package reuse [24].

RQ3: The authors mentioned that the selected algorithms perform differently in terms of processing speed and classification accuracy [5], and they inform that Naive Bayes, Logistic regression, IB1, IBk, Random Forest have better performance than the VFI and J48.[5]. For instance, the authors in [22] inform us that the learning precision of the formulated approach is 67–95%

Generally, the ML-based techniques performed well in this phase. Nonetheless, the results are not compared with other traditional techniques (non-ML-based techniques). It's considerable to observe that although the researchers made an effort to provide impartial results, there may still be some degree of subjectivity, as long as all results are related to the construction of the training set, which is based on a manual design pattern labeling task.

3.3. Software construction

This phase involves turning the software design document into code using a programming language. It results in program code; thus, it is the logical one.

RQ1: The studies show that ML models are used for code generation [25-26], documentation generation [27–28], and code modification [29–31]. The popular models for converting ideas into code are ChatGPT, Codex, and Alphacode. ChatGPT and Codex are models by OpenAI. It interacts in a conversational way. As it is widely known, ChatGPT answers follow-up questions, challenges incorrect assumptions, and rejects improper demands. In addition, it is able to generate code [32]. Moreover,

Codex is a general-purpose programming model, as it can be applied to any programming task [33].

Additionally, Alphacode is general-purpose programming, it can be applied to programming problems that require for deeper reasoning [34].

RQ2: The selected studies show that a wide range of ML techniques are able to be applied to various code generation tasks. The popular model types used by selected studies include recurrent neural networks [25-28] and convolutional neural networks [30]. ChatGPT is trained using supervision and reinforcement learning (RL). In the supervised learning, human trainers would provide conversations in which they played both sides, the user, and the chat bot side. Then, in the case of reinforcement learning, those people would be given the model-written responses to help them compose their response [32]. This dataset was combined with the Instruct GPT [35] dataset, which was converted to a question-answer format.

As well as Codex based on GPT-3, a neural network trained on text [35], this model has been trained on 179 gigabytes of Python code from software repositories hosted on GitHub projects. At the same time, Alphacode has been trained on 715.1 gigabytes of code on GitHub, in addition to Codeforces problems.

RQ3: In general, the outcomes were not assessed, considering more traditional techniques. Transformer models outperformed RNN models when the two were compared in an evaluative study [36]. Nonetheless, ML models perform imperfectly when evaluated on highly complex, unseen problems [32].

3.4. Software Testing

Testing is defined as "an activity in which a system is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system" (ISO/IEC 24765, 2006) [37].

In the software product development process, software testing is demanded. Any software product must first pass through several different steps before it can be implemented. Testing allows us to identify issues early. Additionally, participating in testing activities gives developers the ability to study the criteria for critical quality aspects, pose queries, and find solutions in advance.

Automation of software testing has been accepted as a realistic technique to get around the complexity and expense of most testing tasks. To find flaws in software systems, testing entails delving into their behavior. Applying machine learning (ML) to different software testing operations has drawn increasing interest [38].

RQ1: Machine learning was applied for statistical software testing [39], performance testing [40], and test case generation [41].

RQ2: Q-learning was used as a model-free RL algorithm in a smart test framework [40]. Furthermore, Model-Inference-Driven testing (MINTest) is used for software test automation [42]. It describes itself as a framework for unit and integration testing on its website [43].

RQ3: The studies show that efficient automated software testing is a challenging activity in software development [40–42]. The resulting test suites greatly improved in terms of defect detection [44].

3.5. Software Maintenance

According to the IEEE Standard, IEEE STD 1219-15193 [45], software maintenance is: "the modification of a software product after its delivery (to the customer), to correct errors, to improve its performance or other attributes, or to adapt the product to a modified environment".

Understanding software maintenance helps practitioners in the industry deal with many of the problems they currently experience, by reducing uncertainty, improving cost-effectiveness, dependability, and other factors [46]. This is the final stage of the SDLC. The software being produced is distributed to end users during this stage of the SDLC, who are then in charge of maintaining and utilizing it in accordance with best practices.

RQ1: The most dominant application of using ML in this phase is bug detection [47–48].

In addition, maintenance software has several forms, for instance refactoring, which includes switching out components or algorithms for more elegant ones, updating data naming standards, and improving the readability or understandability of the code [48]. There are a few studies that discuss building a refactoring model, for instance, An AI-data-based approach to early quality evaluation and enhancement of object-oriented software products was proposed in the paper "A machine learning approach to software model refactoring" [49].

RQ2: Our study shows that a wide range of ML techniques have been applied in this phase. However,

the CNN-based deep learning model is proposed for recognizing duplicate or similar bug reports [47]. Besides, three supervised machine learning algorithms are considered to build the model and predict the occurrence of the software bugs based on historical data by deploying the classifiers logistic regression, Nave Bayes, and decision tree [48].

A deep neural network that learns to detect the existence of functional decomposition in UML models of object-oriented software is used to implement model refactoring [49]. The study's proposed method [48] uses data science techniques to obtain an understanding of multidimensional software design aspects and then applies the knowledge acquired to generalize nuanced interactions between architectural elements.

RQ3: The selected studies don't have very clear and effective evaluation methods. However, some studies show that some algorithms were able to generate 100% accuracy with train and test datasets [48]. On the other hand, the authors mentioned that the selected algorithm is empirically evaluated and shows high accuracy [49]. Furthermore, as with any ML model, the studies ensure that the results depend on the data. For instance, the proposed system in one of the studies provides a high accuracy rate for the same domain datasets and a low accuracy rate for different domain datasets [47]. In addition, the subjective nature of software affects the evaluation process. [49].

Conclusion

In summary, the use of machine learning techniques in the software development lifecycle holds great promise, offering valuable input to its various stages from requirements engineering to software maintenance. ML has demonstrated its effectiveness in solving tasks such as software requirements definition, design problem identification, code generation and test automation. However, there are still major challenges, such as the lack of comparative reliability and productivity analysis with traditional approaches that do not use ML, and the lack of standardized rigorous evaluation methodologies. To fully exploit the potential of ML in software engineering, future research should prioritize reproducible methodologies and rigorous benchmarking to ensure the reliability and usability of these applications at all stages of the SDLC.

References

1. Lwakatare, L.E., Raj, A., Bosch, J., Olsson, H.H. and Crnkovic, I. A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In Agile Processes in Software Engineering and Extreme Programming: 20th International Conference. 2019. pp. 227-243.
2. Shehab M, Abualigah L, Jarrah MI, Alomari OA, Daoud MS. Artificial Intelligence in Software Engineering and inverse. International Journal of Computer Integrated Manufacturing. 2020. vol. 33. no. 10-11. pp. 1129-1144.
3. Durelli, Vinicius HS, et al. "Machine learning applied to software testing: A systematic mapping study." IEEE Transactions on Reliability. 2019. vol.68. no. 3. pp. 1189-1212.
4. Khomh, F., Adams, B., Cheng, J., Fokaefs, M., Antoniol, G., 2018. Software engineering for machine-learning applications: The road ahead. IEEE Softw.2018. vol. 35. no. 5. pp. 81–84.
5. Maneerat N, Muenchaisri P. Bad-smell prediction from software design model using machine learning techniques. In2011 Eighth international joint conference on computer science and software engineering (JCSSE). 2011. pp. 331-336.
6. Talele, P. and Phalnikar, R. Software requirements classification and prioritisation using machine learning. In Machine learning for predictive analysis. 2021. pp. 257-267.
7. Zou, J., Xu, L., Guo, W., Yan, M., Yang, D., and Zhang, X. Which non-functional requirements do developers focuson? An empirical study on stack overflow using topic analysis. In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. 2015. pp. 446–449.
8. Ahmad, A., li, K., Feng, C., and sun, T. "An empirical study on how iOS developers report quality Aspects on stack overflow," international journal of machine learning and computing. 2018. vol. 8. no. 5. pp. 501–506.
9. Treude, C., Barzilay, O., and storey, M. A. How do programmers ask and answer questions on the web? Nier track. In 2011 33rd International Conference on Software Engineering (ICSE). 2011. pp. 804–807.
10. Zou, J., Xu, L., Yang, M., Zhang, X., and Yang, D. Towards comprehending the non-functional requirements through developers' eyes: An exploration of stack overflow using topic analysis. Information and Software Technology.2017. vol. 84. pp.19–32.
11. Ahmad, A., Feng, C., li, K., Asim, S. M., and sun, T. Toward empirically investigating non-Functional requirements of iOS developers on stack overflow. IEEE Access. 2019. vol.7. pp.61145–61169.
12. Yin, H., and Pfahl, D. A preliminary study on the suitability of stack overflow for open innovation in requirements engineering. In Proceedings of the 3rd International Conference on Communication and Information Processing. 2017. pp. 45–49.
13. Bajaj, K., Pattabiraman, K., and Mesbah, A. Mining questions asked by web developers. In Proceedings of the 11th Working Conference on Mining Software Repositories.2014. pp. 112–121.
14. Pinto, G., Castor, F., and Liu, Y. D. Mining questions about software energy consumption. In Proceedings of the 11th Working Conference on Mining Software Repositories. 2014. pp. 22–31.

15. Xiao, M., Yin, G., Wang, T., Yang, C., and chen, M. Requirement acquisition from social q&a sites. In Requirements Engineering in the Big Data Era .2015. pp. 64–74.
16. Rosen, C., and Shihab, E. What are mobile developers asking about? A large-scale study using stack overflow. Empirical Software Engineering.2016. vol. 21. no.3. pp. 1192–1223.
17. Abad, Z. S. H., Shymka, A., Pant, S., Currie, A., and Ruhe, G. What are practitioners asking about requirements engineering? An exploratory analysis of social q&a sites. In 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW). 2016. pp. 334–343.
18. Pinto, G. H., and Kamei, F. What do programmers say about refactoring tools? An empirical investigation of stack overflow. In Proceedings of the 2013 ACM workshop on Workshop on refactoring tools. 2013. pp. 33–36.
19. A. G. Jivani, "A comparative study of stemming algorithms," International Journal of Computer Applications in Technology.2011. vol. 2. pp. 1930–1938.
20. A. Khan, B. Baharudin, L. H. Lee, and K. Khan, "A review of machine learning algorithms for text-documents classification," Journal of Advances in Information Technology.2010. vol. 1, pp. 4–20.
21. Fernandes E, Oliveira J, Vale G, Paiva T, Figueiredo E. A review-based comparative study of bad smell detection tools. InProceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering. 2016. pp. 1-12.
22. Ferenc R, Beszedes A, Fulop L, Lele J. Design pattern mining enhanced by machine learning. In21st IEEE International Conference on Software Maintenance (ICSM'05) 2005. pp. 295-304.
23. Zanoni M, Fontana FA, Stella F. On applying machine learning techniques for design pattern detection. Journal of Systems and Software. 2015. vol. 103. pp. 102-17.
24. Selvarani R, Mangayarkarasi P. A Dynamic Optimization Technique for Redesigning OO Software for Reusability. ACM SIGSOFT Software Engineering Notes. 2015. vol. 40. no. 2. pp.1-6.
25. Agashe R, Iyer S, Zettlemoyer L. Juice: A large scale distantly supervised dataset for open domain context-based code generation. arXiv preprint arXiv:1910.02216. 2019.
26. Shin EC, Allamanis M, Brockschmidt M, Polozov A. Program synthesis and semantic parsing with learned code idioms. Advances in Neural Information Processing Systems. 2019. vol. 32.
27. Takahashi A, Shiina H, Kobayashi N. Automatic Generation of Program Comments based on Problem Statements for Computational Thinking. In2019 8th International Congress on Advanced Applied Informatics. 2019. pp. 629-634.
28. Shido Y, Kobayashi Y, Yamamoto A, Miyamoto A, Matsumura T. Automatic source code summarization with extended tree-lstm. In2019 International Joint Conference on Neural Networks. 2019. pp. 1-8.
29. Tufano M, Watson C, Bavota G, Penta MD, White M, Poshyvanyk D. An empirical study on learning bug-fixing patches in the wild via neural machine translation. ACM Transactions on Software Engineering and Methodology (TOSEM). 2019. vol. 28. no. 4. pp.1-29.
30. Zhu Z, Xue Z, Yuan Z. Automatic graphics program generation using attention-based hierarchical decoder. InAsian Conference. 2018. pp. 181-196.
31. Kim Y, Kim H. Translating CUDA to opencl for hardware generation using neural machine translation. In2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO) 2019. pp. 285-286.
32. Gozalo-Brizuela R, Garrido-Merchan EC. ChatGPT is not all you need. A State of the Art Review of large Generative AI models. arXiv preprint arXiv:2301.04655. 2023.
33. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374. 2021.
34. Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. Competition-level code generation with alphacode. Science. 2022. Vol. 378. No. 6624. pp. 1092–1097.
35. Bhavya, B., Xiong, J., and Zhai, C. Analogy generation by prompting large language models: A case study of instructgpt. arXiv preprint arXiv:2210.04186 .2022.
36. Dehaerne E, Dey B, Halder S, De Gendt S, Meert W. Code Generation Using Machine Learning: A Systematic Review. IEEE Access. 2022.
37. S. Ahmed, "Overview of software testing standard ISO/IEC/IEEE 29119", Int. J. Comput. Sci. Netw. Secur.2018. vol. 18. no. 2. pp. 112-116.
38. Durelli VH, Durelli RS, Borges SS, Endo AT, Eler MM, Dias DR, Guimaraes MP. Machine learning applied to software testing: A systematic mapping study. IEEE Transactions on Reliability. 2019. vol. 68. no. 3. pp. 1189-212.
39. Baskiotis N, Sebag M, Gaudel MC, Gouraud SD. A Machine Learning Approach for Statistical Software Testing. InIJCAI. 2007.pp. 2274-2279.
40. Moghadam MH, Saadatmand M, Borg M, Bohlin M, Lisper B. Machine learning to guide performance testing: An autonomous test framework. In2019 IEEE international conference on software testing, verification and validation workshops (ICSTW) 2019. pp. 164-167.
41. Tuncali CE, Fainekos G, Ito H, Kapinski J. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In2018 IEEE Intelligent Vehicles Symposium (IV) 2018. pp. 1555-1562.
42. Battina DS. Artificial Intelligence in Software Test Automation: A Systematic Literature Review. International Journal of Emerging Technologies and Innovative Research (www. jetir. org| UGC and issn Approved), ISSN. 2019. pp. 2349-5162.
43. Rankin C. The software testing automation framework. IBM Systems Journal. 2002. vol. 41. no. 1. pp. 126-139.
44. Briand LC, Labiche Y, Bawar Z. Using machine learning to refine black-box test specifications and test suites. In2008 The Eighth International Conference on Quality Software 2008. pp. 135-144.
45. IEEE Standard I2 19- 1992. Sofrware Maintenance Standard. published by IEEE Standards Office. P.O. Box 1331. Piscataway. NJ 08855-1331.
46. Levin S, Yehudai A. Towards software analytics: Modeling maintenance activities. arXiv preprint arXiv:1903.04909. 2019 Mar 9.
47. Kukkar A, Mohana R, Kumar Y, Nayyar A, Bilal M, Kwak KS. Duplicate bug report detection and classification system based on deep learning technique. IEEE Access. 2020. vol. 8. pp. 200749-200763.

48. Immaculate SD, Begam MF, Floramary M. Software bug prediction using supervised machine learning algorithms. In2019 International conference on data science and communication (IconDSC) 2019. pp. 1-7.
49. Sidhu BK, Singh K, Sharma N. A machine learning approach to software model refactoring. International Journal of Computers and Applications. 2022. vol. 44. no. 2. pp. 166-177.

Asaad Jameleh. Ph. D student, faculty of software engineering and computer systems, ITMO University, St. Petersburg, Russia. E-mail: jamelehasaad@gmail.com

Avksentieva Elena. Ph.D., Associate professor, faculty of software engineering and computer systems, ITMO University, St. Petersburg, Russia. E-mail: eavksenteva@itmo.ru

Исследование методов машинного обучения для программной инженерии

Ж. Асаад, Е. Ю. Авксентьева

Университет ИТМО, Санкт-Петербург, Россия

Аннотация. Среды машинного обучения (ML) предлагают разнообразие методов и инструментов, которые помогают решать задачи в различных областях, включая программную инженерию (SE). В настоящее время большое количество исследователей интересуют возможности использования различных методов машинного обучения в программной инженерии. В данной статье приводится обзор методов машинного обучения, применяемых на каждом этапе жизненного цикла разработки программного обеспечения (SDLC). Вклад данного обзора значителен. Во-первых, при анализе источников из библиографических и реферативных баз данных было выявлено, что тематика интеграции методов машинного обучения в программную инженерию актуальна. Во-вторых, в статье поставлены вопросы и рассмотрена методология данных исследований. Кроме того, систематизированы методы машинного обучения по их применению на каждом этапе разработки программного обеспечения. Несмотря на огромное количество научных работ по использованию методов машинного обучения в программной инженерии, требуются дальнейшие исследования для достижения всесторонних сравнений и синергии используемых подходов, значимых оценок, основанных на детальных практических реализациях, которые могли бы быть приняты индустрией. Таким образом, будущие усилия следует направить на воспроизводимое исследование, а не на изолированные новые идеи. В противном случае большинство из этих применений останется мало реализованными на практике.

Ключевые слова: машинное обучение, инженерия программного обеспечения, жизненный цикл разработки программного обеспечения.

DOI 10.14357/20718632230408 **EDN** ADARZK

Литература

1. Lwakatare, L.E., Raj, A., Bosch, J., Olsson, H.H. and Crnkovic, I. A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In Agile Processes in Software Engineering and Extreme Programming: 20th International Conference. 2019. pp. 227-243.
2. Shehab M, Abualigah L, Jarrah MI, Alomari OA, Daoud MS. Artificial Intelligence in Software Engineering and inverse. International Journal of Computer Integrated Manufacturing. 2020. vol. 33. no. 10-11. pp. 1129-1144.
3. Durelli, Vinicius HS, et al. "Machine learning applied to software testing: A systematic mapping study." IEEE Transactions on Reliability. 2019. vol.68. no. 3. pp. 1189-1212.
4. Khomh, F., Adams, B., Cheng, J., Fokaefs, M., Antoniol, G., 2018. Software engineering for machine-learning applications: The road ahead. IEEE Softw.2018. vol. 35. no. 5. pp. 81–84.
5. Maneerat N, Muenchaisri P. Bad-smell prediction from software design model using machine learning techniques. In2011 Eighth international joint conference on computer science and software engineering (JCSSE). 2011. pp. 331-336.

6. Talele, P. and Phalnikar, R. Software requirements classification and prioritisation using machine learning. In Machine learning for predictive analysis. 2021. pp. 257-267.
7. Zou, J., Xu, L., Guo, W., Yan, M., Yang, D., and Zhang, X. Which non-functional requirements do developers focuson? An empirical study on stack overflow using topic analysis. In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. 2015. pp. 446–449.
8. Ahmad, A., li, K., Feng, C., and sun, T. "An empirical study on how iOS developers report quality Aspects on stack overflow," international journal of machine learning and computing. 2018. vol. 8. no. 5. pp. 501–506.
9. Treude, C., Barzilay, O., and storey, M. A. How do programmers ask and answer questions on the web? Nier track. In 2011 33rd International Conference on Software Engineering (ICSE). 2011. pp. 804–807.
10. Zou, J., Xu, L., Yang, M., Zhang, X., and Yang, D. Towards comprehending the non-functional requirements through developers' eyes: An exploration of stack overflow using topic analysis. Information and Software Technology.2017. vol. 84. pp.19–32.
11. Ahmad, A., Feng, C., li, K., Asim, S. M., and sun, T. Toward empirically investigating non-Functional requirements of iOS developers on stack overflow. IEEE Access. 2019. vol.7. pp.61145–61169.
12. Yin, H., and Pfahl, D. A preliminary study on the suitability of stack overflow for open innovation in requirements engineering. In Proceedings of the 3rd International Conference on Communication and Information Processing. 2017. pp. 45–49.
13. Bajaj, K., Pattabiraman, K., and Mesbah, A. Mining questions asked by web developers. In Proceedings of the 11th Working Conference on Mining Software Repositories.2014. pp. 112–121.
14. Pinto, G., Castor, F., and Liu, Y. D. Mining questions about software energy consumption. In Proceedings of the 11th Working Conference on Mining Software Repositories. 2014. pp. 22–31.
15. Xiao, M., Yin, G., Wang, T., Yang, C., and chen, M. Requirement acquisition from social q&a sites. In Requirements Engineering in the Big Data Era .2015. pp. 64–74.
16. Rosen, C., and Shihab, E. What are mobile developers asking about? A large-scale study using stack overflow. Empirical Software Engineering.2016. vol. 21. no.3. pp. 1192–1223.
17. Abad, Z. S. H., Shymka, A., pant, S., Currie, A., and Ruhe, G. What are practitioners asking about requirements engineering? An exploratory analysis of social q&a sites. In 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW). 2016. pp. 334–343.
18. Pinto, G. H., and Kamei, F. What do programmers say about refactoring tools? An empirical investigation of stack overflow. In Proceedings of the 2013 ACM workshop on Workshop on refactoring tools. 2013. pp. 33–36.
19. G. Jivani, "A comparative study of stemming algorithms," International Journal of Computer Applications in Technology.2011. vol. 2. pp. 1930–1938.
20. Khan, B. Baharudin, L. H. Lee, and K. Khan, "A review of machine learning algorithms for text-documents classification," Journal of Advances in Information Technology.2010. vol. 1, pp. 4–20.
21. Fernandes E, Oliveira J, Vale G, Paiva T, Figueiredo E. A review-based comparative study of bad smell detection tools. InProceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering. 2016. pp. 1-12.
22. Ferenc R, Beszedes A, Fulop L, Lele J. Design pattern mining enhanced by machine learning. In21st IEEE International Conference on Software Maintenance (ICSM'05) 2005. pp. 295-304.
23. Zanoni M, Fontana FA, Stella F. On applying machine learning techniques for design pattern detection. Journal of Systems and Software. 2015. vol. 103. pp. 102-17.
24. Selvarani R, Mangayarkarasi P. A Dynamic Optimization Technique for Redesigning OO Software for Reusability. ACM SIGSOFT Software Engineering Notes. 2015. vol. 40. no. 2. pp.1-6.
25. Agashe R, Iyer S, Zettlemoyer L. Juice: A large scale distantly supervised dataset for open domain context-based code generation. arXiv preprint arXiv:1910.02216. 2019.
26. Shin EC, Allamanis M, Brockschmidt M, Polozov A. Program synthesis and semantic parsing with learned code idioms. Advances in Neural Information Processing Systems. 2019. vol. 32.
27. Takahashi A, Shiina H, Kobayashi N. Automatic Generation of Program Comments based on Problem Statements for Computational Thinking. In2019 8th International Congress on Advanced Applied Informatics. 2019. pp. 629-634.
28. Shido Y, Kobayashi Y, Yamamoto A, Miyamoto A, Matsumura T. Automatic source code summarization with extended tree-lstm. In2019 International Joint Conference on Neural Networks. 2019. pp. 1-8.
29. Tufano M, Watson C, Bavota G, Penta MD, White M, Poshyvanyk D. An empirical study on learning bug-fixing patches in the wild via neural machine translation. ACM Transactions on Software Engineering and Methodology (TOSEM). 2019. vol. 28. no. 4. pp.1-29.
30. Zhu Z, Xue Z, Yuan Z. Automatic graphics program generation using attention-based hierarchical decoder. InAsian Conference. 2018. pp. 181-196.
31. Kim Y, Kim H. Translating CUDA to opencl for hardware generation using neural machine translation. In2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO) 2019. pp. 285-286.
32. Gozalo-Brizuela R, Garrido-Merchan EC. ChatGPT is not all you need. A State of the Art Review of large Generative AI models. arXiv preprint arXiv:2301.04655. 2023.
33. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374. 2021.
34. Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. Competition-level code generation with alphacode. Science. 2022. Vol. 378. No. 6624. pp. 1092–1097.
35. Bhavya, B., Xiong, J., and Zhai, C. Analogy generation by prompting large language models: A case study of instructgpt. arXiv preprint arXiv:2210.04186 .2022.
36. Dehaerne E, Dey B, Halder S, De Gendt S, Meert W. Code Generation Using Machine Learning: A Systematic Review. IEEE Access. 2022.

37. S. Ahmed, "Overview of software testing standard ISO/IEC/IEEE 29119", Int. J. Comput. Sci. Netw. Secur.2018. vol. 18. no. 2. pp. 112-116.
38. Durelli VH, Durelli RS, Borges SS, Endo AT, Eler MM, Dias DR, Guimaraes MP. Machine learning applied to software testing: A systematic mapping study. IEEE Transactions on Reliability. 2019. vol. 68. no. 3. pp. 1189-212.
39. Baskiotis N, Sebag M, Gaudel MC, Gouraud SD. A Machine Learning Approach for Statistical Software Testing. InIJCAI. 2007.pp. 2274-2279.
40. Moghadam MH, Saadatmand M, Borg M, Bohlin M, Lisper B. Machine learning to guide performance testing: An autonomous test framework. In2019 IEEE international conference on software testing, verification and validation workshops (ICSTW) 2019. pp. 164-167.
41. Tuncali CE, Fainekos G, Ito H, Kapinski J. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In2018 IEEE Intelligent Vehicles Symposium (IV) 2018. pp. 1555-1562.
42. Battina DS. Artificial Intelligence in Software Test Automation: A Systematic Literature Review. International Journal of Emerging Technologies and Innovative Research (www. jetir. org| UGC and issn Approved), ISSN. 2019. pp. 2349-5162.
43. Rankin C. The software testing automation framework. IBM Systems Journal. 2002. vol. 41. no. 1. pp. 126-139.
44. Briand LC, Labiche Y, Bawar Z. Using machine learning to refine black-box test specifications and test suites. In2008 The Eighth International Conference on Quality Software 2008. pp. 135-144.
45. IEEE Standard I2 19- 1992. Sofrware Maintenance Standard. published by IEEE Standards Office. P.O. Box 1331. Piscataway. NJ 08855-1331.
46. Levin S, Yehudai A. Towards software analytics: Modeling maintenance activities. arXiv preprint arXiv:1903.04909. 2019 Mar 9.
47. Kukkar A, Mohana R, Kumar Y, Nayyar A, Bilal M, Kwak KS. Duplicate bug report detection and classification system based on deep learning technique. IEEE Access. 2020. vol. 8. pp. 200749-200763.
48. Immaculate SD, Begam MF, Floramary M. Software bug prediction using supervised machine learning algorithms. In 2019 International conference on data science and communication (IconDSC) 2019. pp. 1-7.
49. Sidhu BK, Singh K, Sharma N. A machine learning approach to software model refactoring. International Journal of Computers and Applications. 2022. vol. 44. no. 2. pp. 166-177.

Асаад Жамилех Университет ИТМО, Санкт-Петербург, Россия. Аспирант. Область научных интересов: модели и методы искусственного интеллекта, программная инженерия, паттерны проектирования GoF. E-mail: jamelehasaad@gmail.com

Авксентьева Елена Юрьевна Университет ИТМО, Санкт-Петербург, Россия. Доцент, кандидат педагогических наук. Область научных интересов: компьютерные сети, надежность вычислительных комплексов и компьютерных сетей; модели и методы искусственного интеллекта. E-mail: avksentievaelena@rambler.ru