Holistic Traffic Prediction for Smart Cities: A Full-Cycle Approach

94879 - Foundations of Operationalizing AI
Heinz College of Information Systems and Public Policy
Carnegie Mellon University
Fall 2024

By:

Michael Bogush

Shreyansh Goenka

Simon Isaza-Villegas

Rujuta Parulekar

Era Singla

1. Introduction

Objective of the Project

The primary objective of this project is to develop, deploy, and monitor a machine learning model for real-time traffic prediction using the METR-LA dataset. Building upon previous experience with Kafka for real-time predictions, this project expands the scope to encompass the operationalization of machine learning models in a real-world context. Advanced tools such as Kubeflow are utilized for model experimentation and tracking, Kubernetes for deployment, and Evidently for real-time model monitoring.

The ultimate goal is to optimize traffic management systems in Los Angeles by providing accurate predictions of traffic conditions. This initiative aims to reduce congestion and improve the flow of vehicles across the city, thereby addressing one of the most pressing urban mobility challenges.

Project Scope

The project is divided into three main phases. The first phase involves model experimentation, where we experimented with two machine learning models—Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU)—to forecast traffic speeds. Although Kubeflow was initially considered for tracking, the final model tracking and experimentation were managed using MLflow to ensure better version control and integration with deployment systems.

In the second phase, the best-performing model was containerized using Docker and deployed in a scalable Kubernetes environment. This enabled real-time serving of predictions via a RESTful API, making the model accessible for practical applications in traffic management.

The third phase focused on model monitoring. The integration of the Evidently monitoring dashboard ensured that the deployed model maintained optimal performance over time. This phase included setting up alerts for model drift or performance degradation, allowing for proactive maintenance and updates.

Learning Outcomes

By completing this project, we aimed to achieve several key learning outcomes. We built and evaluated two machine learning models on the METR-LA dataset, focusing on time-series forecasting techniques suitable for traffic prediction. Deploying the chosen model using Docker and Kubernetes provided hands-on experience with real-time, scalable deployment practices.

Implementing real-time model monitoring using Evidently ensured robust performance in production environments, teaching us how to maintain and monitor machine learning models effectively. Additionally, we developed communication skills by presenting the project's key findings through a video presentation and a live class Q&A session.

Scenario Description and Context

As Los Angeles continues to experience population growth, traffic congestion has become a critical issue affecting both the economy and the quality of life of its residents. Traffic management systems require predictive capabilities to efficiently manage vehicle flow and mitigate congestion. This project addressed this need by developing, deploying, and monitoring a traffic flow predictive model using the METR-LA dataset to optimize traffic management in Los Angeles.

Dataset

The dataset used in this project is the METR-LA Traffic Dataset. It consists of speed readings from 207 loop detectors across Los Angeles. The readings are recorded every five minutes, creating a rich temporal sequence that captures traffic conditions citywide. This dataset is ideal for time-series forecasting and machine learning applications aimed at predicting traffic speeds and detecting congestion patterns. The features include traffic speed, timestamps, and spatial data for each detector.

---

2. Phase 1: Model Experimentation

Model Building

Two models were selected for experimentation: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). The LSTM model is an advanced recurrent neural network capable of learning long-term dependencies in sequential data, making it suitable for time-series forecasting tasks such as traffic prediction. The GRU model is a variant of the standard recurrent neural network that utilizes gating mechanisms to control the flow of information, offering computational efficiency and potentially faster training times.

Both models were implemented using TensorFlow and were designed to capture the temporal dependencies inherent in traffic data. The architectures included a recurrent layer followed by a dense layer to map the outputs to the desired prediction space.

Data Preprocessing

The METR-LA dataset required several preprocessing steps to prepare it for modeling. Missing data were handled by interpolating missing values, which maintained data

continuity and ensured that the temporal patterns were preserved. Normalization was applied using Min-Max scaling to bring all features within the range of 0 and 1, which is essential for neural network models to perform optimally.

Feature selection involved choosing relevant features such as traffic speed and timestamps. To enhance temporal feature representation, additional features like time-of-day and day-of-week indicators were added. This allowed the models to capture periodic patterns in traffic flow.

Experimentation with MLflow

Initially, we explored the use of Kubeflow for managing and tracking the experiments. However, we faced challenges related to package compatibility and resource overheads. To address this, we shifted to MLflow for the final setup. MLflow allowed us to effectively manage the experiment tracking, hyperparameter tuning, and model versioning while offering simpler integration with the deployment pipeline.

Experiment tracking with MLflow involved logging hyperparameters, model architectures, training metrics, and artifacts. This comprehensive tracking ensured that all experiments were reproducible and that the impact of different hyperparameters could be systematically analyzed.

Hyperparameter tuning was conducted systematically, adjusting parameters such as learning rates, batch sizes, and the number of epochs to optimize model performance. MLflow's version control facilitated easy rollback and comparison between different experimental setups, which was essential for reproducibility and collaboration among team members.

Model Evaluation

Models were evaluated using Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared (R^2) as performance metrics. The results from the PowerPoint presentation are as follows:

	LSTM	**GRU**
MAE	0.0408	0.044
MSE	0.0096	0.0104
R^2	0.9099	0.9021

These values indicate that the LSTM model outperformed the GRU model across all metrics. Visualizations such as bar charts and line graphs were generated to compare model performances.

Key Findings

The LSTM model consistently outperformed the GRU model across all metrics. The lower MAE and MSE values for the LSTM model indicated that its predictions were more accurate and less prone to large errors. Furthermore, the higher R^2 score of the LSTM model demonstrated its superior ability to explain the variance in traffic data. Based on these results, the LSTM model was selected for deployment.

---

3. Phase 2: Model Deployment

Model Packaging

The selected LSTM model was containerized using Docker to ensure consistency across different deployment environments. The Dockerfile specified the base image (python:3.8-slim), installed necessary dependencies such as TensorFlow and Flask, and copied model files into the container. This containerization encapsulated the model and its dependencies, simplifying the deployment process.

A Flask application was developed to serve the model predictions via a RESTful API. This involved creating an endpoint (POST /predict) that accepts input data and returns predicted traffic speeds. The API was designed to be stateless and scalable, facilitating integration with other systems and services.

Kubernetes Deployment

The deployment process involved configuring Kubernetes YAML files to define deployments, services, and ingress controllers. The deployment configuration specified details such as the number of replicas, container image to use, and resource allocations. Services were defined to expose the deployment internally within the cluster and externally to users.

Scalability considerations were addressed by configuring the deployment for multiple replicas and setting up Horizontal Pod Autoscaling based on CPU utilization. This ensured that the system could handle increased load and provided high availability.

API Documentation

The API can be accessed via the POST /predict endpoint. The request format requires a JSON object containing an array of normalized feature values corresponding to the input features expected by the model. The response format is a JSON object containing the predicted traffic speed. Sample requests and responses are provided in the API documentation.

---

4. Phase 3: Model Monitoring

Monitoring Setup

Evidently was integrated into the model to monitor performance metrics in real-time. The Evidently dashboard was deployed within the Kubernetes cluster, allowing for seamless integration with the deployed model. Metrics such as input feature distributions, prediction distributions, MAE, RMSE, and Mean Absolute Percentage Error (MAPE) were tracked.

Key insights from the monitoring phase include stable MAE values around 0.05, indicating consistent prediction quality. The Mean Error remained close to 0, suggesting that the model's predictions were unbiased. However, MAPE values were higher for extreme traffic conditions, signaling a need for further tuning to improve performance in such cases.

Alerts and Maintenance

Alerts were configured by setting thresholds for performance metrics, such as an MAE exceeding 0.06, to trigger notifications. A maintenance plan was established to review model performance periodically and retrain the model if significant drift was detected. This approach ensured that the model remained reliable over time.

---

5. Final Thoughts and Lessons Learned

Challenges Faced Across Phases

Throughout the project, several challenges emerged, each offering valuable learning experiences in operationalizing AI within real-world contexts.

One of the major challenges was managing and preprocessing the high-dimensional traffic data from the METR-LA dataset. Handling missing data, ensuring temporal continuity, and selecting the appropriate features were crucial tasks for building a reliable model. The large volume of data necessitated careful consideration of memory management and processing time, particularly during the model training phase. Balancing accuracy and computational efficiency required a deep understanding of data engineering techniques.

Hyperparameter tuning also presented a significant challenge. Extensive experimentation was required to identify the optimal settings for various model parameters, such as learning rates, batch sizes, and the number of epochs. Initially, Kubeflow was explored to handle experiment tracking and hyperparameter tuning. However, due to challenges related to package compatibility, resource overhead, and overall complexity, we faced difficulty integrating Kubeflow into our workflow. These challenges included issues with connections to the ports to display user interface in the local machine. After building the docker image, the incompatibility issues started to come up. Additionally, we also faced issues with setting up Kubeflow on a Windows machine and inconsistent deployment.

As a result, the team transitioned to MLflow, which provided a more streamlined and efficient setup for managing experiments, hyperparameter tuning, and versioning of the models. MLflow's simplicity and seamless integration with our deployment pipeline allowed us to overcome these initial hurdles and proceed more efficiently with the final model tracking and tuning process.

Another challenge was configuring the Kubernetes environment for scalable and reliable deployment. Kubernetes provides powerful tools for orchestrating containers, but navigating its complexities required a thorough understanding of container orchestration. Properly configuring the deployment files (e.g., YAMLs), exposing services, and ensuring efficient resource allocation were key tasks. Networking issues, such as ensuring external access to the model's API and configuring load balancing, had to be carefully managed. Additionally, ensuring fault tolerance and auto-scaling to handle fluctuating traffic loads added further complexity to the deployment process.

The integration of the Evidently monitoring dashboard into the deployment pipeline was another area that posed challenges. Ensuring compatibility between the model's output and the monitoring system required a clear understanding of the interface between the deployed model and Evidently. Configuring appropriate metrics for real-time tracking and understanding how to interpret performance data also demanded significant effort. Setting up meaningful alerts to track model drift and degradation in real time was crucial for ensuring long-term model stability, but it required a deep understanding of both the model's behavior and the monitoring tool's capabilities.

Additionally, a significant technical hurdle we faced involved software version and package compatibility issues. As the project evolved, maintaining compatibility between different versions of the libraries and frameworks, such as TensorFlow, Flask, and Docker, became increasingly difficult. Version mismatches frequently caused errors that disrupted our workflow and increased debugging time. This experience underscored the importance of setting up a stable, consistent environment from the beginning.

To overcome this challenge, we recognized the necessity of establishing a clean, isolated environment containing only the required libraries and dependencies. Installing unnecessary packages not only increased the risk of compatibility issues but also added unnecessary overhead to the application, impacting both performance and maintainability. The use of virtual environments (such as venv or conda) and Docker containers proved essential in mitigating these issues. These tools ensured that only the necessary dependencies were installed, reducing the complexity of managing dependencies and making the application more lightweight, efficient, and less prone to version conflicts. Moreover, they facilitated smoother transitions between local development and production environments.

Additionally, maintaining a clear and explicit record of package versions using files such as requirements.txt or environment.yml was crucial. This allowed team members to work in consistent environments and minimized discrepancies between local development and deployment in Kubernetes. It also made it easier to reproduce the environment at different stages of the project, reducing the risk of unforeseen errors in production.

These challenges underscored the importance of thorough planning, systematic testing, and maintaining flexibility in workflows when operationalizing machine learning models. Each difficulty provided opportunities to refine our approach and improve the overall robustness of the project, especially in ensuring compatibility, scalability, and long-term performance stability.

Recommendations for Future Work

For future iterations, exploring advanced modeling techniques such as Diffusion Convolutional Recurrent Neural Networks (DCRNN) could potentially improve performance. These models are specifically designed for traffic forecasting and can capture spatial dependencies more effectively.

Enhanced monitoring could be achieved by implementing more sophisticated tools and automated retraining pipelines. Incorporating additional metrics and anomaly detection algorithms could provide deeper insights into model performance.

Broadening the application's scope to test the model's applicability to other cities and incorporating additional data sources, such as weather and event data, may further improve prediction accuracy. This expansion could demonstrate the model's generalizability and value in different contexts.

---

6. Conclusion

The project successfully developed, deployed, and monitored a real-time traffic prediction model using the METR-LA dataset. The LSTM model demonstrated superior performance and was effectively operationalized in a scalable environment. Continuous monitoring ensured the model's reliability over time. This project has the potential to significantly impact traffic management systems in Los Angeles by providing accurate traffic predictions, aiding in congestion reduction, and improving urban mobility.

When working with multi-dimensional data, visualizing and monitoring model performance can be challenging, particularly with tools like Evidently, which typically accepts 2D data. Our traffic prediction project, utilizing speed data from 207 detectors in LA, encountered this issue. The data's 3D structure (samples, time steps, detectors) needed transformation to comply with Evidently's requirements. We successfully flattened the data by considering each feature as "detector i at time step j," enabling compatible visualization. This experience highlights the importance of adaptable data manipulation for effective model monitoring and visualization. Evidently's 2D data requirement can be met with creative data reshaping, demonstrating the need for flexible data transformation strategies when working with multi-dimensional data. This transformation allowed us to leverage Evidently's capabilities for monitoring and analysis, ensuring seamless integration and effective model evaluation. By reshaping data to meet tool requirements, we can unlock comprehensive model insights and optimization opportunities.

---

7. References

- Li, Y., Yu, R., Shahabi, C., & Liu, Y. (2018). Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. *Proceedings of the International Conference on Learning Representations (ICLR)*.
- METR-LA Dataset Repository: https://github.com/liyaguang/DCRNN
- TensorFlow Documentation: https://www.tensorflow.org/
- Kubeflow Documentation: https://www.kubeflow.org/
- Kubernetes Documentation:
- Evidently AI Documentation:

---

8. Appendix

Visualizations, code snippets, and detailed API documentation are provided in the appendices accompanying this report.