Global Big Data Conference

www.globalbigdataconference.com

Twitter : @bigdataconf

#GlobalAI

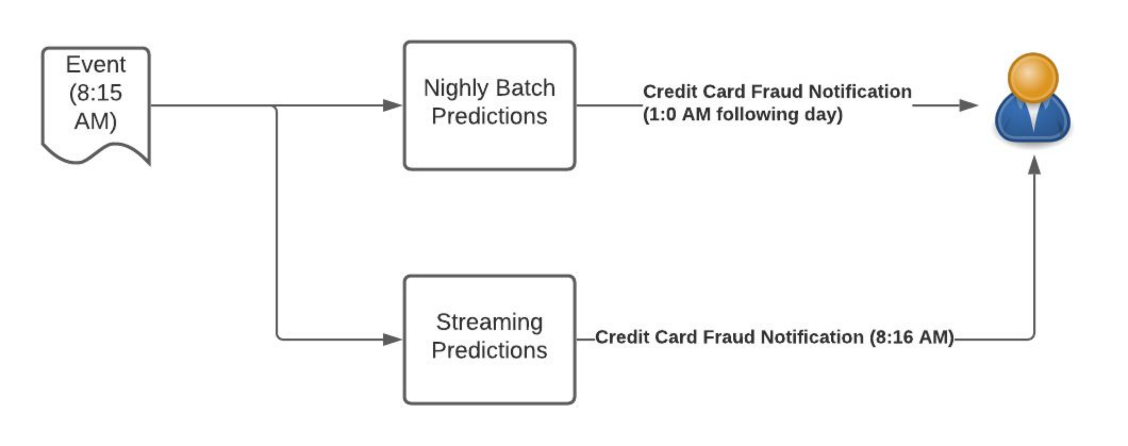
Streaming ML Ops

Data Streaming & ML Ops For Big Data & AI

Sameer Wadkar (Staff Field Engineer at Domino Data Lab)
October 2021 - Global Artificial Intelligence Virtual Conference

Need for Streaming ML

- Fraud Detection (Ex. Credit Card Fraud Detection algorithms may run per transaction)
- Network log analysis for intrusion detection
- Recommendations based on Click-Stream analysis



It is the not just the accuracy of the prediction, the timing of its delivery impacts the "actionability" and "business impact" based on the prediction

Use-Case - Credit Card Fraud Detection

- Transaction Event/features arrive in a stream
- **Online Learning Model** predicts if the transaction is a fraud
- Truth arrives later
- Model periodically **retrained** based on predictions and truth arrivals (Supervised) or just the feature arrival (UnSupervised)
  - We will use unsupervised learning algorithm Half Space Trees
- **Metrics** based on predictions and arrival of the truth
  - The metric we will demonstrate is ROCAUC

Continually refine the model & metric by learning on one data point at a time

```
#Supervised
from river import linear_model
model = linear_model.LogisticRegression()
for x, y in dataset:
    y_pred = model.predict_predict_one(x)
    Y_hat = model.predict()
    model.learn_one(x, y)

#Unsupervised
model = compose.Pipeline(
   preprocessing.MinMaxScaler(),
   anomaly.HalfSpaceTrees(seed=42)
)
model.learn_one(x)
```

In theory the model can be retrained every time the truth/features for a given prediction arrives.

What do we lose? Reproducibility of the prediction which also impacts Auditability

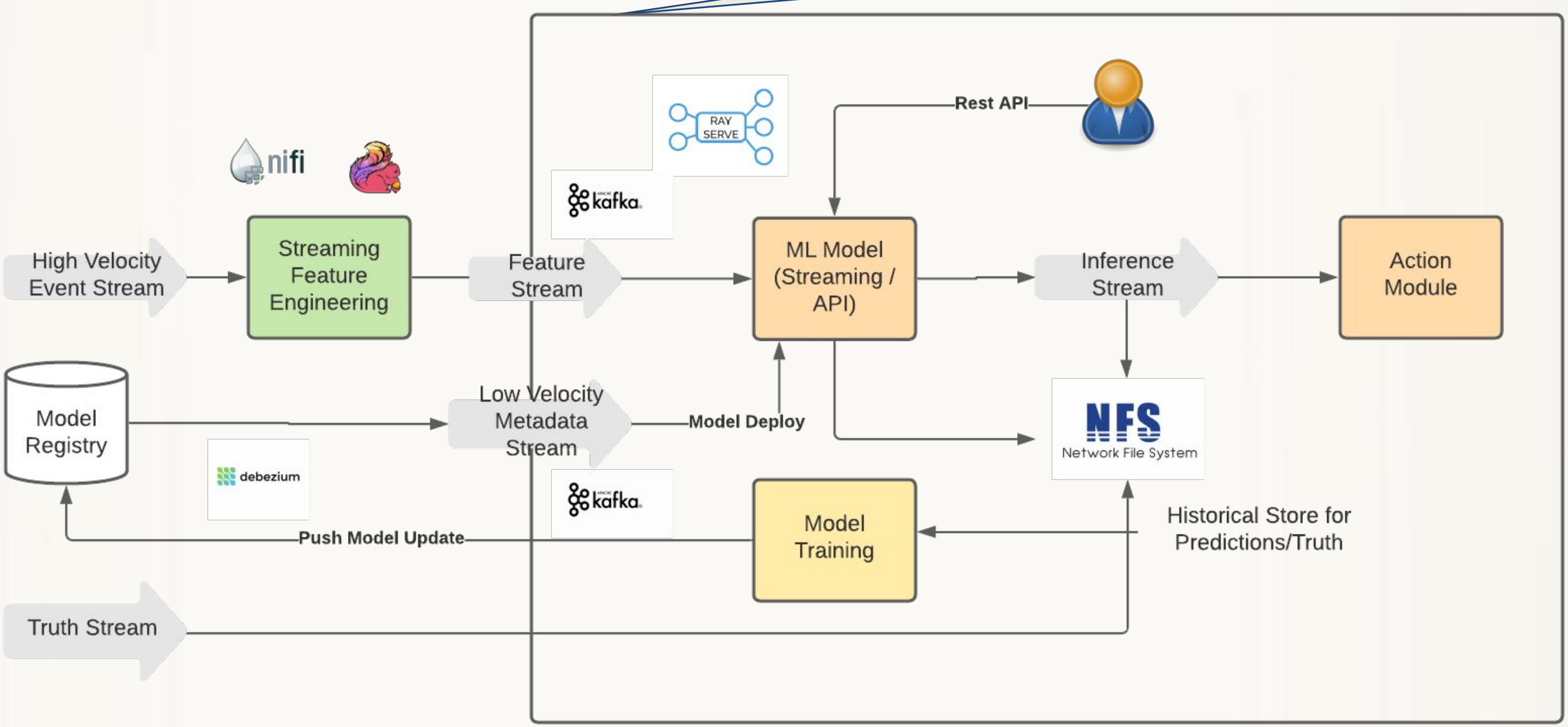
Library used - https://github.com/online-ml/river
Our Repo - https://github.com/dominodatalab/mldemo-streaming-online-learning/

Architecture must support the following Non Functional Requirements

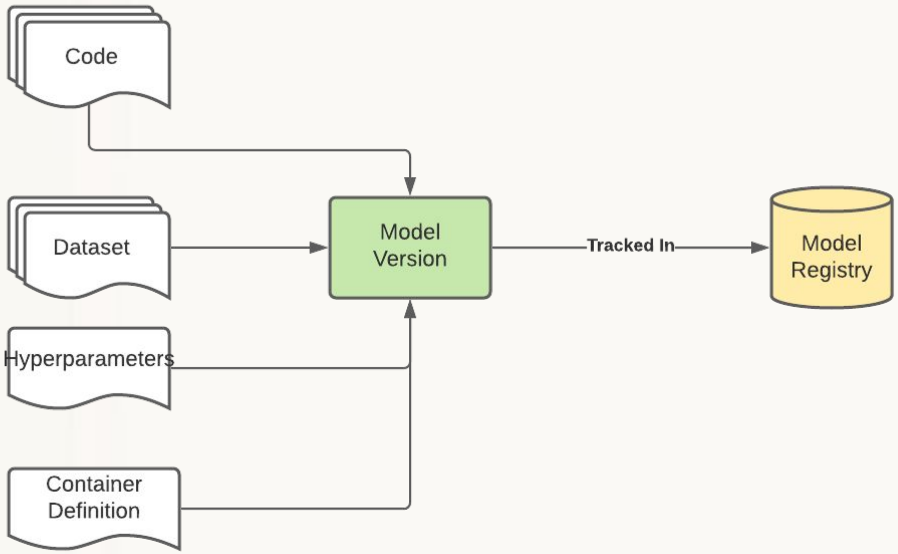
- **Scalability** - Design is agnostic to increasing loads **(Scale out)**
- **Model Versioning -** Every model version should be **reproducible**
- **Prediction Reproducibility -** Every inference should be reproducible (Model Version is part of the prediction output)
- **Multiple Deployment Modes -** Inference on Stream (Push) and API (Interactive)
- **No Deployment Downtime** - Model are continually refined and deployed without pausing the stream

Architecture - Streaming ML Inferencing

We will focus on components in this box



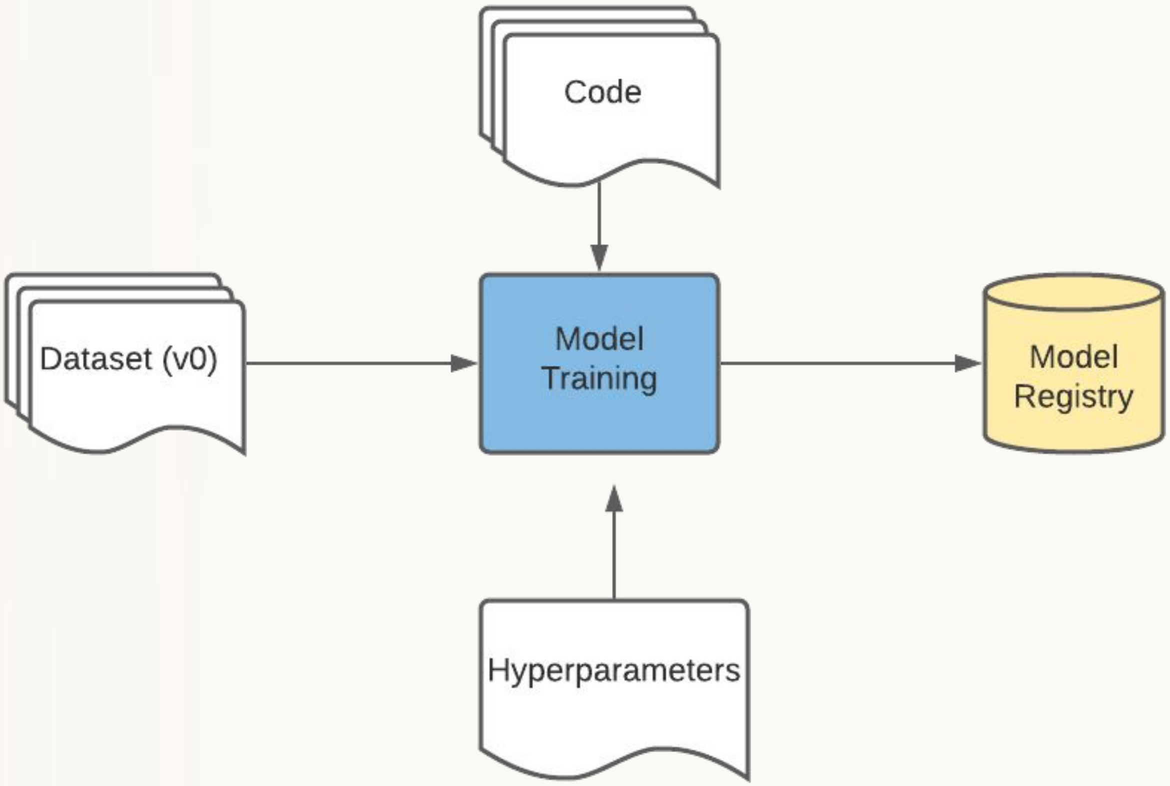
What is an ML Model Version?



Reproducibility of Model Versions is necessary for Audit purposes

[DEMO]

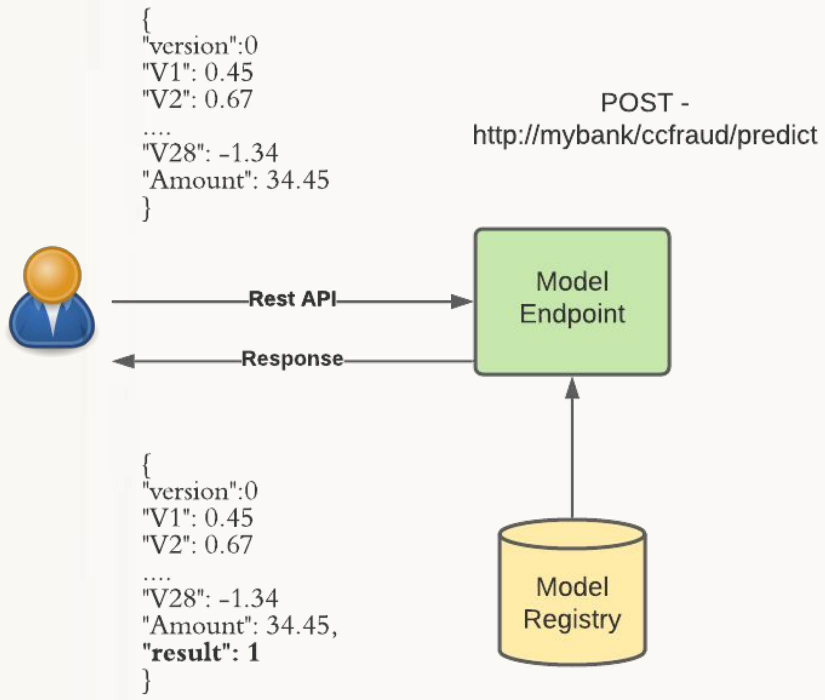
1 - Training version 0 of the model



Model Registry tracks:

1. Training Code
2. Training Dataset
3. Hyperparameters
4. Model artifact (ex. pickle file)
5. Model Metrics

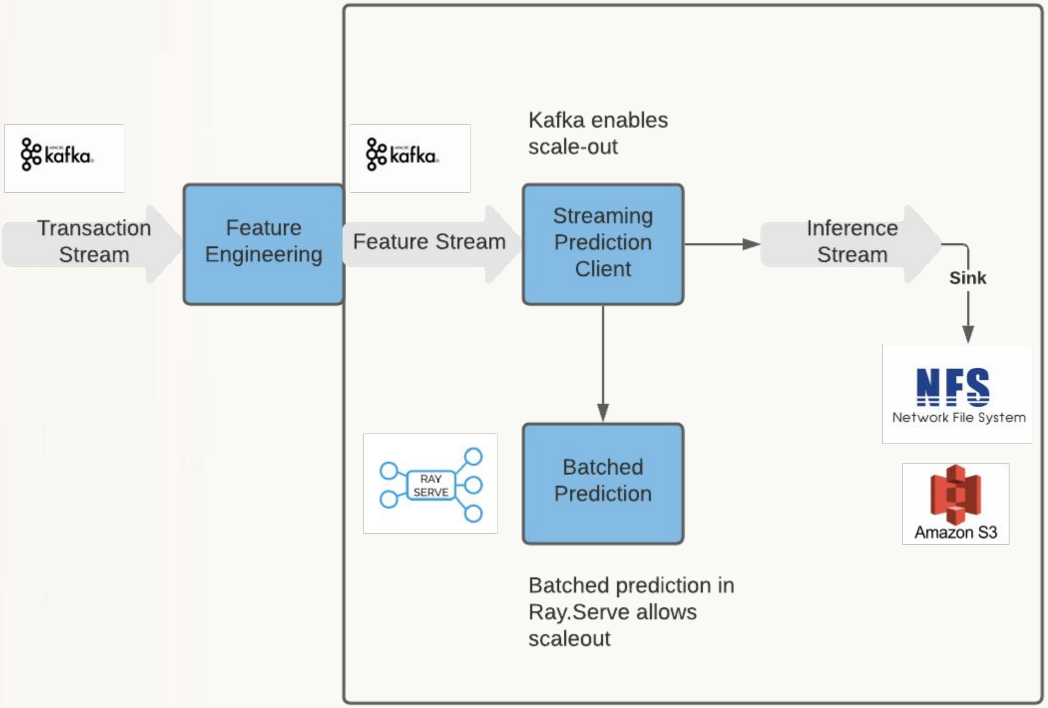
2 - Deploy Interactive Model (Rest Endpoint)



Deploy as an Interactive Endpoint

1. Simplifies testing and trials
2. Compare results across model versions

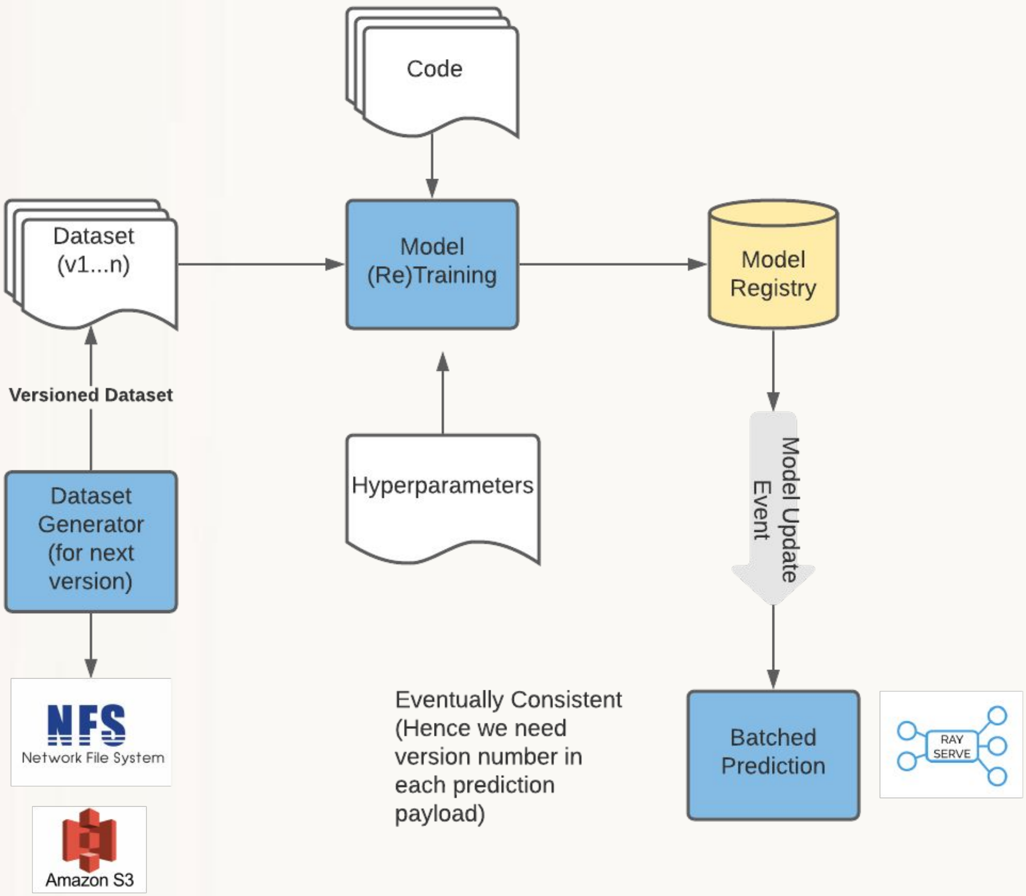
3 - Deploy Streaming Predictions



Components

1. Real world events arrive in transaction stream
2. Feature Engineering consumes the events and places features on feature stream
3. **Ray.Serve supports scale out predictions using the familiar REST methods**
4. Inferences are placed on on Inference Stream
5. Inferences and truth sink to a NFS

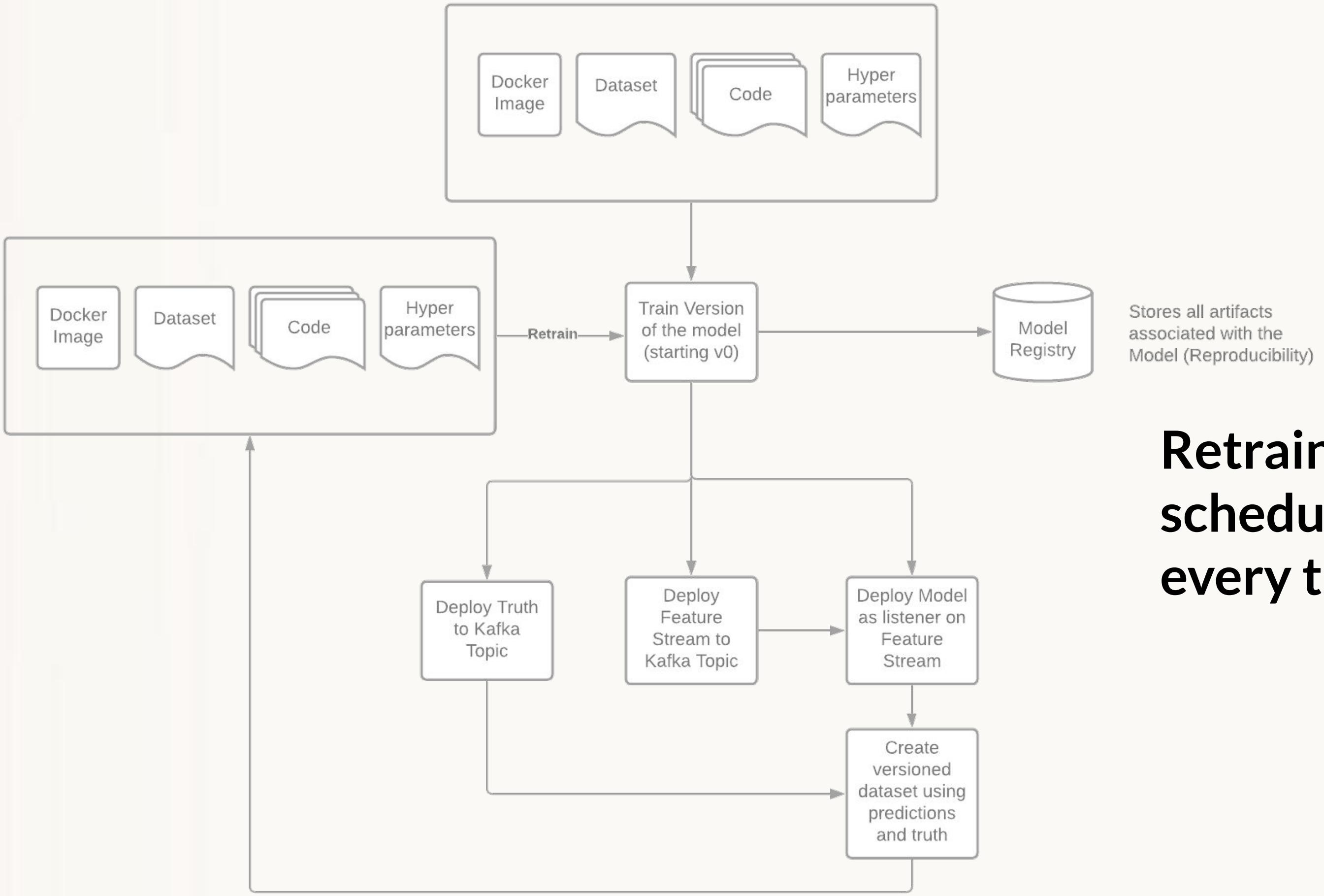
4 - Finally retrain Model on Schedule



Components

1. Periodically retrain model
2. Push model updates to the serving layer
3. Each partition of the serving layer gets the update independently
4. Version number needed in predictions for traceability and auditability

Online Learning - A Practical Approach Summary



Retraining occurs on schedule vs. on arrival of every truth datapoint

Design Considerations in Streaming ML

- **Back-Pressure Management**
  - Events are arriving faster than your systems ability to process them
- **Back-Pressure needs to be measured**
  - Prediction_Time - Event_Time (**Prediction Latency**)
  - Ingestion_Time - Prediction_Time (**Platform Backlog**)
- **Root causes of Back-Pressure**
  - Unexpected surge of events (**Efficient Auto-Scaling**)
  - Streaming Platform is sized insufficiently (**Early Scale-Out**)
    - Horizontal Scalability is an important design decision
    - Efficient local state management is crucial
    - Efficient code or protocols is important - Ex. Text Based event payload vs. Binary Protocols
    - *Cost Management will drive your design decisions*

Design Considerations in Streaming ML

- Alerts need to be **Exactly Once** for good customer experience
  - **Exactly Once** is impractical in streaming platforms
  - "**At Least Once**" and **Idempotence** are practical design methods
  - **Idempotence** - Reprocessing produces same results but can lead to repeated alerts which results in **degraded Customer Experience (CX)**
  - Processing should be **Idempotent** but CX must be **"Exactly Once"**

- Feature Engineering is not instantaneous (we assumed it is)
  - In the real world it is expensive (Especially when large amount of windowed state is used to engineer features)
  - Data arrive late & out of order in a real world system.
  - See Appendix for related talks

Resources

Library used - https://github.com/online-ml/river
Our Repo - https://github.com/dominodatalab/GlobalBigDataConf-StreamingMLOps
Dataset - https://maxhalford.github.io/files/datasets/creditcardfraud.zip

Appendix

- **Apache Kafka or Alternatives** - Event Streaming platform
- **Apache Flink** - Streaming engine and state management (Scale-out)
- **Debezium** - Push metadata changes to Kafka Stream (Enable feature engineering/model deployment without restarts)
- **Domino Data Lab** - ML Platform for ML Workflow management
- **Ray Serve** - Efficient scaleout serving layer
- **Feature Engineering and Feature Store**
  - Flink Forward San Francisco 2018: "Embedding Flink Throughout an Operationalized Streaming ML Lifecycle"
  - Flink Forward San Francisco 2019: Adventures in Scaling from Zero to 5 Billion Data Points per Day