Abstract

The following case study outlines a machine learning pipeline for the deployment of reinforcement learning (RL) models. The results from the models must be available to a variety of applications. It must also be possible to provide feedback to the models currently in production. There should be continuous monitoring in case of data drift and to ensure model performance. The models will address a range of ML problems such as speech-to-text, OCR, and object detection. Open source technologies will be preferred.

Data Ingestion and Preprocessing

You cannot achieve good model performance on bad data. That's why it's critical to choose a data ingestion system that can handle the type and amount of incoming data. The right choice of system will depend on the type of problem at hand. The data ingestion stage is the importing of data from various sources into the pipeline. That might mean everything for large files from blob storage or real time data ingestion.

In this pipeline Apache Kafka will be used for handling real-time data streaming applications. Kafka is an event-streaming platform. It can handle large amounts of data in real time. It will therefore allow our pipeline to be scalable and able to handle a variety of workloads. Kafka is a durable solution that can handle incoming data persistently. This ensures no data is lost during processing. Kafka will be critical for streaming applications like video and speech to text. However, Kafka will not be used for large static files, which will be handled by other systems more suited for batch processing.

Preprocessing is needed to ensure the models receive data that is clean, consistent and in the correct format. Apache Flink will be used for preprocessing. This involves real-time data transformation. For example this might mean transforming raw audio into normalized formats for a speech-to-text model. Flink can be used both to process streams and batch data.

Model Deployment

It would be possible to host the model directly in the fastAPI server. While this would work this approach lacks versioning control and a variety of other features. I will instead host the model with Seldon. Seldon provides a number of features that ensures reliability, scalability and long term monitoring. Most importantly Seldon offers versioning of models, much like how git can version code.

Seldon is Kubernetes-native, which allows for high scalability and availability. This is important for handling both batch processes and real-time inference. Seldon's built-in support for Prometheus monitoring will ensure that notifications are generated if the model fails to perform in real-world scenarios.

Automatic Retraining and Feedback Loop

It is critical for the pipeline to take in user correction to the model in question. This will allow for the model to adapt to data drift and other changing conditions. In real world application one cannot assume that incoming data will match the training data of the model. The feedback loop must collect, store and effectively use incoming data to improve the model.

A FastAPI server will handle incoming feedback from client applications. This feedback will eventually be stored in the blob storage with the rest of the training data. When the next training run occurs this feedback will be taken into account.

An automatic retraining can be triggered by a number of different metrics. The exact parameters for retraining will depend on the application. The job of starting retraining will be done by the FastAPI server. Some parameters to trigger retraining might be a certain volume of new data, low performance, data drift, latency or simply a scheduled retraining.

A dedicated service such as Apache Airflow or KubeFlow Pipelines could also be used to handle retraining. These can be particularly useful if there is a lot of complexity involved.

Model Monitoring

Human feedback may not be enough to detect issues with the model in production. There it is a good idea to use a tool to continuously monitor certain key metrics. Prometheus is an open source monitoring and alerting toolkit that is used to collect and store metrics. It will store key performance metrics such as response time, prediction accuracy, resource usage and error rates.