Design Document for Taxi Fare Prediction System using Big Data Analytics

Hrutik Naik, Shrumi Dedhia, Aditya Kumar

Introduction:

The NYC TLC Taxi Fare Prediction project leverages an innovative ML pipeline that harnesses the power of Amazon SageMaker, AWS CodePipeline, and AWS CodeDeploy to predict taxi fares in New York City. With an emphasis on data-driven accuracy and efficiency, this project represents a crucial intersection of data science and real-world application. By incorporating a publicly available New York green taxi dataset, the project seeks to build a robust model that accurately estimates taxi fares, catering to the specific dynamics of the city's taxi services.

The architecture of the system revolves around an intricate deployment pipeline, facilitated by Amazon SageMaker's comprehensive suite of tools for model development, training, and deployment. Additionally, AWS CodePipeline orchestrates the automated release pipeline, ensuring swift and reliable updates. The incorporation of AWS CodeDeploy introduces the blue/green deployment strategy, reducing downtime and mitigating risks by running two identical production environments, enabling seamless transitions between the "blue" and "green" endpoints.

Moreover, the integration of Amazon API Gateway provides a seamless interface for external access to the ML model's functionalities, ensuring secure and efficient communication. Simultaneously, AWS CloudWatch offers comprehensive monitoring and observability, allowing real-time tracking of performance metrics and providing actionable insights into the system's health and performance. This integration ensures robust management and monitoring of the system, enabling swift responses to potential issues and ensuring a seamless user experience.

This report delves into the intricate workings of the NYC TLC Taxi Fare Prediction project, highlighting the significance of integrating diverse AWS services, including Amazon API Gateway and AWS CloudWatch, to create a comprehensive and reliable ML deployment pipeline. By focusing on the intricacies of the project's architecture, data preprocessing, model training, and deployment, the report aims to provide a comprehensive understanding of how modern ML and Big Data technologies can be harnessed to derive actionable insights and predictions within the context of the NYC taxi fare ecosystem.

Architecture Overview:

The architecture consists of four key components: Data Ingestion, Data Processing, Machine Learning Model Training, Model Deployment and Model Monitoring. These components interact through a well-defined pipeline, allowing for effective data handling and predictive model deployment.

Amazon SageMaker:

- Data Preprocessing: SageMaker offers a range of tools and capabilities for data preprocessing, allowing the project to handle the New York green taxi dataset efficiently. This dataset is processed and transformed to ensure its suitability for training the ML

model. Preprocessing steps may include data cleaning, normalization, feature engineering, and data splitting into training, validation, and test sets. These steps are crucial for ensuring the accuracy and reliability of the trained model.

- Model Training: The project utilizes the XGBoost algorithm, which is a built-in algorithm in SageMaker known for its efficiency in handling large datasets and providing high prediction accuracy. XGBoost is an optimized gradient boosting library that is widely used for regression and classification tasks. It works by iteratively training multiple weak models and combining them to create a strong predictive model. During the training phase, the algorithm learns from the New York taxi dataset, capturing patterns and relationships between various input features such as trip duration, passenger count, and trip distance, and the corresponding taxi fares.
- Model Deployment: Once the model is trained, SageMaker facilitates its seamless deployment, allowing the integration of the trained model into a production environment. The deployment process ensures that the model is ready to make real-time predictions and handle incoming inference requests. SageMaker's deployment capabilities include automatic scaling to handle varying workloads, monitoring for performance and resource utilization, and integration with other AWS services for streamlined operations.

AWS CodeDeploy:

- Blue/Green Deployment Strategy: AWS CodeDeploy facilitates the creation of two separate and identical production environments: the "blue" and "green" endpoints. This strategy allows the project to have two fully operational and independent environments that can run simultaneously. The "blue" environment represents the currently active production environment, while the "green" environment serves as the new target environment for the deployment of the updated ML model. Having these parallel environments ensures continuous availability of the application during the deployment process.
- Risk Minimization and Downtime Reduction: By implementing the blue/green deployment strategy, CodeDeploy minimizes the risks associated with deploying new updates or changes to the production environment. The simultaneous existence of both the blue and green endpoints enables the project to minimize potential disruptions to the application's functionality and performance. This strategy significantly reduces downtime, ensuring that end-users can access the application without interruptions or errors during the deployment phase.
- Traffic Switching: CodeDeploy seamlessly manages the switching of traffic between the blue and green endpoints, allowing for a smooth transition from the old version to the updated version of the ML model. During the deployment process, a fraction of the traffic, typically a small percentage, is gradually redirected to the green endpoint for testing and validation. This controlled traffic switching ensures that any potential errors or issues can be identified and addressed before the full deployment is completed. Once the deployment is validated, the traffic is gradually shifted entirely to the green endpoint, making it the new active production environment.

Amazon API Gateway:

- The Amazon API Gateway acts as a fully managed service that makes it easier for developers to create, publish, maintain, monitor, and secure APIs at any scale. In our project, the API Gateway serves as the entry point for your RESTful API that exposes the ML model's functionality for external access. Its primary functions include:
    - Creation and Deployment of APIs: The API Gateway allows to create RESTful APIs that can be used to access the machine learning model. You can define the API's structure, methods, request/response formats, and authentication mechanisms.
    - Integration with AWS Lambda: Integrating the API Gateway with AWS Lambda functions to execute the backend logic required for processing the incoming requests. This integration allows for the seamless execution of your ML model predictions.
    - Request and Response Handling: The API Gateway efficiently manages the incoming HTTP requests and routes them to the appropriate backend services. It also handles the responses from these services and sends them back to the client applications.
    - Security and Access Control: The API Gateway provides various security mechanisms such as API keys, IAM (Identity and Access Management) roles, and custom authorizers to control access and ensure that only authorized clients can interact with your API.

AWS CloudWatch:

- AWS CloudWatch is a monitoring and observability service that provides data and actionable insights to monitor applications, respond to system-wide performance changes, optimize resource utilization, and get a unified view of operational health. In our project, AWS CloudWatch is used for:
    - Logging and Monitoring: CloudWatch aggregates and stores logs, allowing us to monitor and troubleshoot applications in real time. It collects and tracks metrics, such as API latency, response codes, and error rates, to provide a comprehensive view of the performance and health of your API endpoints.
    - Alarms and Notifications: CloudWatch enables us to set alarms on specific metrics to automatically initiate actions or send notifications when certain thresholds are crossed. This feature allows us to proactively manage and respond to any issues or anomalies in your system.
    - Dashboards and Visualization: CloudWatch offers customizable dashboards that allow us to visualize key performance metrics and operational data. We can create comprehensive dashboards that provide a real-time view of our API's performance, helping us identify trends, patterns, and potential issues quickly.
    - Integration with Other AWS Services: CloudWatch seamlessly integrates with various AWS services, allowing you to monitor not only your API Gateway but

also other AWS resources, such as Amazon S3, EC2 instances, and Lambda functions, providing you with a holistic view of your entire application stack.

By integrating these AWS services, the system effectively streamlines the entire process of developing, training, deploying, and monitoring ML models for predicting taxi fares in New York City. This comprehensive approach ensures high accuracy, reliability, and scalability, making it a robust solution for handling large-scale datasets and real-time prediction requirements.

Challenges Overcome:

- Curation: The data preprocessing and feature engineering steps exemplify the curation aspect. We handled the cleaning, transformation, and organization of the dataset to ensure its quality and suitability for training the ML model. This process involved tasks such as handling missing values, normalizing data, and creating new features from the existing data points.
- Storage: The project involved storing and managing the large dataset, including the training and validation data, in an efficient and scalable manner. Amazon S3 (Simple Storage Service) likely played a significant role in providing a reliable and cost-effective storage solution for the project, allowing it to securely store and retrieve data when needed.
- Analysis: The entire project was centered around data analysis, specifically the development of a predictive model based on the analysis of historical taxi trip data. Through the utilization of the XGBoost algorithm and other data analysis tools, we were able to extract valuable insights from the dataset to train the ML model effectively.
- Visualization: The process of visualizing the data and model results is crucial for gaining a comprehensive understanding of the model's performance. This visualization aids in interpreting the model's predictions, identifying patterns, and evaluating the accuracy of the model's outputs.

System Architecture:

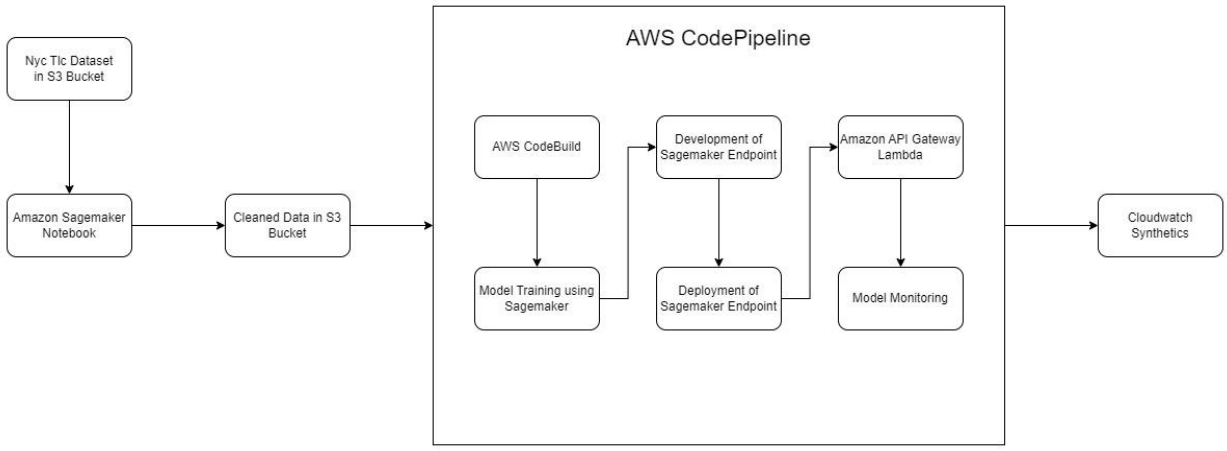

Fig 1: System Architecture UML Diagram

Figure Description:

To initiate the development of our predictive system, we first upload the converted CSV file containing New York City TLC (Taxi and Limousine Commission) ride details to an S3 bucket. This step allows us to preprocess and construct our prediction model using the dataset.Amazon Sagemaker Notebook plays a crucial role in orchestrating the creation of our predictive system. The initial phase involves preprocessing the dataset, entailing tasks such as feature engineering and eliminating redundant data. Following this, the refined dataset is stored back into the S3 bucket, enabling us to commence the model-building process.The meticulously cleaned data is then returned to the S3 bucket, serving as the starting point for the subsequent stages. AWS CodeBuild comes into play at this juncture, undertaking operations on the dataset and applying the XGBoost algorithm to generate the fare prediction model.Upon the completion of model training, our focus shifts towards the development of a Sagemaker Endpoint, which serves as an essential component of the predictive system. Thorough testing of the Endpoint ensures its readiness for deployment.Once the Endpoint has been thoroughly tested, it is deployed, enabling us to commence the prediction process. Amazon API Gateway handles the incoming prediction requests, ensuring a seamless interaction between the users and the predictive system.Model Monitoring, facilitated through CloudWatch, is employed to oversee the overall performance and health of the system, ensuring its effective functionality and performance.

Activity Diagram:

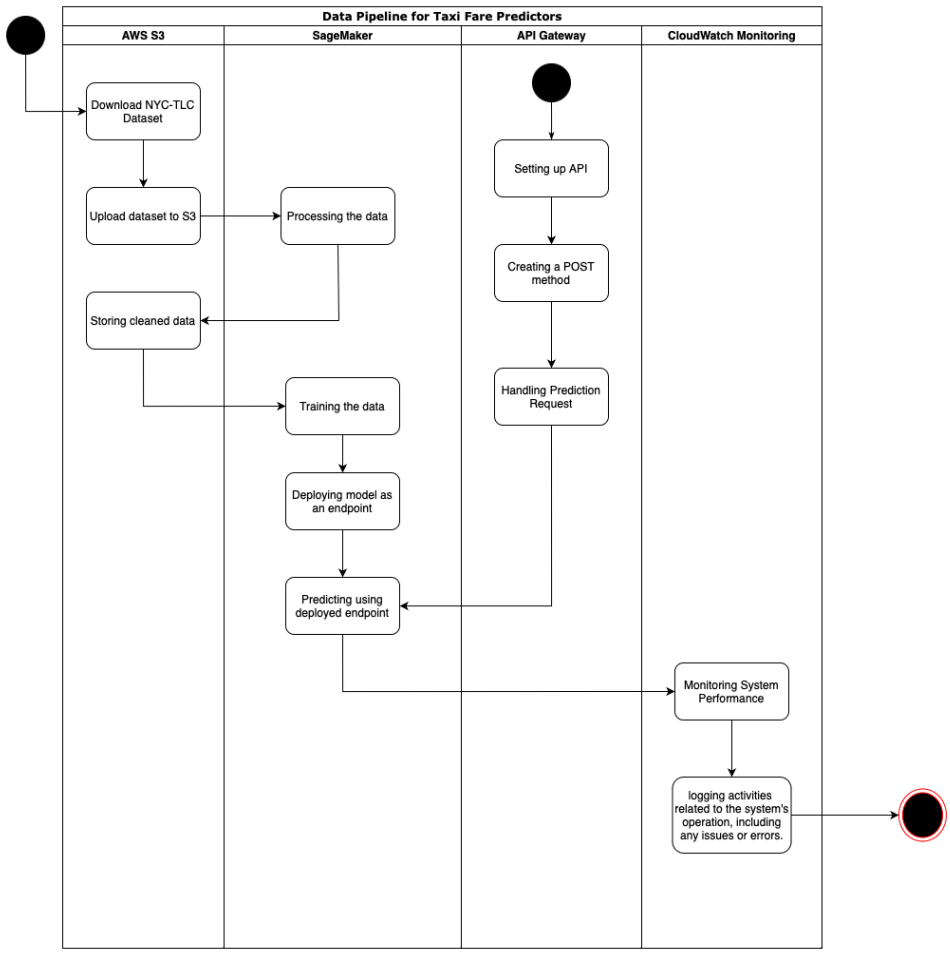

Fig 2: Activity UML Diagram

Figure Description:

This activity diagram illustrates the workflow for taxi fare prediction using AWS services. It shows how AWS S3 stores the input data, AWS SageMaker trains the machine learning model, API Gateway handles prediction requests, and AWS CloudWatch monitors system performance and logs activities.

Use Case Diagram:

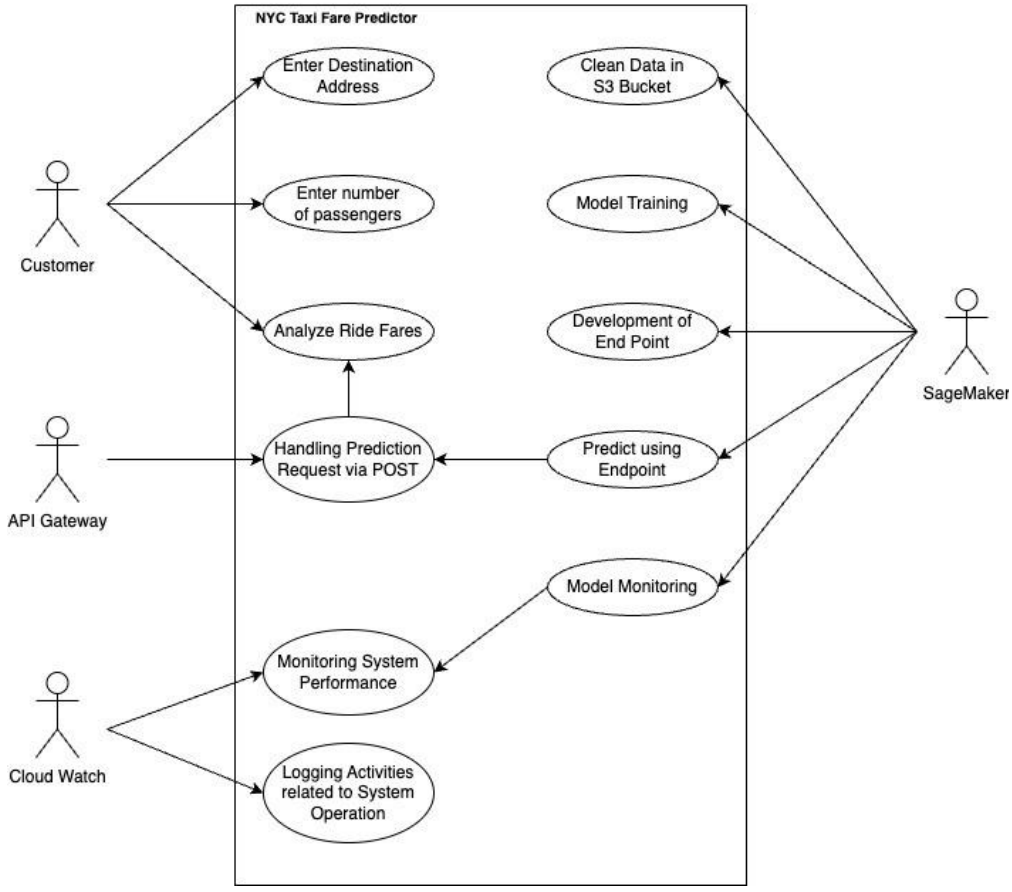

Fig 3: Use Case UML Diagram

Figure Description:

In UML, use-case diagrams model the behavior of a system and help to capture the requirements of the system. Use-case diagrams describe the high-level functions and scope of a system. Use-case diagrams illustrate and define the context and requirements of either an entire system or the important parts of the system.

For our project, we have the following actors/system that are responsible for predicting the fare of taxi in New York City:

1) Customer: A customer is the user who is going to use the system to book a cab. His interaction is limited to the front end of the system where he enters his destination, number of passengers and finally selects the cab based on the predicted fare generated by the system.

2) SageMaker: AWS SageMaker does the role of cleaning the data in S3 bucket, training and maintaining the model, developing an endpoint and predicting using that endpoint.

3) API Gateway: API Gateway does the role of handling prediction requests via POST by interacting with the endpoint created by SageMaker and then using the results received to provide options to the user for booking the taxi.

4) CloudWatch : CloudWatch ensures that the system is running healthy by interacting with SageMaker to monitor the model and system performance. It also helps create an overview of the running by actively logging the reports onto the system

Conclusion:

The proposed architecture for the taxi fare prediction system leverages the capabilities of big data technologies to address key challenges in data processing and predictive modeling. By employing a Data Pipeline Architecture Style, the design ensures seamless data flow and efficient interaction between the system components, enabling the system to handle the core big data challenges effectively. The comprehensive design facilitates the development of a scalable, reliable, and accurate taxi fare prediction system, contributing to the advancement of data-driven solutions in the transportation domain.