Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2023, 10(6):38-41

Research Article ISSN: 2394 - 658X

Automated Testing Strategies for Machine Learning Models in DevOps Pipelines

Naresh Lokiny

Senior Software Developer
Email: lokiny.tech@gmail.com

ABSTRACT
Automated Testing Strategies for Machine Learning Models in DevOps Pipelines address the critical need for reliable and efficient testing methods in the context of deploying machine learning models. This paper explores the challenges and opportunities of integrating automated testing into DevOps pipelines for machine learning applications. It examines the unique requirements of testing machine learning models, such as data quality, model accuracy, and performance metrics, and proposes strategies for implementing automated testing practices to ensure the robustness and reliability of machine learning deployments. By analyzing best practices, tools, and techniques for automated testing in the context of DevOps pipelines, this paper aims to provide insights for organizations looking to optimize their testing processes and enhance the quality of their machine learning applications in production environments.

Keywords: Automated Testing, Machine Learning Models, DevOps Pipelines, Continuous Integration, Data Quality, Model Accuracy, Performance Metrics, Testing Strategies, Validation, Reliability, Robustness, Deployment, Quality Assurance, Software Development, Production Environments

INTRODUCTION

The rapid advancement of machine learning technologies has revolutionized the way organizations develop and deploy intelligent applications. However, ensuring the reliability and performance of machine learning models in production environments poses unique challenges, particularly when integrated into DevOps pipelines for continuous delivery. Automated testing is a critical component of DevOps practices, enabling organizations to validate the functionality, accuracy, and robustness of their software applications. In the context of machine learning models, automated testing plays a crucial role in verifying data quality, model performance, and ensuring that the models behave as expected in real-world scenarios. This paper explores the significance of automated testing strategies for machine learning models within DevOps pipelines, addressing the complexities and considerations involved in testing machine learning application.

WHAT IS AUTOMATED TESTING?

Automated testing is a process in which the tester uses special tools to put the software through its paces and find any bugs that may exist. Also, it has been around since the early 1990s but it has only started to gain popularity in recent years as a result of the rise in the popularity of Agile development and Continuous Integration (CI).
Automated testing is an integral part of the development process. It can help in identifying bugs and defects early on in the development lifecycle, which can save time and money on fixing them later on. Automated tests are also known to be more reliable than manual tests because they are less prone to human errors.
The automated testing process has many advantages over manual testing. These include:
• Reduced developer effort and overall costs
• Improved quality and consistency
• Faster release cycles
• Easy distribution of tests across multiple devices or locations
• Better reporting capabilities and others

It's expected that the global automation testing market size will grow from 20.7 billion USD in 2021 to 49.9 billion USD by 2026 with a compound annual growth rate (CAGR) of 19.2%. This is due to the rapid adoption of mobile and web-based applications. Also, modern technologies such as IoT, AI, and machine learning are quickly expanding which opens a great opportunity to test them.

EXAMPLES OF AUTOMATED TESTING

Usually, when it comes to automated testing, the first thing that comes up is software testing with the purpose of quality assurance of the software. But besides software testing, automated testing can include some other types of testing such as hardware, security, performance, and others.

Hardware testing: validates a product's quality before it leaves the factory, using special hardware and software automated tests. The product being tested is generally called the UUT (Unit Under Test). For instance, automated hardware tests may include stimulation of the UUT mechanically (e.g. vibration, actuation, pressure, temperature changes) or electrically (e.g. supplies power, triggers). After that, the acquired data is analyzed and reports are generated.

Security testing: also known as cyber testing, is a set of automated tests that we run against a software program, a network device, or an entire IT infrastructure to look for vulnerabilities that a hacker could exploit. Testing may include looking for known vulnerable versions of systems (e.g., old versions of a webserver), testing password forms to see if they can be broken by brute force or dictionary attacks, or attempting to overload a system to see if it will reveal information (DDoS, Brute Force).

Performance testing: is a software testing process used for testing the speed, response time, and stability of an application under a particular workload. The primary goal of performance testing is to identify and remove performance bottlenecks in software applications. Some of the types of performance testing are.

Load testing: the application is tested with the expected number of users to see how it works.

Stress testing: It includes application testing under extreme workloads. Finally, the most relevant automated testing type for us is software testing.

THE MOST COMMON EXAMPLES OF SOFTWARE TESTING ARE:

Unit tests: performed for the individual components of the software or its function. Basically, they isolate one specific component or function and test them separately. During this process, it's possible to understand how each unit functions in an application.

Integration tests: come after unit tests. The main purpose of integration tests is to find out any irregularity between the interactions of different components of the software.

Acceptance tests: the main goal of these tests is to prove if the application is doing what is intended to do. They evaluate the system from the perspective of the end-user in a production-like environment.

TESTING CONVENTIONAL SOFTWARE VS TESTING MACHINE LEARNING PROJECTS:

Testing machine learning projects is challenging and there is no one standard way of doing it. Since ML projects are heavily dependent on data and models that cannot be strongly specified as a priori, testing ML projects is a more complex challenge than testing manually coded systems. In contrast to most conventional software tests, ML project tests must include data tests, model tests, as well as tests in production.

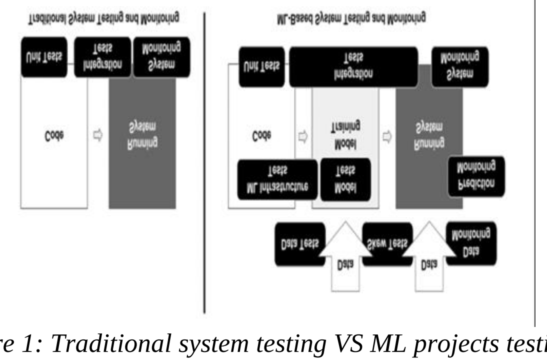

Figure 1: Traditional system testing VS ML projects testing

1. ML projects have a lot more uncertainty than traditional software. In many cases, we don't know if the project is even technically possible, so we have to invest some time to conduct research and give an answer. This uncertainty harms good software practices such as testing because we don't want to spend time on testing ML projects in an early stage of development, which might not receive the green light for further continuation.
2. On the other hand, the more the project stays without testing, the more technical debt it accumulates. As the ML project matures, we must start focusing more and more on increasing testing coverage and paying that technical debt.

3. In contrast to conventional software testing, in ML testing we need to pay special attention to data testing. ML systems differ from traditional software-based systems in that the behavior of ML systems is not specified directly in code but is learned from data. If the input data has some defects then we can’t expect that the ML model will produce optimal results.
4. Lastly, it’s crucial to make sure that the ML system works correctly not only in development and launch but that it continues to work correctly in production as well. In traditional software, tests are only run in the development environment, and it’s assumed that if a piece of code reaches production, it must have been tested and works properly. Over time in ML projects, some external conditions might cause data shifts, or we might change the data source and provider. That’s why we need to continue testing and monitoring the ML system using output logs and dashboards.

CHALLENGES IN MACHINE LEARNING TESTING

As we described in the previous section, testing ML projects is way more complicated than testing conventional software. Besides that, there are a lot of things that we need to pay attention to. Especially since some of them are happening downstream in the ML system pipeline. For instance, anomalies in the prediction might not be because of the model but because of the input data.

SOME OF THE CRITICAL ISSUES IN AN ML PROJECT ARE

Data issues: Missing values, data distribution shift, setup problems, the impossibility of reproducing input data, inefficient data architecture, etc.

Model issues: Low model quality, large model, different package versions, etc.

Deployment issues: unstable environment, broken code, training-serving skew, etc.

Types of automated tests in machine learning There is no one rule for the classification of automated tests in machine learning. Therefore, In the article below, we have roughly divided automated tests into some categories.

Smoke Testing

Smoke testing is the simplest type of testing and should be implemented as soon as a project is started. The main purpose of the smoke test is to make sure that the code runs successfully. It might sound trivial, but this test is beneficial in ML projects. Usually, ML projects include a lot of packages and libraries. These packages provide new updates from time to time. The problem is that sometimes new updates change some functionalities in the package. Even if there is no visible change in the code, there might be changes in the logic that present a more significant issue. Also, maybe we want to use some older releases of more stable and well-tested packages.

Unit Testing

After smoke testing, the next logical testing to implement is unit testing. As it’s mentioned above, unit tests isolate one specific component and test them separately. Basically, the idea is to split the code into blocks or units and test them one by one separately. Using unit tests is easier to find bugs, especially in the earlier development cycle. It’s way convenient to debug the code since we can analyze isolated pieces rather than the whole code. Also, it helps design a better code because if it’s hard to isolate some pieces of the code for unit tests, it might mean that the code is not well structured.

Integration Testing

After unit tests, it’s useful to test how components work together. For that, we use integration testing. Integration testing doesn’t necessarily mean testing the whole ML project altogether but one logical part of the project as a single unit. For instance, feature testing might include several unit tests but all together they are part of one integration test. The primary goal of integration testing is to ensure that modules interact correctly when combined and that system and model standards are met. In contrast to unit tests which can run independently, integration tests run when we execute our pipeline. That is why all unit tests can run successfully but still integration tests can fail.

Regression Testing

With regression testing, we want to make sure that we won’t encounter some bugs which we’ve seen before and already fixed i.e., we want to ensure that new changes in the code won’t reintroduce some older bugs. Because of that, when submitting a bug fix, it’s a good practice to write a test to capture the bug and prevent future regressions. In ML projects, regressions testing can be used when datasets become more complex, models are regularly retrained, and we want to maintain a minimum performance of the model. Every time we encounter a difficult input sample for which our model outputs an incorrect decision, we might add it to a difficult case data set and integrate that test into our pipeline.

Data Testing

As its name suggests, data testing includes all tests in the ML project that are related to any kind of data testing. All previous tests, except smoke testing, might include data testing, in most cases. The purpose of the separate section about data testing is to give ideas and some examples about what can be tested when we are working with data. Since the behavior of most ML projects heavily depends on data, this section is especially important. Below, we’ll cover some tests that can be useful for data validation.

CONCLUSION

In conclusion, automated testing strategies for machine learning models in DevOps pipelines are essential for ensuring the reliability, accuracy, and performance of machine learning applications in production environments. The integration of automated testing practices within DevOps workflows allows organizations to validate data quality, model accuracy, and performance metrics, thereby enhancing the robustness and effectiveness of machine learning deployments. By leveraging automated testing tools and techniques, organizations can streamline the testing process, improve efficiency, and mitigate risks associated with deploying machine learning models. The adoption of automated testing strategies in DevOps pipelines not only facilitates continuous validation and monitoring of machine learning applications but also contributes to overall software quality and reliability. As organizations continue to embrace machine learning technologies in their software development processes, prioritizing automated testing practices within DevOps pipelines will be crucial for optimizing the performance and effectiveness of machine learning models in real-world scenarios.

REFERENCES

[1]. S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software engineering for machine learning: A case study. In Proc. ICSE, 2019.

[2]. S. Banerjee, S. Chakraborty, A. K. Debnath, and R. N. Mahapatra. DevOps for machine learning. IEEE Software, 36(6):100–107, 2019.

[3]. S. Basiri, J. N. Amaral, and L. C. Briand. Machine learning in software testing: A bibliographic survey. ArXiv, abs/2006.10232, 2020.

[4]. N. Botezatu, M. Mircea, and R. Marinescu. Machine learning models in DevOps: A systematic literature review. In Proc. SEKE, 2020.

[5]. Y. Chen, D. Drachsler-Cohen, and T. Menzies. A literature review on the use of machine learning in software testing. In Proc. ASE, 2018.

[6]. L. Cicchetti and F. Qureshi. A survey on machine learning in DevOps. In Proc. ICSE-SEIP, 2020.

[7]. M. D. Ernst, R. Belli, and B. S. Meyer. Machine learning for DevOps: A systematic mapping study. ArXiv, abs/2006.05244, 2020.

[8]. A. G. R. Fernandes and A. M. Simão. Machine learning in continuous integration and continuous deployment pipelines: A mapping study. In Proc. SAC, 2020.

[9]. D. Gavran, D. M. R. Filho, and U. Kulesza. Machine learning for DevOps: A systematic literature review. In Proc. EASE, 2019.

[10]. E. Guzman, S. McIntosh, and A. E. Hassan. A machine learning approach for DevOps knowledge sharing. Emp. Soft. Eng., 24(3):1561–1590, 2019.

[11]. S. Hovsepyan, V. Garousi, and I. Macia. Machine learning in software testing and quality assurance: A systematic mapping study. In Proc. QSIC, 2019.

[12]. J. J. Jiang, B. H. Ren, and S. T. Xie. A survey of machine learning in software testing. J. Syst. Soft., 154:103-117, 2019.

[13]. Y. Li, B. Xie, and L. Zhang. A comprehensive survey of machine learning in DevOps. In Proc. ICSE, 2020.

[14]. M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, and D. Poshyvanyk. How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms. In Proc. ICSME, 2018.

[15]. N. Meng, Y. Zhou, and D. Lo. A survey of machine learning in software testing. ACM Comput. Surv., 51(1):1–35, 2018.

[16]. P. Morrison, T. V. Pham, and A. M. Memon. Machine learning in software testing: State of the art and future directions. In Proc. ASE, 2018.

[17]. S. M. M. Rahman, Y. Z. Li, and C. K. Roy. Machine learning for DevOps: A systematic mapping study. In Proc. ICSE, 2019.

[18]. C. Zhang, H. Zhang, and B. Xie. Machine learning in DevOps: A systematic mapping study. ArXiv, abs/2006.05780, 2020.