Applied Data Analytics - Class Project – MLops: An Overview

Shreyan Chakraborty (MDS202237)
Soham Sengupta (MDS202241)

Mentor: *Dr. M.Chandramouli*

November 22, 2023

Why do we need MLOps?

MLOps, or Machine Learning Operations, is the amalgamation of Machine Learning, DevOps, and Data Engineering. The primary goal of MLOps is to streamline and optimize the process of taking a machine learning model from development to production and maintaining it. This methodology is required because machine learning models are inherently different from traditional software applications. They interact dynamically with the data they consume, requiring continuous monitoring, retraining, and maintenance to ensure they perform effectively when deployed. MLOps provides a structured framework for collaboration across diverse teams, managing the ML lifecycle, and automating the ML pipeline, leading to more robust, scalable, and manageable machine learning systems.

What are the benefits?

The adoption of MLOps offers a multitude of benefits:

- Improved Collaboration: MLOps encourages collaboration across data scientists, ML engineers, and operations teams, fostering a culture of shared responsibility.
- Efficient Scaling: With MLOps, models can be scaled efficiently to handle increased loads and complex workloads without a proportional increase in operational issues.
- Faster Deployment: Automation of the ML pipeline enables faster deployment of models, reducing the time from development to production.
- Better Model Management: Version control for models and data ensures traceability, repeatability, and compliance.
- Continuous Improvement: Continuous integration and delivery allow for iterative improvements to models, leading to better performance and accuracy.

MLOps – Pipeline

The MLOps pipeline constitutes several stages:

- Data Management: Handling, versioning, and pre-processing of datasets.
- Experimentation: Model development, including feature engineering, model training, and hyperparameter tuning.
- Version Control: Tracking different versions of models and datasets.
- Testing: Validating model performance with a focus on robustness and generalization.
- Deployment: Rolling out the model to a production environment.
- Monitoring and Operations: Continuous monitoring of model performance, data drift, and operational health.
- Feedback Loop: Incorporating feedback to refine models and trigger retraining if necessary.

What are the challenges?

While MLOps provides a structured framework for deploying and maintaining ML models, it comes with its own set of challenges:

- Complex Data Management: Handling massive datasets with privacy, security, and governance considerations.
- Model Drift and Reproducibility: Ensuring models remain accurate over time and can be reproduced or rolled back if necessary.
- Infrastructure Complexity: Deploying models across diverse environments requires robust infrastructure that can be complex to manage.
- Skill Gap: Bridging the gap between ML expertise and operations know-how.
- Regulatory and Compliance Issues: Adhering to industry-specific regulations and standards can be difficult.

Why Machine Learning Models Fail In Production?

Machine learning models often fail in production for several reasons:

- Data Drift: The statistical properties of the operational data can change, leading to model degradation.
- Concept Drift: The real-world concept the model is predicting may change over time, leading to inaccurate predictions.
- Training-Serving Skew: Differences in how data is handled in training versus production can cause issues.
- Poor Generalization: Models may overfit to the training data and perform poorly on unseen data.
- Operational Issues: Infrastructure problems such as scalability, resource contention, and deployment errors can lead to model failures.

Understanding these pitfalls is crucial for implementing effective MLOps practices that minimize production risks and ensure that machine learning systems remain valuable and effective in the long term.

Abstract: Hidden Technical Debt in Machine Learning Systems

The paper argues that while machine learning provides a robust toolkit for complex predictions, it's dangerous to assume that the benefits are without costs. The authors introduce the concept of "technical debt" in the context of machine learning, which refers to the ongoing maintenance and unforeseen challenges that arise as ML systems evolve. They discuss various risk factors and challenges associated with ML systems, including hidden feedback loops, undeclared consumers, and data dependencies.

Introduction

- The machine learning community has gained extensive experience with real-world systems.
- "Technical debt" is a metaphor introduced to describe the long-term costs associated with quick software development.
- ML systems can lead to unique forms of technical debt due to their complex nature.

Complex Models Erode Boundaries

- Strong abstraction in traditional software engineering helps maintain code, but ML often blurs these boundaries.
- Issues like "entanglement" can make isolation of improvements challenging in ML.
- Solutions might involve isolating models or leveraging sub-problems.
- Models built for one problem might be misused for another, leading to "correction cascades."
- "Undeclared consumers" are issues where a prediction from an ML model is used elsewhere without proper tracking or documentation.

Data Dependencies Cost More than Code Dependencies

- Data dependencies in ML can be more problematic than code dependencies.
- Data used in ML can be unstable, meaning the signals can change over time or are of varying quality.
- Solutions include creating versioned copies of data or maintaining multiple versions of the same signal.
- Over-reliance on particular features or

Scopes of the Project

After an in-depth analysis of the paper presented to us, we discerned multiple critical domains that warrant thorough exploration. These focal areas not only highlight the current significant themes within the subject but also hint at promising avenues for future

research. The insights gained from this study underscore the importance and potential impact of these sectors, offering a comprehensive roadmap to drive our subsequent investigative endeavors and shape the trajectory of upcoming advancements in this field.

Version Control for ML Models

- Develop an MLOps tool that can handle versioning for both ML models and their associated data sets.
- Implement rollback features, allowing developers to revert to previous model versions and datasets.

Automated Technical Debt Detector

- Create a tool that scans ML code and data pipelines to identify potential sources of technical debt (e.g., underutilized data dependencies).
- Provide actionable feedback and solutions for mitigation.

Isolation Environments for ML Models

- Design a system to isolate and test individual ML models, especially when changes are made, to ensure they don't adversely affect other models or systems.
- This can help prevent "correction cascades."

ML Entanglement Visualizer

- Build a visualization tool that can graphically represent the entanglement in complex ML models.
- Offer suggestions to reduce entanglement and improve model maintainability.

Data Dependency Auditor

- Create a tool that tracks and visualizes data dependencies in ML systems.
- Highlight unstable data sources and suggest alternatives or stabilizing strategies.

Undeclared Consumers Tracker

- Design a monitoring system that tracks where ML model predictions are being used across different systems or applications.
- Alert developers or system administrators about potential undeclared consumers.

Correction Cascades Predictor

- Implement a system that predicts potential correction cascades when changes are made to an ML model.
- Provide insights into the downstream effects of model changes.

MLOps Dashboard for Technical Debt Management

- Create an all-in-one dashboard that provides insights into various aspects of technical debt in an ML system.
- Include features for tracking model versions, data dependencies, undeclared consumers, and more.

Automated ML Feature Optimizer

- Based on the concept of underutilized data dependencies, design a tool that identifies redundant or low-impact features in ML datasets.
- Offer automated suggestions for feature pruning to enhance model efficiency.

Feedback Loop Detector

- Design an MLOps solution that identifies potential hidden feedback loops in ML systems which might lead to biased or skewed predictions over time.
- Suggest mitigation strategies.

Abstract: Scaling Machine Learning as A Service

Machine learning as a service (MLaaS) is imperative to the success of many companies as they need to gain business intelligence from big data. Building a scalable MLaaS for mission-critical and real-time applications is a very challenging problem. This paper presents the scalable MLaaS built for Uber that operates globally. We focus on several scalability challenges, how to scale feature computation for many machine learning use cases, how to build accurate models using global data and account for individual city or region characteristics, and how to enable scalable model deployment and real-time serving for hundreds of thousands of models across multiple data centers. Our technical solutions are the design and implementation of a scalable feature computing engine and feature store, a framework to manage and train a hierarchy of models as a single logical entity, and an automated one-click deployment system and scalable real-time serving service.

1 Introduction

Machine learning (ML) and Artificial Intelligence (AI) have been rapidly transforming many industries such as e-commerce, transportation and healthcare. ML and AI have traditionally been trained in a single machine using tools such as Scikit-learn or R with trained models being deployed manually. However, in the big data and on-demand real-time service era, this severely limits the scale of application of ML. Recognizing a need, many companies such as Facebook and cloud providers such as Microsoft, Amazon, and Google have been building machine learning platforms to automate and scale machine learning. ML is critical to Uber's business in several ways. It finds wide applicability in predicting supply and demand, recommending restaurants for UberEATS, mapping, and self-driving cars, for example. To unleash the true power of machine learning, we have been building a company-wide ML platform within Uber. The goal of the platform is to make it extremely easy and fast for any engineer or data scientist to develop and deploy ML solutions in production. This is particularly challenging for several reasons. First, Uber applications interact with many people and businesses, such as restaurants, in real-time, which generate a large amount of data. Training with big data needs to be scalable to enable fast iteration of model development and frequent model deployment: predictions need to happen in real-time.

2 System Architecture:

The architecture section details the ML workflow at Uber, which includes data acquisition, feature engineering, model training, deployment, and monitoring. The UberEATS Estimated Time to Delivery (ETD) feature is used as a case study to illustrate the practical application of this workflow.

3 Data Preparation:

The paper discusses the continuous collection and storage of data, highlighting the use of SQL as a common querying interface and the integration of metadata to facilitate the sharing of data across different use cases.

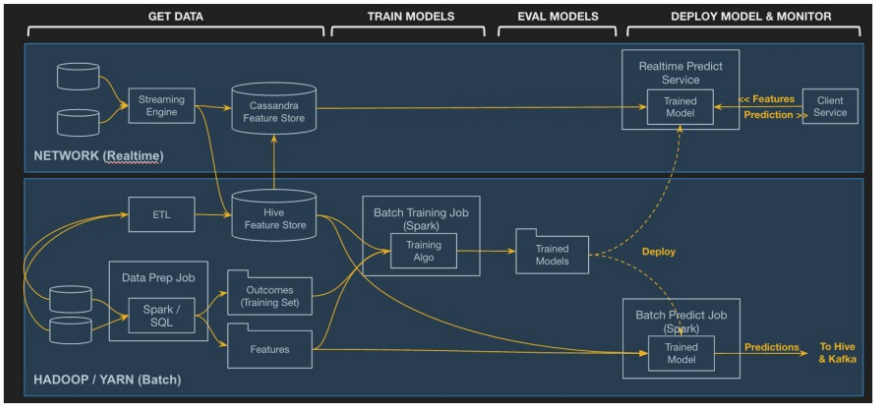

Figure 1: Data Preparation pipeline

4 Feature Store:

To streamline the building of machine learning pipelines and enable real-time predictions, Uber developed a shared feature computation engine and feature store for batch and real-time machine learning pipelines.

5 Model Training:

The platform supports various ML algorithms and aims to be highly scalable and reliable. For deep learning algorithms, support is provided for Caffe and TensorFlow.

6 Model Deployment and Management:

In the paper, the authors explore the multifaceted process of deploying and managing machine learning models at Uber. Deploying a model is the process by which a trained machine learning model is made available to make predictions or decisions in a production environment. This is a critical step in the machine learning pipeline as it translates the developed models into practical, value-adding tools for the end-users.

Uber's approach to model deployment focuses on automation and scalability to handle the vast number of models required for their diverse applications, from UberEATS to self-driving cars. The main challenges in this domain involve ensuring that models are updated with new data, handling the high throughput of predictions requested by the applications, and maintaining model performance at scale.

Management of machine learning models includes monitoring the performance of models in real-time, diagnosing and troubleshooting issues, and continuously updating models with new data to ensure that they remain accurate over time. Uber's strategy likely involves a robust infrastructure that supports continuous integration and deployment (CI/CD) pipelines, allowing for the smooth transition of models from development to production.

Moreover, at scale, model management must address the following challenges:

– Versioning and Model Tracking: Keeping track of numerous model versions, their performance metrics, and the specific datasets they were trained on.

- Performance Monitoring: Implementing systems to monitor model accuracy, detect drift in data distributions, and trigger retraining workflows as necessary.
- A/B Testing: Comparing different models in a live environment to determine which performs best, without disrupting the user experience.
- Rollback Mechanisms: Ensuring the ability to quickly revert to previous model versions if a new deployment results in unexpected issues.
- Automated Retraining and Deployment: Creating systems to automatically retrain models on new data and deploy them without manual intervention, while ensuring that they meet predefined performance criteria.
- Resource Management: Efficiently allocating computational resources to different models, especially in a microservices architecture where different components may need to scale independently based on demand.
- Uber's deployment infrastructure must be designed to be resilient and fault-tolerant, ensuring that the machine learning models can handle peak loads and diverse scenarios without downtime or degradation of service.
-

By addressing these challenges, Uber aims to create a dynamic and responsive ML ecosystem that can adapt to changing conditions and maintain the highest standards of service for their users.

7 Real-Time Prediction and Live Monitoring:

Real-time predictions are essential for services like UberEATS ETD. The system must offer low-latency predictions and be capable of monitoring model performance in real-time to ensure high-quality service delivery.

8 Partitioned Models:

Uber's innovative approach streamlines the machine learning process through a partitioned model system. Instead of juggling the complexities of hundreds of isolated models, the system employs a hierarchical structure. This allows the models to leverage data from various geographical partitions—ranging from a global scale down to specific countries or cities. By doing so, it not only simplifies management and deployment but also enhances the models' relevance and performance by training them on more targeted, localized datasets. This method efficiently tailors predictions and insights to the unique characteristics and trends of each area, providing a more nuanced and actionable intelligence

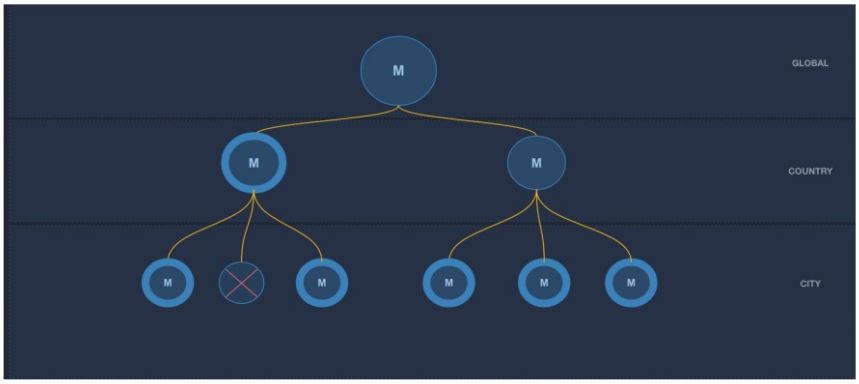

Figure 2: llustration of a partitioned model: the red cross means that a model is not trained due to lack of data in this partition, or that its model performance is worse than the one from its parent's

9 UberEATS ETD Use Case:

This section expands on the UberEATS ETD model, which utilizes a partitioned model approach to predict delivery times using data from individual cities The ETD model uses different types of features as follows:

- Restaurant features such as location, average meal preparation time, average delivery time, average demand during lunch
- Contextual features such as the time of day, or day of week
- Order features such as the number of items or the total cost
- Near real-time features such as information about the past N orders

UberEATS order and context features are supplied by the app. Restaurant and near real-time features are obtained by the service backend from the Feature Store.

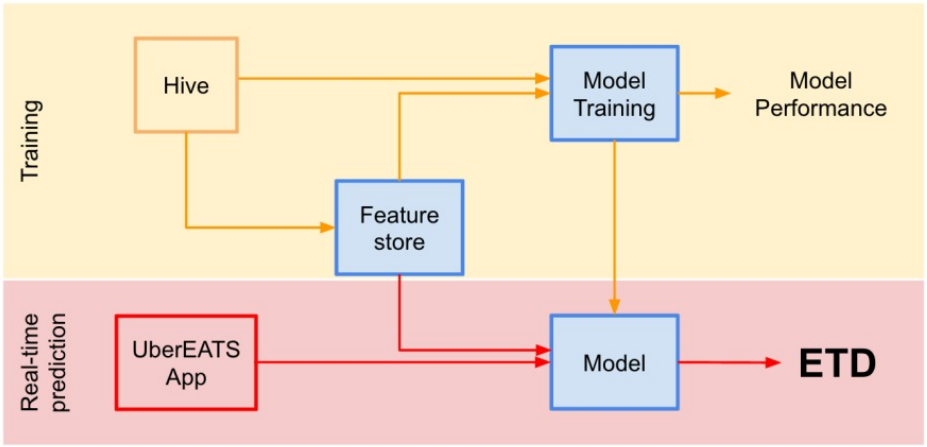

Figure 3: UberEATS Etd

The process encompasses both the training phase and the prediction phase, where the model leverages the Feature Store to access requisite features. Specifically, the model em-

ploys gradient-boosted trees as its predictive algorithm and undergoes training through Spark MLlib, a scalable machine learning library. Upon deployment, these models are activated to deliver predictions in real-time, enhancing the UberEATS application's functionality as soon as a user initiates the app.

10 Challenges and Solutions:

The challenges section elaborates on the issues faced during real-time service, including the need for low latency and the ability to handle high-traffic periods. The solution described involves the use of TChannel, a high-performance RPC framework developed by Uber, and the provision of live monitoring and automated model deployment.

11 Implementation:

This repository contains an example implementation of an MLOpsTool and a Flask-based visual dashboard for managing machine learning model versions and datasets.

- MlOps Repository