Deep Science Publishing
https://doi.org/10.70593/978-93-49910-08-9_9

DeepScience
Open Access Books

Chapter 9: Performance monitoring and optimization of machine learning models in production environments

9.1. Introduction

Significant progress has been made toward deploying machine learning (ML) models in real-world production environments. Such models are able to analyze large datasets, generate predictions, or classify and categorize data in seconds or minutes. They are capable of automating high-stakes, important tasks, such as transcribing digital recordings of court proceedings, automatically approving loans for thousands of customers every minute, and identifying failures in manufacturing machinery through sensor data analysis. Highly successful models may soon be analyzed and tuned by automated systems for optimization and performance monitoring in the same way that highly optimized systems for hyperparameter tuning exist today. Given the investment, promise, and pervasiveness of ML, there is a compelling need to explore effective techniques and systems to operationalize performance monitoring, diagnostic analysis, and performance optimization of such models efficiently, accurately, and effectively (Urs & Zaharia, 2019; Sharma, 2020; Mahmoud, 2021).

Even though many mature ML frameworks exist today to build data processing pipelines, train, and deploy ML models into production settings, there currently exists a lack of automated software systems for model performance monitoring and optimization. Today's ML systems often lack adequate ML model tuning, performance monitoring, and diagnostic analysis capabilities built into them. As a result, ML engineers spend a lot of time and effort building and deploying custom solutions to achieve these ML model lifecycle capabilities. Such custom systems are often hard to implement, fragile, difficult to maintain, and may not necessarily be scalable to the ML workload for a given problem. In addition, many of them may lack adequate security

and privacy protections against unauthorized access, or robust, end-to-end, performance monitoring and analytic capabilities. Many of the significant business costs and risks of deployed models stem from these fundamental limitations. A survey of companies using ML technology found that development and operational challenges came up frequently, and that market need is sufficient for companies to pay for solutions. Performance monitoring of deployed models is a fairly broad and active area of research (Urs & Zaharia, 2019; Sharma, 2020; Mahmoud, 2021).

Fig 9.1: Performance Monitoring and Optimization of Machine Learning

9.1.1. Background and Significance

Machine learning technology has advanced tremendously in the last twenty years because of the simultaneous advancements in computational power, storage capacity, networking complexity, availability of data, and algorithms. These advancements make it possible to deploy complex and large machine learning models to production, generating significant value for society in terms of revenue, time savings, labor wage

reduction, and performance improvements in diverse areas, including finance, healthcare, advertising, ecommerce, and so forth. However, there is an increasing realization that deploying a machine learning model is just the first step in taking advantage of its predictive power. There is a business need to monitor the performance of machine learning models in production, in the context of real data, realistic user behavior, and other external dependencies. By using these monitoring platforms, organizations can catch degradation of model performance, data drift or shadow data issues, model saturation, and so forth. As needed, users can pause model updates and performance, and roll back to the previous version while addressing the reasons for the model or data performance degradation. This has led to the emergence of a number of startups, doing monitoring, auditing, and compliance of machine learning systems.

In parallel, there are further discussions in the academic community about exploring ideas on how to mitigate issues caused by bad inputs during model serving. Misleading data can obviously lead to incorrect outputs. More advanced ideas such as self-supervised learning, multi-task learning, test-time training, and test-time optimization also seek to use additional data and computations at prediction time to further improve the output quality. However, an organization should balance the cost of additional computations versus the value of improvements. Moreover, a model can also be deducted in an instance-dependent manner, depending on the type of input dataset. In this chapter, we create a framework where we can audit models on methods to simulate data drift, shadow data issues, model saturation, as well as insight tire dying.

9.2. Understanding Machine Learning Model Performance

Performance evaluation is essential for the successful deployment of any machine learning model, and the specific performance metric used is usually chosen according to the type of problem - classification, regression or clustering - and the type of machine learning algorithm employed. Numerous metrics are available for evaluating machine learning model performance, each measuring different aspects of model performance. For most supervised machine learning problems, model performance is evaluated from a set of test data that is not visible to the model. Model performance is typically reported as a number between 0 and 1 or as a percentage, where higher values usually indicate better performance. It is important to be careful about considering what metrics to report at least model performance and model training time, as well as making appropriate model performance comparisons using those metrics.

The choice of metric is also influenced by the type of business context in which the model will be used. For example, spam classification should ideally have false-negative and positive error rates close to 0, but the business may be more concerned with the false-positive error rate than the false-negative error rate. Despite being a relatively

simple concept, there are subtleties involved in measuring model performance. First, although most performance metrics are reported as numbers in the range 0-1, numerous metrics are designed to provide very different ranges. For example, accuracy is evaluated as the ratio of correctly classified samples divided by the total number of samples, which falls in the range 0-1.

9.2.1. Key Performance Indicators (KPIs)

When machine learning models are employed in production, it is essential that the performance of the models be assessed to determine whether the solutions are achieving the expected results. An important process in this monitoring is the updating or refining of a system when the performance is not desirable anymore because of data drift, model decay, or alterations in business strategy. Performance assessment thus requires measurement of the discrepancies between the predictions made by the ML production system and the actual results corresponding to the input data generated by customers or other end-users.

Prior to development of a machine vision system, the desired goals of a deployment should be established. This is part of the system definition life cycle. The model which best solves the problem must comply with business objectives and customer-related issues such as latency constraints, uptime requirements, and the cost of mistakes. It is important that the KPIs defined by data scientists reflect the unique production environment and address the priorities set in the definitions. In addition, these KPIs should remain constant throughout the life of the machine learning system. Also, since the purpose of the model is to optimize a larger business goal, it is essential to define the model performance only in context. The KPIs representing the highest business-level goals are often specified in the production environment. Output KPIs may be forensics and robustness related, since model failures are tied to business solutions and indicate many of the underlying issues in most production scenarios.

9.2.2. Common Performance Metrics

Certain model performance metrics are ubiquitous and can be calculated and compared independently of the task in a uniform manner. In this section, I provide a few examples of such commonly computed performance metrics. Note that the metrics discussed in this section are evaluated on the test set defined in Section 6.2.

Cross-Entropy Loss (Log Loss). The negative log likelihood of the ground truth from different label distributions is commonly used for evaluating classification models. Typically, for each sample in the test set, the cross-entropy loss is calculated as the

negative log of the model predicted probability for the true label. Then, the average loss across samples is calculated:

Loss(y, ŷ) = -\frac{1}{N} \sum\limits_{i}^{N}y_i\log{\hat{y}_i},

where y = [y_1, \dots, y_N]^T is the true label, ŷ = [ŷ_1, \dots, ŷ_N]^T are the model predicted probabilities computed by a softmax function after the linear classification layer, N is the number of samples, y_i is a one-hot encoded vector for sample i, and ŷ_i is the model predicted probability for label j of sample i.

Precision/Recall/F1-score. Precision/recall is a commonly used method to evaluate binary or multilabel classification tasks. Given a threshold, precision is defined as the fraction of samples correctly classified as positive out of all samples labeled as positive by the classifier:

Precision = \frac{TP}{TP + FP},

where TP is the number of true positives, and FP is the number of false positives. Recall is defined as the fraction of correctly classified positive samples out of all positive samples:

9.3. Challenges in Production Environments

At the heart of all work processes involving AI and machine learning are models that require maintenance and upkeep, just like the software and hardware resources of an organization. Maintaining the validity of machine learning models, deployed as part of a broader AI system and creating value by virtue of generating predictions, recommendations, or generating decision actions, is used less than conventional software for the decision making of an organization. In part, this is due to the reliance on complex machine learning systems, which question the need for transparency and explicability after deployment. The traditional presumption that validation is only required earlier in the life cycle is also relevant. As decision making increasingly relies on model predictions after their original development and training, it begs the question of monitoring model predictions, and of what actions should be taken if there is a deterioration in the quality of predictions.

Machine learning models deployed in a production environment must communicate with external environments, mostly the streams of data that are used to train the models. A large number of questions and requirements emerge, not least of which is whether the data stream remains stationary from the time the model is created and validated. Many of the requirements that need to be dealt with are similar to those faced by AI and machine learning practitioners during the training and creation of valid, generalizable

models. The production phase also differs from the training lifecycle in what happens if a performed model decisively deteriorates with respect to pre-specified metrics after being deployed. In conventional software, errors found after deployment are less frequent, more costly than those found as part of the conventional lifecycle, and a determinant driver of many software development processes and phases. In most cases involving machine learning models, prediction errors are present, for example, an online recommender may have an accuracy of only 20%, but these error rates are declining, thus providing a reliable, time-sensitive suggestion regarding action.

9.3.1. Data Drift and Concept Drift

The input data fed into a model, or the data distribution, is often different to the one that existed when the model was trained. This difference is termed data drift or covariate shift. Most supervised models are sensitive to the distribution of input features because the features are either used directly or in some function to model the output. For example, all classifiers use some function of the features to compute the probability of different classes. Sensitivity to data drift implies that small shifts in the distribution will often lead to shifts in model performance, and large shifts will almost always lead to significant performance drops.

Many times we find that a classifier is predicting the correct class, but this is simply an accident of the model having learned spurious correlations. In such cases, drift is possible without any significant change in model performance. Performance drop is a necessary but not sufficient condition for data drift detection. The model making the wrong predictions may correctly assign the data to the prior class distribution, but have a zero likelihood when tested on prior class conditional distributions. This leads to a word of caution when using metrics such as KL divergence.

When such data distributions are fed into the model dashboard, instead of the actual prediction probabilities, monitoring dashboards that visualize data statistics would raise alarms for data drift alerts, visualizations where audio can also be added. Alarms for data drift detection models can also be set up for data collection pipelines. Such data shifts are detected by comparing some features of interest that were present during model development and those that are present on actual production data.

9.3.2. Resource Constraints

When a machine learning model works as expected in a testing environment where all of its resources are unlimited (mostly time, memory, and processing power) it is often the case that this does not hold true in a production environment. Resource constraints

in a production environment often limit the performance of machine learning models to such an extent that it may not be cost effective or even possible to use the model at all. Sometimes, inferences take excessive time or resources to produce, or memory or processing capacity to run the model locally is insufficient. Even if the response time is acceptable, the computational and financial costs of running the model in production may be excessive. Additionally, resource constraints do not – with rare exceptions – only affect big, complicated models. Even a simple linear regression model stored on a very large device, such as a smartphone, may lead to an unacceptable combination of latency and resource consumption if a feature in the feature set is produced by a CPU-intensive process which is run for every new input that has to be processed at that moment. It is therefore essential to recognize and understand any possible resource constraints before deploying a machine learning model. If, in spite of considerable risks and trade-offs, resources are constrained, then a regular retraining or fine-tuning schedules can help maintain a functioning model for a longer time, preferably throughout the model lifecycle.

9.4. Monitoring Techniques

Several monitoring techniques have been introduced to date, with each approach designed to address certain monitorability constraints. These monitoring techniques can be grouped into two categories: real-time monitoring and batch strategy monitoring. The first approach is suitable for real-time production pipelines where immediate corrective action is warranted. The second batch monitoring approach is designed for production pipelines where batch scoring techniques are more appropriate. A summary of the two techniques is presented.

Real-time Monitoring Tools

There are three classes of real-time monitoring tools developed to address certain unique constraints for models deployed in production environments. These classes include middleware layer monitoring, logging, and custom functions. Middleware monitoring tools are monitoring systems designed to sit in the middle of the input and output path in the input transformation, model application, and output processing locations in production pipelines. These tools require minimal engineering grunt work, by supporting commonly used ML pipelines without modification.

In our experience, most production ML pipelines are, however, highly customized scripts built from popular ML Python libraries. These highly customized scripts do not have the necessary plug-and-play functionality needed to support middleware monitoring. As such, they lack the common programmable APIs needed to plug into the polish and completion functions of the middleware. The primary functions of the

middleware include data validation, model validation, model monitoring and predictive drift checks, predictive monitoring, and automatic model switching, model retraining, and model scheduling. For machine learning, pipelines with unique middleware scraping functions designed to recompute model training errors or stats, model inference errors or stats, or class predictions are also needed.

9.4.1. Real-time Monitoring Tools

The concept of monitoring is central in Information Technology. A proper monitoring system can inform a company about relevant aspects of its services and warn it about issues that might lead to downtime and end-user satisfaction. In the Software Development Lifecycle, monitoring is one of the pillars of the Operation and Maintenance phase. The Operations team is responsible for ensuring that all services are running properly by using tools that catch misbehaviors in real-time.

Fig 9.2: Real-time Monitoring Tools

At this point, you may be asking yourself why machine learning models are different than any other software services. The answer is that data-driven services present a unique set of complexities that need proper monitoring tools to ensure proper performance. In

particular, the quality of the data used during prediction needs a specialized set of monitoring tools to ensure that the model is applied to the appropriate data range. If the model is used outside its data range, it might start producing invalid outputs. These outputs might have serious, real-world consequences. Think, for instance, about a credit approval model that is applied to a previously unseen demographic but still has the final verdict dictated by its predictions. If that model starts to inaccurately approve new clients purely based on the model prediction, the financial institution might suffer heavy losses.

In order to perform such monitoring functions, a monitoring tool can be created. A monitoring tool can sample in real-time the predictions done by the model, fetching the input values as well as the output values. The monitoring tool should be capable of detecting prediction momentum as well as shifts and drifts in the input data distribution.

9.4.2. Batch Monitoring Strategies

We have already discussed how several tools enable seamless and automated periodic monitoring of ML models deployed in production. For certain model types or specific performance metrics that matter, this might be sufficient for a large variety of use cases, but these periodic tool checks might not catch the drift or drop in performance at the right time preventing potential hazards to the organization and client, especially if the notifications or alerts are not triggered at the correct threshold levels. Moreover, some sensitive industries may be required to keep track of the model's performance data more scrupulously along the entire lifespan of the model in production. This kind of requirement is frequently rather common in niche industries such as financial services, pharmaceutical industry, and data-driven government strategies, to mention a few. These industries have a major need for batch monitoring modeling strategies that allow for a more persuasive analysis of all factors affecting the model output, potentially outlining the areas needing more focus or scrutiny. Keeping this in mind, we cover some batch monitoring modeling strategies that allow for a more explicit exploration and analysis of ML models in production. Batch approaches can be simple post-hoc analysis tools using the test set or holdout validation set to detect data drift or anomalous prediction values. For example, a classifier monitoring strategy may track the accuracy of the model over time to verify that the value remains stable. If not, the testing data can be used to assess the model further. It is also very typical to plot the predicted values against the target value for classification tasks just like with regression tasks checking for outliers. The same goes for logits in classifiers as they expose the classifier's confidence. These quantitative and qualitative approaches may also bring to light prediction failures, possibly allowing for data-driven mitigation strategies.

9.5. Performance Evaluation Frameworks

While performance statistics from the testing phase provide some evidence about how a model may behave in production, they do not provide strong evidence about how a model will behave in production. The statistics are based on a sample of data from a particular distribution. Model decisions on test data are also not executed. Demonstrable performance using a model in production should involve execution and yet, opening a new model to large volumes of new data can be high-risk. Performance evaluation frameworks allow for safety in model decision making while providing feedback about the model's behavior. The insights from these frameworks can help in making decisions for deployment with accuracy and efficacy.

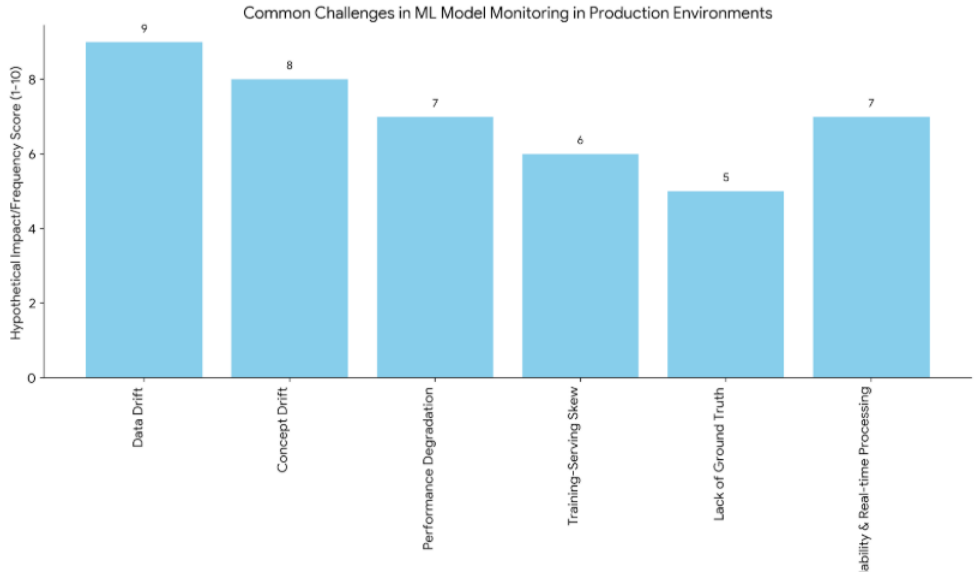

Fig : Optimization of Machine Learning Models in Production Environments

A/B Testing

The primary purpose of A/B testing is to test competing hypotheses to assess which model performs better with real user interactions. It is best for comparing two candidate models that differ only in a small number of features. In an A/B test, the users are randomly assigned to a control group and a treatment group. The control group is exposed to the current model being used in production while the treatment group is exposed to the new model. The model performance for the two groups is recorded and compared using, chiefly, two performance measures. One that determines success of the operation or activity and a measure such as mean squared error, root mean log error or any other task-specific performance measure. A/B testing is primarily used for supervised models in production to monitor performance and behavior against metrics of interest.

Shadow Testing

Shadow testing is a method for comparing the performance of a new machine learning pipeline and the production pipeline without any impact on the users. In shadow testing, raw or feature-engineered input features are fed to the candidate model and predictions are generated but not acted upon. In parallel, predictions are made for the same data using the current model in production. The model decisions for each of the two pipelines are compared using task-specific performance measures.

9.5.1. A/B Testing

Performance evaluation of machine learning systems in production environments is hard, since there are lots of confounding factors that affect the ground-truth signal when assessing model performance on the user-facing system. A/B testing, also known as split testing, is a method in which a certain percentage of traffic is routed to an alternative "B" model while the remaining traffic continues being served by the "A" model (the original model). During this period, the output of the experiment is monitored which allows for comparison of the two variants with respect to defined success metrics. The statistical significance of the difference between the two metrics can be computed and a decision can be made.

For any changes being attempted or any new model being deployed, a data-driven evaluation during A/B testing helps create trust in the new system or in the changes being done to it. For example, in the case of the proposed new system being a new model altogether with a change of modeling strategy, it would give a clear sense of trustworthiness of the new algorithm design itself and possibly would help others in the organization working in the same or similar domains to understand the benefits of the new modeling strategy and apply it in their domains as well. A successful testing also allows for comfortable switching of traffic to the new system and a shorter duration of the delay before the new system is put into effect fully or threshold for sufficient confidence is achieved. Further, it makes subsequent switchbacks less painful. However, a caveat is that while running an A/B test, you usually cannot be serving the two systems simultaneously in the rest of the infrastructure in parallel unless they are set up already in a distributed fashion to handle requests from both systems, with possibly no guarantees on model versions.

9.5.2. Shadow Testing

A lighter alternative to A/B testing that evaluates a subset of predictions made by a model in production is known as shadow testing. In our fraud detection use case, imagine that

we build a new model with improved performance metrics based on cross-validation but are unsure whether the new model will actually provide improved performance in production. The shadow testing mechanism can help us determine whether the new model will in fact offer increased performance while avoiding the business costs associated with any errors. Shadow testing is similar to A/B testing. The difference is that A/B testing will return responses based on either Model A or Model B depending on the random number drawn, while in shadow testing, Model A will be used for all responses, and only Model B will be tested for a small subset of responses. These responses will be used to compare the models, typically using performance dimensions such as precision and recall.

Shadowing can be performed in two ways: passive shadowing and active shadowing. In passive shadowing, only the subset of responses for which we run Model B will be accounted for in calculating the performance dimensions by comparing Model A and Model B. In active shadowing, on the other hand, both models will be used for the responses, but all responses will be factored in the performance dimension calculations.

While passive shadowing can give a good estimate of differences in the precision and recall between the models, active shadowing can provide a better estimate of differences in overall error rates between two models, especially when the sizes of the two models used in shadowing are very different.

9.6. Conclusion

This chapter presented a collaborative approach towards models performance monitoring and optimization. The proposed solution is exploratory, where an interactive and flexible interface fosters user engagement and co-creation, necessary for all semi-automatic processes that influence crucial enterprise decisions. A set of plugins was implemented to build a flexible and modular interface. An experimentation and case study were carried out with several stakeholders, in order to create actionable and reliable performance reports, define usability requirements over user interfaces, as well as validate and identify stages and decisions in the model life cycle that would require automated user engagement. The prototype was put to the test with one model used for credit risk analysis, revealing how the solution can provide meaningful alerts on models performance degradation, as well as insights on user-defined data changes and user help in further improving model reliability.

In addition to the framework, which was used to create the designed modules and corresponding models to create the reports' recommendations, we propose four additional measures to reinforce the model audit. Those recommendations are based on the work developed which uncovered the fragility of several use-case based models due

to important bias problems, data leakage and overfitting. If those measures are implemented in addition to our approach, we believe that model monitoring will be even more robust, and not only able to signal model performance issues, but also far more reliable audits of the models performance over time. Modulo a few additional resources and slight modifications in the logic used for the performance degradation checks, the solution can also be adapted and integrated to any monitoring system designed for other high-stakes models on different domains.

9.6.1. Emerging Trends

Recently, several trends related to performance monitoring of machine learning models emerged: the move to the cloud, machine learning as a service, edge computing and the digital twin, the focus on interpretability, connection to the business, and the awareness of the dark side of machine learning deployment.

The shift to the cloud presents many benefits related to deployment time, scaling and cooperation, and technology. Traditionally, researchers implemented machine learning algorithms as libraries in languages. Users had dependency issues when integrating the libraries into frameworks. For production purposes, researchers have made several implementations of algorithms and techniques in many technologies, although using the best implementation is critical for maximum performance. This integration level increases so much with the cloud or machine learning as a service that deployment and execution become trivial, and users should focus only on proper modeling. The cloud offers data servers, storage, processing, analytics tools, machine learning libraries, and communication frameworks. With the cloud, a company can build a complete production loop from model training to feature and model updating and offer to improve predictive performance automatically.

Edge computing replaces data centers deployed far from devices that provide tons of information with devices that are close to the data center or device sensors. With edge computing, an equilibrated portion of data preprocessing will be executed close to the sensors or devices, near the point of origin, with another portion processed on the cloud. There will be a shift from cloud computing to edge computing and then back to the cloud. The sensors will connect to the cloud using short-range communication protocols. The near and far infrastructures will exchange information to work as a digital twin, a digital replica of physical assets. With the digital twin and edge computing, predictive monitoring will be usable on lower investment budgets.

References

Mahmoud, Q. H. (2021). Cloud-Native Applications in Java: Build robust, scalable, and reactive software with modern cloud-native architectures. Packt Publishing.

Sharma, A. (2020). Mastering MLOps: Building and Managing ML Models Using Google Cloud, Azure, and AWS. Packt Publishing.

Urs, H., & Zaharia, M. (2019). Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing. O'Reilly Media.

White, T. (2015). Hadoop: The Definitive Guide (4th ed.). O'Reilly Media.

Wittig, A., & Wittig, M. (2023). Amazon Web Services in Action (3rd ed.). Manning Publications.