Article title: Machine Learning in Production: From Experimented ML Model to System
Authors: Pritom Bhowmik[1]
Affiliations: Computer Science & Engineering, IEM, Kolkata, India[1]
Orcid ids: 0000-0003-4585-6024[1]
Contact e-mail: pritom01bh@gmail.com

DOI: 10.14293/S2199-1006.1.SOR-.PPKHXND.v1
Preprint first posted online: 08 June 2022
Keywords: Machine Learning Pipeline, Neural Architecture Search, Principal Component Analysis, Model optimization, Dimensionality Reduction, Directed Acyclic Graph, Data Orchestrators, Model Decay

Machine Learning in Production: From Experimented ML Model to System

Pritom Bhowmik
B. Tech. (Computer Science & Engineering)
Institute of Engineering & Management, Kolkata, India
Email: pritom01bh@gmail.com
ORCID: 0000-0003-4585-6024

Abstract

Production ML pipeline refers to a complete end-to-end workflow of a machine learning product ready for deployment. In recent years, companies have vastly invested in Machine Learning research; developers are developing new tools and technologies to make ML more flexible. Now, we can experience AI in most devices around us, from home appliances to cars. When we want to develop an AI-powered product, it is vital to understand the crucial workflows of the ML. Academic research to develop an ML model and a production ML pipeline are entirely different scenarios. From business problems, data collection to deploying the model is an acutely iterative process. Most of the time, Data scientists and Machine Learning Engineers need to deal with issues like data shift, concept shift, model decay, etc. Sometimes, there are need to change the complete ML architecture or how the features are engineered in the dataset. It will become tedious if someone is working in such an environment and lacks an understanding of the entire workflow of the ML pipeline. Though every ML project is different, a data scientist/ ML engineer/ data engineer must understand the end-to-end workflow of the ML pipeline for the product they are developing. The challenge starts with a business problem. We may face different domain problem statements that need to be solved with Machine Learning. How the data will be collected is also a big concern. Data pre-processing, data validation, data monitoring, feature engineering, Model Selection, hyperparameter tuning, model optimization, model performance analysis, performance evaluation, detecting bias, model deployment, post-deployment analysis & monitoring are the crucial processes to make your model production-ready. The main contribution of this research paper is to present a complete picture of the end-to-end workflows of a production-ready ML pipeline. The process can apply to any production ML project though some workflow or steps may differ due to the domain or use-case's demand. A proper ML pipeline architecture should be easily maintainable, scalable, and reusable. Because as the machine learning project grows, it becomes more and more complex. So, performing regular updates and scaling will become easy for data scientists & ML engineers if the pipeline is well designed and automated.

Index Terms: Machine Learning Pipeline, Neural Architecture Search, Principal Component Analysis, Model optimization, Dimensionality Reduction, Directed Acyclic Graph, Data Orchestrators, Model Decay

Introduction

The production ML model is about developing a service or a product using Machine Learning development and modern software development. A machine learning pipeline project starts with data ingestion and ends with an output from the trained model. Most of the academic ML research used prepared data, collected, cleaned & labeled with the supervision of experts. However, data is the most laborious and most time-consuming part of an ML project in real-world machine learning use cases.

Moreover, the model's performance in production depends mostly on how data is collected and labeled, how the dataset is feature engineered and validated, model optimizations, etc. In an ML research environment, static datasets are used to develop models, where datasets remain the same during the whole duration of the development.

Nevertheless, in a real-world ML project, data is collected continuously from several systems and fed to the ML model, where data is dynamic and regularly shifting. That is why it takes frequent assessments and iterative training to perform well in the production environment. It starts with data collection, accurate labeling (supervised learning), minimizing dimensionality, feature engineering, handling rare conditions to maintain fairness, deployment for serving, and post-deployment model maintenance. Each of the stages of this life cycle is critical for the success of the ML product. Working in a production ML project brings many new challenges. Model development is only around 5% of the entire development process, and that is why production machine learning is very different from academic research settings. So, building the model is hard but moving it into a production setting is harder. Maintaining the data quality and model's performance are the new sets of challenges data scientists and engineers face. Selecting the right tools is also a significant challenge. Google, IBM, Databrick, and AWS offer state-of-the-art toolsets to simplify those tasks. However, there are still enormous challenges to implementing the right tools and technology.

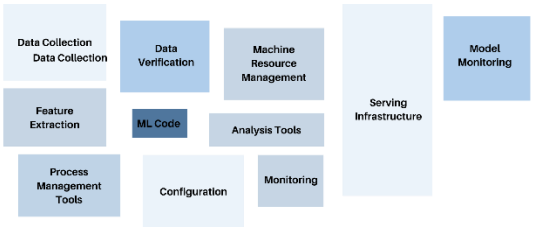

Figure 1 ML Workflow

In this paper, the research focuses on the complete workflows of a production machine learning pipeline to develop a data science product. A real-world ML project starts with business understanding, where the data science team works with domain experts to understand the problem statement. This process can take months sometimes but asking the right question is the key to the project's success. After that, the team starts working with data. In many cases, 60-70% time of the whole project needs to be invested in collecting, cleaning, and feature engineering the dataset. A dynamic dataset needs to build a data pipeline for the data lifecycle to ingest data correctly to the model. Then the ML model needs to be selected using Neural Architecture Search (NAS) process. The next step is optimizing the model's performance to get the ML model's best accuracy. These processes are highly iterative and take proper development planning to implement. It is essential to analyze the model and data for data shift, concept shift, and fairness during these periods. Because data continuously collected from different sources can be changed at any time, and a feature can be added or removed. Then the model needs to be compressed or pruned to deploy for a particular platform (mobile/IoT/Web application). Post-deployment analysis needs to be performed continuously during the model's lifetime. Moreover, the whole process must work automatically like a pipeline to make it more scalable and maintainable.

Business Understanding & Identify Data Sources

Business understanding is at the top of any data science lifecycle. The data science team needs to work with customers and stakeholders to understand the business problem initially. Furthermore, formulate questions that define the business goal to be solved. Then the team works on identifying relevant data that help answer the questions. The type of questions defines what machine learning techniques need to be implemented. A Microsoft article defines these questions like this:

- How many/ how much? – Regression Model
- Which category? – Classification Model

- Which cluster/group? – Clustering Model
- Which option should be taken? – Recommendation Model
- Is this anomaly? – Anomaly Detection Model

Formulating the business problem statement into questions is the first crucial task of a data science project, and domain research may sometimes be needed. After answering these questions, the next step is identifying the data sources. Data need to measure the target and features relevant to the questions. Dataset can be collected from different sources depending on the project domain. Data can be collected from an online survey, social media, government records, mechanical systems, websites, e-commerce, etc. Then it needs to be ingested into the analytical visualization tools to understand if the data quality is adequate to answer the questions.

Production ML Pipelines & ML Orchestration

ML pipelines are software architecture to automate, monitor, and maintain ML workflow, including data validation, data processing, model training, analysis, and deployment. Production ML pipelines combine ML & software development and a formalized, scalable, maintainable process with running sequences of tasks. A production ML majorly works on dynamic data where new data are available frequently, sometimes near real-time. A data scientist team needs to automate these time-consuming steps by implementing a machine learning pipeline so that they can focus on developing more accurate models. ML pipeline workflows are different from each other, depending on the architecture but in general, they are almost always directed acyclic graphs (DAGs). According to Apache Airflow, "A DAG is the core concept of Airflow, collecting Tasks together, organized with dependencies and relationships to say how they should run." It is only concerned with how to execute the process. In a pipeline, there are a lot of variables that need to account for, which makes data pipeline idempotency a challenging task. Data quality problems, interruptions in connectivity, late-arriving data, etc., can cause errors. So, data orchestrators produce DAGs (that simply rerun the DAG on the error) which will load the data despite any of these errors occurring.

ML Orchestration helps implement and manage ML pipelines from beginning to end. Production ML pipelines comprise components (each component with the defined task) and work together to enrich the product for the end-users. Data featurization, model training, evaluation & deployment, and monitoring are the crucial ordered steps, and each step depends on its predecessor. To perform well in production, each step/component must coordinate reliably and regularly. Furthermore, pipeline orchestration performs the responsibility for scheduling the various part in an ML pipeline with the help of automation. Airflow, Kubeflow, and Argo are the leading solution for pipeline orchestration frameworks.

Validate & Monitor Production ML Data

Understanding the esoteric statistics of data, evaluating training datasets, and detecting anomalies to fix are crucial steps for data validation. Data Drift and Skew are the common issues ML engineers face regularly. Model's performance decay over time occurs due to training and serving data issues. Moreover, data drift and concept drift are the two reasons for that problem. Data can change over time due to many unexpected events. For example, during the 2020

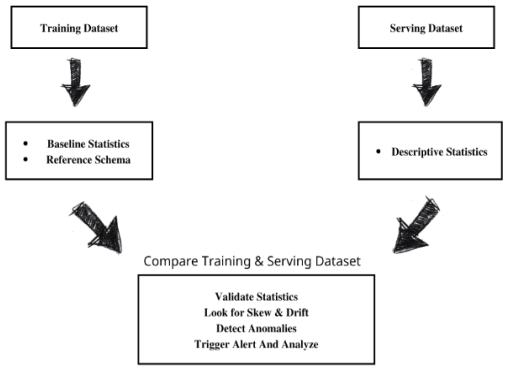

Figure 2 Validate Training & Serving Dataset

COVID-19 lockdown, many people started using their credit cards for online shopping so frequently that the ML model got drifted data. Moreover, that resulted in wrong credit card fraud warning to the customers. Concept drift happens more naturally with time. The data mapping can change with time in real-world dynamic data, and the model also needs to change. Schema skew can occur in data when training and serving data do not confirm the same schema (we can get a string where we are expecting an integer). So, Skew detection needs continuous evaluation of the data coming to the server from different sources once we train our ML model. That makes continuous validation and monitoring extremely essential. To avoid distribution skew, the numerical & categorical features of the evaluation data should be nearly the same range as the training data. Below is the visual representation of both training and evaluation dataset statistics generated by TFDV.

Preprocessing & Feature Engineering at Scale

Applied machine learning often requires strict engineering of the features & pre-processing of the dataset like data cleansing, feature tuning, data transformations, dimensionality reduction, etc. It is crucial to improve the performance of the machine learning model. In a data science project, 80% of time & resources are spent on data preparation. The technique of feature engineering is highly dependent on the particular algorithm. In a real-world production environment, feature engineering on several terabytes of data needs to be feature engineered automated. So, it is ideal to start with a subset of the dataset, performing all the experiments, solving issues, and then scale up to the terabytes where the model will work on the complete dataset. The real challenges we have to deal with are being consistent with the coding approach for training and serving paths, working out on deployment environment during developments, and detecting training-evaluating skews in an early stage. Numerical range and grouping are the categories for feature engineering. Depending on the algorithm, we must understand the scaling, normalizing, standardizing, and grouping used. Scaling converts values to a standard prescribed range, e.g., rescaling an image pixel from the [0, 255] range to the [-1, 1] range so that the convolutional neural network (CNN) performs training and evaluation tasks faster. It is important because at any typical production level computer vision model needs several terabytes of image data, and scaling the dataset will improve the average performance of the model. Furthermore, it helps the model learn each feature's correct weight.

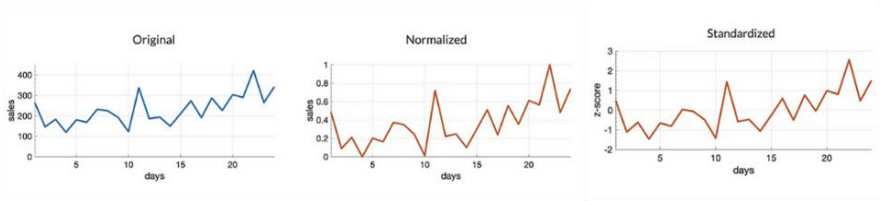

Figure 3 Normalization and Standardization

Standardization is another way of scaling using standard deviation by looking at the data distribution where the data values are centered around the mean. It is good to try standardization and normalization in real-world use cases and compare the results. In some scenarios, we do not want to input raw data into our model; instead, we apply some encoding techniques in a category by grouping that into a Bin/Bucket. For example, we do not want a color name as input if we have a color category in our dataset containing values. The One Hot Encoding technique creates several additional features based on the categorical features' unique values (string/integer). Each unique value in the particular category is added as a feature. This method is effective in neural network classification models. Dimensionality reduction techniques like Principal component analysis (PCA), Kernel PCA, t-SNE, LDA, Backward Elimination, Forward Selection, etc., are used to reduce the number of dimensions of the dataset. In the production machine learning use case, it is crucial to perform a dimensionality reduction technique to eliminate the unnecessary components/features so that the overall performance of the algorithm increase, decreasing the training time, computational resources, and overfitting.

Neural Architecture Search (NAS) & Hyperparameter Tuning

In an experimental machine learning model architecture, we run trial & error on the model by manually tuning the parameters like neuron's number, learning rate, dropout pattern, holdout pattern, hidden layers, etc., to reach a point where we get the most accurate model. And it takes much work for a small experimented model. We need to automate this process in production machine learning to make the model development process efficient. The Keras tuner is one of the solutions which runs the model multiple times, collecting metrics of different parameters each time and optimizing them into the best one. Model selection is also a tedious task if we do it entirely manually. Model selection is the process of understanding which architecture of the model and which topology of the neural network works best in a particular dataset. Neural Architecture Search (NAS) is the process of automating architecture engineering to find the best architecture for the dataset. NAS is categorized into three dimensions: Search space, search strategy, and performance estimation.

- **Search Space**: In the AutoML context, a search space defines a collection of machine learning pipelines from which it searches for a suitable ML solution to the given problem statement. It incorporates prior knowledge about predefined architecture and its properties. Though it simplifies the search, it introduces human bias, which may prevent finding novel architecture solutions.

- **Search Strategy**: Search strategy is how the NAS decides which options in the search space to try. Grid search, Random search, Bayesian optimization, Evolutionary algorithm, and Reinforcement learning are the different search strategies that can explore the space of neural architectures.

- **Performance Estimation Strategy**: The performance estimation strategy is a metric on the search space. It returns a number that corresponds to the performance estimation of the architecture.

Neural Architecture Search (NAS) depends on measuring the accuracy of the different architectures in their trials. In NAS, the search strategy needs to estimate the performance & accuracy of generated architectures to generate better performance & accurate architectures. Lower Fidelity Estimation, Learning Curve Extrapolation, and Weight Inheritance are the different strategies.

Model Optimization: Dimensionality Reduction

To put the model into production and maintenance during its service time requires a cost. And that heavily depends on which computing system the model will run. So, it is essential to work on model resource management in the early phase of the production ML pipeline. Though a dataset contains unnecessary features, a neural network can perform automation feature selection and give expected outcomes. But it will not be an efficient, well-designed model that can be deployed for production. Because there will remain many unwanted features (though the model ignores them), that will take up space and computing resources as the model performs. Sometimes it is plausible that this can cause unwanted noise in the data and degrade the overall model's performance. Each feature contains information that may or may not help a model predict well. As we add more features, we need to increase the number of training examples. We can use manual techniques and also algorithmic

dimensionality reduction techniques for dimensionality reduction. In a predictive model, features must contain information that helps the model predict correctly. We can remove unwanted features directly

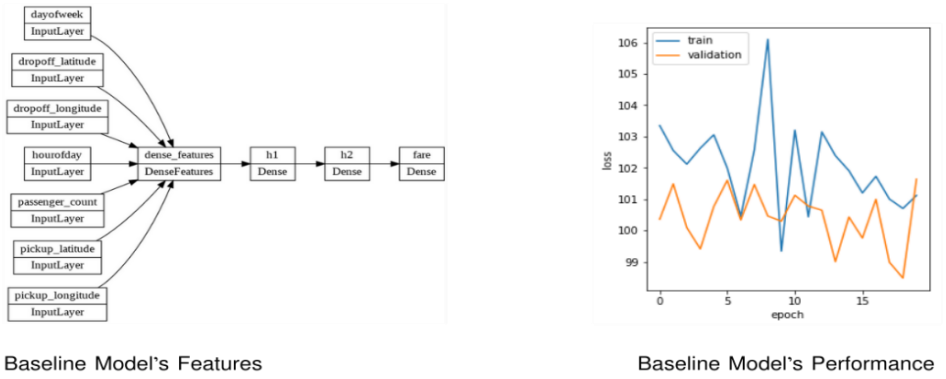

Figure 4 Model's Performance with Baseline Features

from the raw data. Sometimes, new informative features from unwanted data need to be created by aggregating, decomposing, or combining. It is an iterative process that needs continuous data selection and model evaluation. For example, in the NYC taxi fare prediction model, we build a baseline model using dropoff_latitude, dropoff_longitude, passenger_count, pickup_latitude, and pickup_longitude as input features to predict the fare of a particular ride.

So, the baseline model performed poorly. Because the dataset contains a feature about the latitude and longitude of both pickup and dropout points, it cannot determine their distance. Furthermore, taxi fare mainly depends on the traveling distance. We add new features to calculate the distance between each

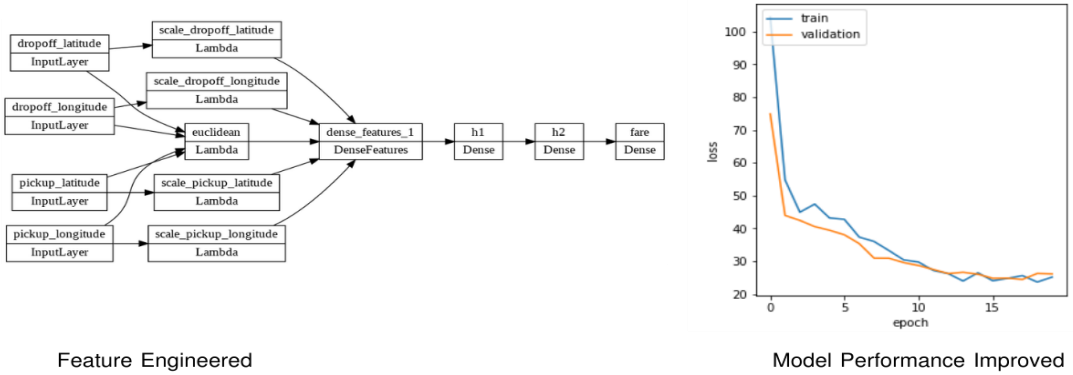

Figure 5 Model's Performance with Feature Engineered

pickup and dropout point. These processes are vastly iterative and need domain understanding to implement.Algorithmic Dimensionality Reduction is an intuitive approach to reducing dimensionality. Moreover, Principal Component Analysis (PCA) is a widely used unsupervised algorithm that creates linear combinations of the original features and learns the principal components of the data. PCA aims to find a lower-dimensional surface to project the data and minimize the squared projection error. The first principal component is the projection direction, which maximizes the variance of the projected data. The second principal component is the orthogonal projection direction to the first principal component; it maximizes the remaining variance of the projected data. Both principal component-1 & principal component-2 comprise a new orthogonal basis for feature space whose axis follows the highest variances of the original data. PCA applies to numerical datasets. For the image dataset, we can use Single value Decomposition (SVD) and Non-Negative Matrix Factorization (NMF) for text data. PCA works on eigen-decomposition, which can be done only by square metrics.

Nevertheless, sometimes, we have sparse matrices, and to decompose this kind of metrics, we have techniques like Single value Decomposition (SVD). To reduce dimensionality, SVD decomposes the original data into constituents, which is used to reduce redundant features from the dataset. Non-Negative Matrix Factorization (NMF) represents data as combinations of commonly occurring visual patterns. It requires the dataset's features to be zero, more significant than zero, or non-negative.

Model Optimization: Quantization & Pruning

In a production ML project, model optimization is the phase where we have to work on model performance optimization and resource management. We expect a real-world case study's highest performance at the minimum cost. The model may be deployed in any embedded, IoT, or mobile applications. It is crucial to identify possible users and the platform. We deploy the machine learning model in the cloud in any typical situation, where a server runs inference and returns the result. However, in some use-case, it is required to perform the ML model locally as part of the device's core functionality. The model needs to be embedded directly into devices that are not connected to the server to serve the model's output. Most of the time, IoT and mobile devices' computing capabilities and storages are very limited. Quantization and Pruning are the techniques used to reduce the compute resources required to serve the ML model. Quantization is a technique that transforms a model into an equivalent model, using lower precision parameters and computation power. Quantization of neural networks is crucial because regular neural networks have millions of connections and take a considerable amount of space and computational resources. So, the quantization model can compute faster, take less space, and run with lower computation power. When quantizing a neural network, the weight parameters and activation nodes computations need to be quantized. This process compresses a small range of floating-point values into a fixed number of information buckets. It reduces the requirement to store a range of the same data type and save bits by saving the range within a smaller range. It is a trade-off process because it may change model accuracy. Quantization can be performed during training or after the model has been trained. TensorFlow-Lite quantizes the baseline model in the example to make it deployable in a mobile device. The model's accuracy decreases from 96% to 93%, and the model size shrinks from 99144 bytes to 24112 bytes. Pruning is another method to reduce the unnecessary nodes and layers in the neural network to reduce the storage and computational cost. The main focus of Pruning is to reduce the number of parameters and operations that have less contribution to the prediction of the neural network. This process makes the training

```
# Save the Keras model
baseline_model.save(FILE_NON_QUANTIZED_H5, include_optimizer=False)

# Save and get the model size
MODEL_SIZE['baseline h5'] = os.path.getsize(FILE_NON_QUANTIZED_H5)

# Print records so far
print_metric(ACCURACY, "test accuracy")
print_metric(MODEL_SIZE, "model size in bytes")
```

```
test accuracy for baseline Keras model: 0.9627000093460083
model size in bytes for baseline h5: 99144
```

Baseline Model's Accuracy & Size

```
# Get the model size
MODEL_SIZE['post training quantized tflite'] = os.path.getsize(FILE_PT_QUANTIZED)

print_metric(MODEL_SIZE, 'model size')
```

```
model size for baseline h5: 99144
model size for non quantized tflite: 84728
model size for post training quantized tflite: 24112
```

```
# Train the model
q_aware_model.fit(train_images, train_labels, epochs=1, shuffle=False)
```

```
1875/1875 [==============================] - 38s 20ms/step - loss: 0.2612 - accuracy: 0.9293
<keras.callbacks.History at 0x7f8dd380e0d0>
```

Quantized Model's Size & Precision have changed

Figure 6 Model Quantization

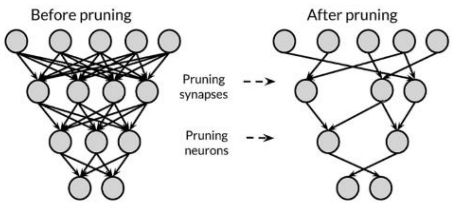

Figure 7 Pruning

faster and uses less storage and computational power. In a limited hardware situation, this is very crucial. This process may cause a decrease in the model accuracy and become a trade-off situation between complexity and performance. The prune_low_magnitude() is a TensorFlow Model Optimization packages method that uses wrappers in a Keras model and prunes it during the training.

Distributed Training & High-Performance Modeling

The model training phase may be simple and fast in research and prototype development. However, training a real-world model for production is a tediously time-consuming task. Furthermore, dynamic datasets are continuously updated and increase with time. The model is also becoming more and more complex. The number of epochs in the model also increases as a result. Because the model development is highly iterative, spending a tremendous amount of time in training is not sufficient. Distributed training allows to speed up the training of a vast and complicated model. Data parallelism and model parallelism are the two types of performing distributed training. In data parallelism, the data is divided into many partitions and given to many workers. Each worker operates on a partition and a copy of the model. Then the model updates are synchronized across the workers. Synchronous training and Asynchronous training are the two ways to perform distributed training using data parallelism. In synchronized training, each node (worker) trains on its current mini-batch of the dataset. This method divides the model into partitions and assigns different accelerators like GPUs and TPUs. However, in this process, only one accelerator is active during the computation, making it inefficient because accelerators are essential for high such high-performance modeling, but they are also expensive. To overcome this issue, GPipe can be used. Gpipe partitions a model across different accelerators and splits a mini-batch of the training set into smaller micro-batches. In this way, accelerators can operate parallel by pipelining the execution process across the mini-batches.

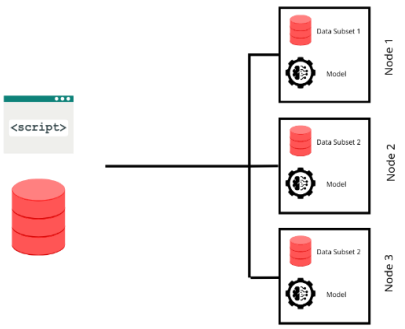

Figure 8 Distributed Training

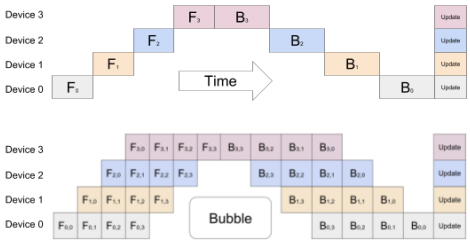

Figure 9 Distributed Training

Model's Performance Analysis & Identifying Bias

Model debugging and model robustness are essential to understanding adversarial attacks on CNN and sensitivity analysis. In production ML, there needs to be a deeper model's performance analysis after the development of the model. Model analysis on individual subsets of the entire dataset is crucial to improve the model's performance after deployment. This process allows further model improvement because the model's performance can decay due to data changes or concept shifts. Black Box evaluation and model introspection are the two ways to analyze the model's performance. In black-box evaluation, the internal structure of the model is not examined but quantifying the model's performance through different metrics and losses. Model introspection methods are used to understand the model's internal structure and improve efficiency and performance by adjusting and iterating the model's architecture. TensorFlow's visualization tool TensorBoard is often used by data scientists and engineers to perform such tasks. A scalable framework like TensorFlow Model

Analysis (TFMA) is often used to deeply analyze the model's performance. It can be integrated with the TFX pipeline so that, before deploying a newly trained or updated version of the model, engineers can perform deep analysis. This can also examine the model's performance against different subsets of the dataset.

Production-Grade Model Serving

Model serving means hosting a machine learning model on-premises or on the cloud and making the model's functionality available via API. We need a production infrastructure and process to put an ML model into production and use it as a service or product. The process is very similar to a production software deployment and DevOps principle; best practices apply to production MLOps. It requires pre-planning from the beginning of the project, or it can lead to serious issues when deploying. So, the data team and engineers should invest proper time and resources in the deployment process from the early stage of Model Development. Model serving means making the trained model available for the end-user by accessing the server or application. It can be deployed on the local storage of a mobile or IoT device as an offline application or an API-based online application/service. So, Batch and online are two types of models serving. A production-grade API has traffic management, access points, pre-processing and post-processing requests, and monitoring model draft functions. There are several ML serving tools like TensorFlow Serving, Amazon SageMaker, Amazon's ML API, IBM Watson, Google Prediction API, etc., for deploying machine learning models in a secure environment with scale.

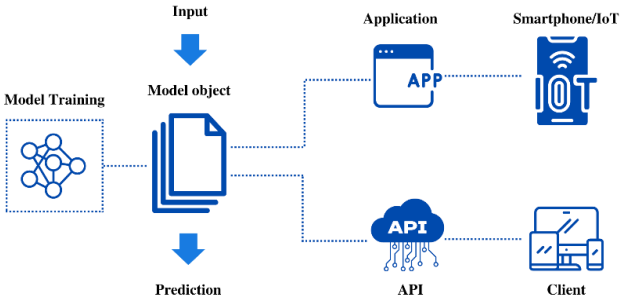

Figure 10 Model Serving

Continuous Performance Evaluation & Monitoring

In an ML pipeline's workflows, model training is performed either offline (batch/static learning), where the model is trained on previously collected data, or online learning, where regularly new data is collected as a data stream. When a model is trained on static data deployed in production, it remains constant until it is retrained. As the model encounter real-world data, it becomes stale, which means model decay. The model trained on steam data can also face such model decay because of data/feature change or concept shifts. Such a phenomenon needs to be monitored regularly after deployment to maintain the model's performance in production. The below graph represents data change during the national lockdown (Covid-19, 2020) because a significant amount of online shopping was increased within a few days. The MIT Technology Review published an article that showed how our sudden online activities change during the pandemic were messed with Artificial Intelligence models. Production ML models can respond to some changes, but if the changes are too

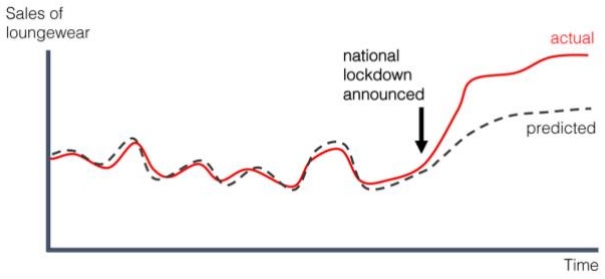

Figure 11 Model Performance Decay

much different from the data the model was trained on, the model will treat the input data as anomalies. The credit card fraud detection model is the perfect example; during the 2020 lockdown, the number of online shopping increased sharply. In the meantime, the use of credit cards also increased. That sudden change of data made the AI model behave strangely, and the system started sending fraud alerts to many credit card users with valid transactions. Sometimes, the change may not be that significant but still cause model decay. Because of that, continuous model performance evaluation and data monitoring after the ML model's deployment in production are crucial.

Monitoring the machine learning model refers to how the data team tracks and understands the model's production performance. Inadequate model monitoring causes incorrect predictions/classification that can affect the business poorly. Data Skews, Model Staleness, and Negative Feedback Loops are the issues an ML model can encounter, and the monitoring system should be able to detect them. Data skew when the training data is not representative of the stream/live data. This can happen if the training data features do not match the features of the live data. Model staleness may occur if we choose the wrong training data set for the model's training. A car manufacturing company's sales prediction model should be trained with regular sales data. If we used the data from the recession when car sales were abnormally low, then the model does not serve well during the healthy economic time. Negative Feedback Loops arise when a model is automatically trained on collected data. But if the data is corrupted or wrongly formatted, the overall model will perform very poorly.

Conclusion

Machine Learning Model deployment and MLOps pipeline implementation can be challenging. But understanding the business problem statement to effectively put an ML in production does not have to be hard if all stages of the development are appropriately planned and executed. Each process of development is highly iterative and time-consuming. But still, in the real world, developing an ML production pipeline that delivers actual business value is extremely challenging. Around 20-25% of the pipeline can deliver business value. Building a successful production ML pipeline needs a diverse set of skills, experience, domain knowledge, and teamwork. Commonly, the first ML pipeline does not meet the business objectives. But as it is an iterative process, proper plans and experimentation will always help meet the expected results. Misalignment between actual business needs and machine learning objectives, testing & validation issues, and non-generalized model training is the top reason machine learning models fail in production. But several tactics can help to make an ML project a success.

- Dependency on Cloud: ML development is a highly iterative process; it needs efficient communication and teamwork. If the team works locally instead of in the cloud, it will slow the whole development process. And it will be harder to implement automation as well. That's why most ML open-source tools are developed for the cloud so that testing, training, validation, and model development become automated and easy to control.
- Leverage a DevOps Approach: A MLOps approach is similar to the DevOps approach. So, ML teams can follow in DevOps approach by implementing the continuous integration (CI) and continuous delivery (CD) model. The ML team can update, change and iterate any part of the development process.
- Investment in monitoring and observability: Healthy data is necessary for a Machine learning model. Without proper clean data and working pipelines, models can perform well. Data changes can occur in the real-world ML scenario and can cause model decay. So, continuous monitoring is the key to a successful production machine learning model.

Machine Learning models do not work like magic, but it is powerful enough to change any industry. So, with proper strategy, processes, planning, and technology, a production machine learning model can deliver competitive advantage and fuel growth across every industry.

References:

1. https://www.jeremyjordan.me/evaluating-a-machine-learning-model/
2. https://www.analyticsvidhya.com/blog/2021/05/machine-learning-model-evaluation/
3. https://madewithml.com/courses/mlops/pipelines/
4. Behrouz Derakhshan, Volker Markl. Continuous Deployment of Machine Learning Pipelines.
5. Doris Xin, Hui Miao, Aditya Parameswaran, Neoklis Polyzotis. Production Machine Learning Pipelines: Empirical Analysis and Optimization Opportunities.
6. Anida Sarajlić, Noël Malod-Dognin, Ömer Nebil Yaveroğlu & Nataša Pržulj. Graphlet-based Characterization of Directed Networks.
7. Xutong Liu, Yu-Zhen Janice Chen, John Chi Shing Lui, Konstantin Avrachenkov. Learning to count: A deep learning framework for graphlet count estimation. 2020
8. https://www.mathworks.com/help/deeplearning/quantization.html?searchHighlight=C&s_tid=doc_srchtitle
9. Software Engineering for Machine Learning: A Case Study (2019) Amershi et al. (Microsoft)
10. A Machine Learning Pipeline to Optimally Utilize Limited Samples in Predictive Modeling
11. TensorFlow-Serving: Flexible, High-Performance ML Serving
12. Wei Jin, 2020. Research on Machine Learning and Its Algorithms and Development
13. J. Schoenberg. Metric spaces and completely monotone functions. The Annals of Mathematics, 39(4):811, 1938
14. https://www.mathworks.com/help/deeplearning/ug/quantization-of-deep-neural-networks.html
15. https://www.mathworks.com/help/deeplearning/ug/quantization-workflow-prerequisites.html
16. https://www.mathworks.com/help/gpucoder/ug/code-generation-for-quantized-deep-learning-networks.html

Funding Acknowledgements

This research did not receive any specific grants from any funding agencies or organizations or company in the public, commercial or not-for-profit sectors.

Pritom Bhowmik

04/06/2022

Statements and Declarations

I wish to draw the attention of the Editor to the following facts which may be considered as potential conflicts of interest and to significant financial contributions to this work. I wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

I confirm that the manuscript has been read and approved by me (only authors) and that there are no other persons who satisfied the criteria for authorship but are not listed.

I also confirm that I have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that I have followed the regulations of our institutions concerning intellectual property.

I confirm that I have provided a current, correct email address which is accessible by the me.

Pritom Bhowmik

04/06/2022

pritom01dev@gmail.com

pritom01bh@gmail.com

Competing Interests Statement

There is no potential competing interest to report.

Pritom Bhowmik

04/06/2022

Author's Details

Pritom Bhowmik
B. Tech. (Computer Science & Engineering)
Institute of Engineering & Management, Kolkata, India
Email: pritom01bh@gmail.com
pritom01dev@gmail.com
ORCID: 0000-0003-4585-6024