Improve Software Development Quality Using ML Practices

Lakshmisri Surya

Sr. Data Scientist & Department of Information Technology.

Abstract

In order to keep up with innovations and incredibly quickly software development, research methods have evolved over the past four decades. Companies today are rapidly investing in machine learning (ML) to remain competitive. The development of applications is fraught with difficulties, and many are now designing AI-based solutions to address these challenges. Additionally, the ability that the system holds to learn inherently adds more uncertainties to the entire system. Considering the rise in popularity of implementing ML into the systems, there are still challenges on how the addition interferes with software development practices. In order to improve software functionality and retain its efficacy, the software is an object that continues to evolve and undergo continual changes. Some difficulties are faced during the production of applications, often with advanced preparation, transparent reporting, and proper process management. Such bugs impair the consistency of applications in one direction or the other, and may lead to failure. Therefore, everyone has to monitor and mitigate these flaws in information engineering in today's competition. Models for software prediction are usually used to map the dynamics of software groups vulnerable to alteration. The paper will also drive the paper to rigorous experimentation from literature reviews to discover all sorts of data collection alternatives. The main views addressed are; machine learning as a technology used in enhancing software development and the parameters quantification which impacts the productivity, functionality, and quality of software.

Key Words

Machine learning, software development, frameworks.

Introduction

Over the past three decades, machine learning (ML) has advanced significantly, from laboratory fascination to realistic application for mainstream industrial use. Inside AI and ML has emerged as the tool of choice for robot control, natural language processing, speech recognition, computer vision, and other applications to create practical software products. Machine learning resources, especially software programs with machine language modules and structures, tools, and databases that include ML functions, can be applied to a system in many ways (Salay et al., 2017). A diverse pattern has surfaced whereby it is relatively easy and inexpensive to build and install ML technologies. Still, it is difficult and costly to sustain them with time due to technological debt. It is because, these types of systems have all the challenges associated with non-ML systems, in addition to an extra range of unique ML problems which needs to be appropriately addressed. There are also unclear assumptions concerning future evidence and ML future effects (Luo, 2016). Current studies have placed more effort into developing platforms for checking and fixing machine learning algorithms and designing structures and ecosystems that enable ML applications to solve ML specific problems.

Despite this progress, tech professionals are also trying to quantify and increase efficiencies using ML's software creation approaches. For the cost-effective production of high-quality and stable ML applications, standardization and

operationalization of software development processes are very important (Prasad et al., 2015). A multi-faceted and dynamic activity is involved in the design of ML based software. Various types of ML implementation processes have also been suggested. These approaches share many similar main steps: monitoring and production data modeling, data curation, and context understanding. Our modern period of digital technology has had a whole new, broad effect on ML. This means that a vast volume of data and a high degree of automation accelerate any need for smart data connection and assessment. Computer solutions have been much more challenging in the last 10 to 15 years. Vast numbers of individual elements are linked to each other through APIs in several modern systems (Das & Behera, 2017). Also, even basic tasks have to be done through a whole variety of different implementations. As a result, this does not simplify the software development process. Therefore, machine learning makes all these things easier through; automatic migration and refactoring, clean code, intelligent assistants in programming, automatic error detection, and troubleshooting.

How Machine Learning Will Enhance Software Development

Machine learning is an AI application technique whereby, from its knowledge, a device learns and enhances its understanding. Also, it does not require a specified type of programming. The critical purpose of machine learning is to make machines understand and learn without human involvement (Dwivedi et al., 2016). The device requires data for the operation in the form of monitoring. In forecasting outcomes, the machine learns to identify trends in the data and continually improves it to get better over time. It's just wrong to keep such a robust technology like Machine Learning apart from the software development model (Luo, 2016). There are various drawbacks attached to the traditional development of software techniques as it is difficult for software engineers to code to a certain degree in a precedent manner. Teaching a program to evaluate the raw data and come up with logic and patterns seems more straightforward.

Machine learning changes the learning process in software development in the following ways: Decision making and prototyping, which required months and even years of preparation, traditionally, to come up with a realistic application. Machine learning, though, helps evaluate the historical success of programs already functioning to identify alternatives (Kim et al., 2016). This benefits the production of apps as it fastens the entire operation to deliver full benefit and lower risks. Under control deployment, from the viewpoint of software design, software deployment is fundamental. Developers update their applications and offer a new version at this stage, which has the likelihood of destroying it (Salay et al., 2017). Machine learning, however, decreases the chances of being destroyed. For any bugs and failures, developers should test their apps. It also handles data security, fixing bugs, and errors whereby ML algorithms promote accelerated identification and warning of violations. When machine learning works well for them, designers do not need to spend their brains discovering glitches and bugs. Research methods using ML quickly detect and correct bugs.

Machine learning also aids data science since it is difficult to develop one of the most potent machine learning algorithms. In order to come up with clustering algorithms, ML development uses machine learning itself. It also lets developers identify bugs in their code by detecting the inevitable glitches (Tantithamthavorn et al., 2016). It also helps in data management since ML can simulate the nature of data and its position. As compared to standard databases, this makes it much quicker and uses less memory. In order to

track and find any problems, app engineers will also be running. Less data means fewer monitoring opportunities and efforts for quicker debugging. ML also brings about efficient codes as it is better to update a million lines with several hundred by machine learning. Not only does this reduce the time involved with coding, but it also renders the system more sustainable (Ghaffarian & Shahriari, 2017). As the software is open to improvements and corrections, teaching also renders the code increasingly flexible, which is why, today, pioneers of ML carry on a new role in training a machine to perform a particular task.

In software development activities, Artificial intelligence demonstrates that they are good at increasing performance. Engineers are implementing these innovations as they realize how useful they are. These tools are influential in nurturing talented programmers and assisting developers in their applications to find and correct bugs (Luo, 2016). Such solutions also offer cloud-based IDEs for designers, smart coding tools, and the ease of implementation control. Much like the experienced ones, young programmers create a significant part of the development culture. AI intends to give these designers a chance to obtain more perspectives on how effective software programs can be designed. AI-powered tools exist that allow designers to focus on software projects (Khanum et al., 2015). These instruments also provide them with the ease of exchanging experiences with young and seasoned developers, helping them learn. In order to advance their careers, young programmers should maximize the use of such resources.

How Machine Learning Ensure Quality on Software Development

Machine learning techniques embody a data framework that takes data from a particular set and makes assumptions about the new data set through having to learn from the data. In order to work mostly on the current data set and predict the trends for the fresh one, machine learning algorithms are developed (Lenarduzzi et al., 2017). ML uses neural network models for quality testing. Neural networks are a collection of structured algorithms that are modified and per the learning method. To develop outcomes and then compare them with set outcomes, the learning process requires input parameters. In producing fast and reliable performance, programs use automation to extract trends from data and interpret the overwhelming amount of data. It is an essential accomplishment in scientific software engineering science to develop effective machine-learning evaluation models, but that is not the only one (Pandey et al., 2017). Equivalence of such models is still required, extensively in software engineering, where it is already a problem to explore and capture information. Many other algorithms are chosen and run-on software data obtained from intermediate programs coming from different ML approaches. Most of these methods deliver very high conceptual output models, and others offer very intelligible and interpretable models.

ML professionals identify market fields that will significantly benefit from ML designs and the relevant information in the comprehension process. ML professionals can consult with clients about what ML designs can handle in terms of standards and what it is not capable to handle (Kim et al., 2016). Most notably, through the execution of preliminary trials in a clear program context, ML professionals frame and scale the implementation activities. The process of developing a machine learning system involves four essential steps. The data curation phase consists of collecting data from various sources, creating test datasets, training, validation, and pre-processing of data. Since knowledge always comes from multiple sources, ML professionals can sew data together and cope with lost or incomplete data by pre-processing data (Cline et al., 2017). Data marking is needed to add labels that are software

oriented to and record to construct an acceptable dataset for supervised learning algorithms.

The feature engineering phase cooperates in all activities that reshape the specified data in a format that is easy to understand, such as feature selection and extraction for ML techniques. ML professionals tune, train, and choose ML models using the preferred attributes during model development (Zanoni et al., 2015). The process modeling phase involves model evaluation, model training, and feature engineering. Trying to adjust configurations and finding possible problems with the existing model or prior stages are used in model tuning. In model evaluation, the performance model on the test dataset is analyzed by clinicians using pre-defined evaluation steps. ML professionals export the design into a pre-defined framework during the development and tracking phase and typically construct a Web or an API application with the system as an interface (Pandey et al., 2017). ML professionals also intend for the model to be retrained with new results. The output of the model is checked continuously for anomalies or unintended consequences, while the input data is observed to decide if they shift in time to be inappropriate for the system.

Software design using Machine Learning

Firstly, for ML structures, the high-level architecture design is relatively fixed. Implementation, data modeling, function engineering, data cleaning, and data compilation usually comprise ML systems' design. The architecture for non-ML software applications, on the other hand, is a more innovative method that incorporates multiple architectural separations of software products and produces definitions of actions (Khanum et al., 2015). The clustered architecture style is commonly favored for ML systems due to a large amount of data. In structural and detailed architecture, a distributed architectural style typically contributes to complexities. Secondly, in modules, ML systems put less focus on low coupling versus non-ML software applications. Even though various components have independent features and functionality in ML systems, they are tightly interrelated (Jindal et al., 2015). For example, data modeling output is based on data processing. Thirdly, for ML frameworks, comprehensive architecture is more versatile than non-ML program schemes. As a result, the thorough modeling of ML structures will be time-consuming and iteratively carried out. Software developers appear to perform plenty of trials in order to design an optimal model.

Maintenance of Software developed Using Machine Learning

According to research, it is evident that minimal effort is required to maintain ML systems compared to traditional software systems. It is because, ML systems run through a predictable decrease in terms of performance with time. Therefore, in aiding robust results, ML systems always need to support automated maintenance (Ghaffarian & Shahriari, 2017). This means that, once a decrease in performance occurs, the system perceives the degradation and offers training to new models of data in an online way by implementing the latest data that has emerged. Furthermore, the management of ML systems configuration cooperates in a vast amount of content as compared to software that does not contain ML design. The main reason behind this is that the ML frameworks cooperate in not only the specific codes but also parameters, hyperparameters, and data. When designing machine learning-based systems, involves iteration and experimentation (Parra et al., 2015). Also, the models of performance would thus vary per the model. Also, to find an optimal combination of all the involved parts that bring about optimal performance, management of configuration is essential to keeping track of hyperparameters, data, architecture, algorithm choice, and associated tradeoffs and models. As a result, the control of structure for machine

learning systems is rendered complex as compared to software designs that do not implement ML designs.

ML algorithms have proven to have the most significant value practically in various domains of software development and maintenance. They are implemented to address poorly managed problems, and limited knowledge exists for humans to develop more effective algorithms. Also, they are used in cases where large databases contain various regularities that are implicit to discovery and in situations where the software can adapt to the conditions that are changing (Malhotra, 2015). Fortunately, the sector of software engineering is rendered a most fertile ground where most of the software developments and tasks that involve maintenance of software developed are formulated as problems that are learned and approached using algorithms of learning. Configuring data management is a complicated art of software development. The general strategy is to model system data into a transparent format that configuration information corresponds to source code form (Tantithamthavorn et al., 2016). For machine learning techniques, this general strategy still applies, but is difficult to enforce, since the device information scale is too big relative to conventional systems.

Quality is indeed crucial in creating a strong and stable ML framework, in addition to information quantity. Garbage in, garbage out represents what is identified from machine learning software as they are fed into the software. Actual information consists of missed values, imbalanced information, outliers, etc. Until creating models, ML professionals must analyze the data (Singh & Chug, 2017). Future research could develop tools for visualization of data that provide data overview, help to locate various irregularities, allow practitioners to concentrate on where cleaning is needed for the data. Good-quality models during design, moreover, will not perpetually guarantee the highest performance of the systems (Cline et al., 2017). A machine learning model deteriorates in precision inside a constantly changing world as soon as the program is put into development. Professionals are supposed to realize that there has never been a finished iteration of a ML algorithm that require to be continuously modified and changed, introducing new data and retraining models. For related development, online reviews and output assessment of ML systems are promising fields.

How the Research on Improvement of Quality Software Development Using ML will help the United States

Machine learning is rendering the process of deploying, developing, and designing software that is cheaper, better, and faster. As a result, this will significantly improve all United States operations that involve quality software designs. Additionally, the research will help the U.S. make software coders, business analysts, testers, and project managers more effective and more productive, thus enabling the nation to develop high-quality software at meager costs. Additionally, machine learning may become one of the most crucial factors that will help the U.S. meet its rising demands for custom made software. Also, deploying and developing custom software is one of the essential elements of the companies' level of innovation within the U.S., resulting in high performing organization designing most of their most critical solutions related to software. Furthermore, the software that implements machine learning in its design process promises to limit all the country's low software quality (Das & Behera, 2017). The new software developed by machine learning processes holds a remarkable impact on the process of software development in the nation, such as decreasing the number of software developers, automatically generating more than half of the required tests for assurance of quality and catching bugs even before testing and reviewing the codes.

Conclusion

Using ML to improve software development quality has been a theoretical concept for a very long time. Recently, organizations and institutions are implementing the idea effectively. Also, most of the ML technologies remain within their infancy regarding what most designers wish will come to be a reality. Furthermore, the benefits of implementing ML in software development are more apparent. The only thing remaining is allocating the resources, which are vital in building the routines and algorithms. It is also a recommendation to companies that are already implementing ML initiatives to extend them in software development and testing to be at peace with the revolution of ML, which is becoming part of the niche. This research is an embryonic step in supporting ML systems development, serving as a basis for the upcoming research in the field, and suggestions that support designers in essential development tasks. ML clinicians should adjust their attitude to overcome confusion, and tolerate the uncertainty of preliminary trials and ML algorithms' randomness. They may benefit from scientific programming, which often includes the creation of exploratory processes. Also, for ML practitioners to gaining reproducibility, version management tools for code, data and variables will play a crucial role. Machine learning experts should also make an appropriate effort to maintain the details for use in software development. In order to promote the effective creation of ML applications, future research should make additional effort to include collaborative and real-time methods for debugging. Online input and output assessment for machine learning-based systems are the most fertile fields for a prospective study to cope with data's quick advancement.

References

[1] Cline, B., Niculescu, R. S., Huffman, D., & Deckel, B. (2017, January). Predictive maintenance applications for machine learning. In *2017 annual reliability and maintainability symposium (RAMS)* (pp. 1-7). IEEE.

[2] Das, K., & Behera, R. N. (2017). A survey on machine learning: concept, algorithms and applications. *International Journal of Innovative Research in Computer and Communication Engineering*, *5*(2), 1301-1309.

[3] Dwivedi, A. K., Tirkey, A., Ray, R. B., & Rath, S. K. (2016, November). Software design pattern recognition using machine learning techniques. In *2016 IEEE Region 10 Conference (TENCON)* (pp. 222-227). IEEE.

[4] Ghaffarian, S. M., & Shahriari, H. R. (2017). Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM Computing Surveys (CSUR)*, *50*(4), 1-36.

[5] Jindal, R., Malhotra, R., & Jain, A. (2015, September). Predicting Software Maintenance effort using neural networks. In *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions)* (pp. 1-6). IEEE.

[6] Khanum, M., Mahboob, T., Imtiaz, W., Ghafoor, H. A., & Sehar, R. (2015). A survey on unsupervised machine learning algorithms for automation, classification and maintenance. *International Journal of Computer Applications*, *119*(13).

[7] Kim, M., Zimmermann, T., DeLine, R., & Begel, A. (2016, May). The emerging role of data scientists on software development teams. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)* (pp. 96-107). IEEE.

[8] Lenarduzzi, V., Stan, A. C., Taibi, D., Tosi, D., & Venters, G. (2017, September). A dynamical quality model to continuously monitor software maintenance. In *The European Conference on Information Systems Management* (pp. 168-178). Academic Conferences International Limited.

[9] Luo, G. (2016). A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, *5*(1), 1-16.

[10] Malhotra, R. (2015). A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, *27*, 504-518.

[11] Pandey, N., Sanyal, D. K., Hudait, A., & Sen, A. (2017). Automated classification of software issue reports using machine learning techniques: an empirical study. *Innovations in Systems and Software Engineering*, *13*(4), 279-297.

[12] Parra, E., Dimou, C., Llorens, J., Moreno, V., & Fraga, A. (2015). A methodology for the classification of quality of requirements using machine learning techniques. *Information and Software Technology*, *67*, 180-195.

[13] Prasad, M. C., Florence, L., & Arya, A. (2015). A study on software metrics based software defect prediction using data mining and machine learning techniques. *International Journal of Database Theory and Application*, *8*(3), 179-190.

[14] Salay, R., Queiroz, R., & Czarnecki, K. (2017). An analysis of ISO 26262: Using machine learning safely in automotive software. *arXiv preprint arXiv:1709.02435*.

[15] Singh, P. D., & Chug, A. (2017, January). Software defect prediction analysis using machine learning algorithms. In *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence* (pp. 775-781). IEEE.

[16] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2016). Comments on "Researcher bias: the use of machine learning in software defect prediction". *IEEE Transactions on Software Engineering*, *42*(11), 1092-1094.

[17] Zanoni, M., Fontana, F. A., & Stella, F. (2015). On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, *103*, 102-117.