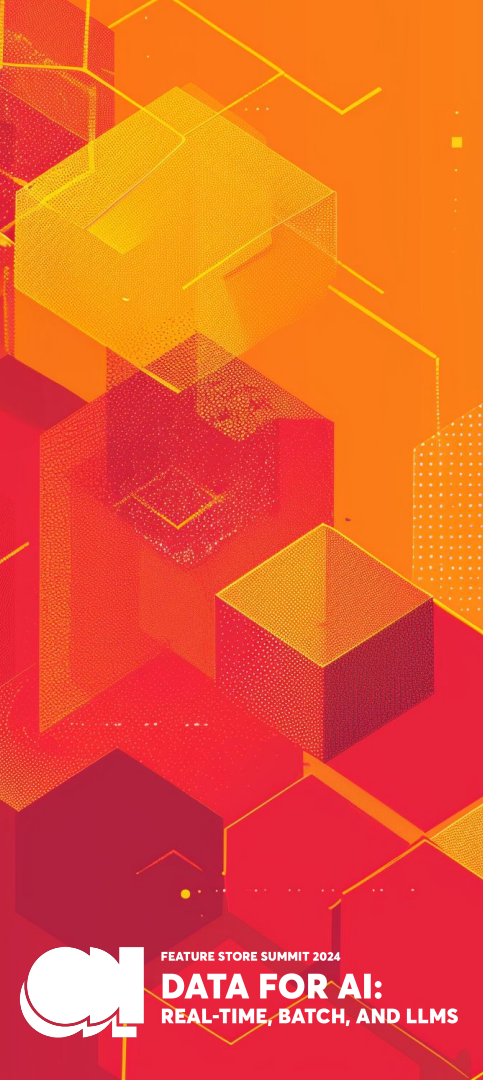

TurboML's platform to leverage fresh data for ML

Arjit Jain, Co-founder and CTO, TurboML

Organized by HOPSWORKS

What does fresh data mean for ML?

Data freshness can be defined as the delay between when the data is created and when it can be used in your application

Fresh data for ML: Making data, with a given freshness, accessible to your ML system

Example: Being able to use the user's latest transactions and behavioural data to detect fraud

Fresh data for ML: Features

Most common way to use fresh data is to update features

Feature logic + Freshness level → Mode of compute (on-demand vs async)

Aggregation based features can be heavily optimized with stream processing

Examples: Average transaction amount for a user in the last 30 mins

Is that all fresh data can do for ML?

Absolutely not!

Even though it helps provide relevant context for models, the models themselves can get outdated.

Feature freshness - Couple of seconds/mins

Model freshness - More than a few weeks/months

Is that all fresh data can do for ML?

Fraud Detection

- Labels are scarce, especially for newer fraud patterns, e.g. money mules
- Need to incorporate learnings from these to detect and prevent similar frauds as early as possible

Solution: Improve model freshness

Is that all fresh data can do for ML?

Recommendations

- Non-stationary, highly dynamic setting
- Real-time features capture only local context. Need to capture even global trends

Solution: Improve model freshness

Fresh data for ML: Beyond features

ML is highly data-driven. Fresh data can be used to

- update ML models
  - label-efficient (cold start, rare events etc)
  - concept-drifts (non-stationary, dynamic distributions)
- improve deployments (model selection, calibration etc)
- experiment and iterate faster

and much more!

How to improve model freshness?

Example: Target model freshness of 1 day

- Suppose we're training the model on past 6 months data
- Naive approach: Daily retraining jobs

Limitations:

- Independent training jobs → more likely to be bug-prone
- Highly resource intensive
- Doesn't work if training takes longer than required freshness (e.g. 1 hr freshness but model training takes 2 hrs)

Feature computation and model training

We can draw parallels between feature aggregations and model retraining

Consider the following feature:

- Average transaction amount for a user over last 24 hours, that needs to be computed every hour

Same approach that works for feature computation can work for model training

Stateful incremental compute

For feature aggregations

Fetch all transactions in the last 24 hours. Aggregate them.

Stateless approach:

For the same query after 1 hour, recompute the above from scratch.

Stateful incremental approach:

For the same query after 1 hour, look at only the changes at the ends of the 24 hr window.

Stateful incremental compute

For models

Stateless approach:

Algorithm + Data → Model

Stateful incremental approach:

Algorithm + New Data + Model State → Updated Model State

Model (yesterday) + data from yesterday → Model (today)

Stateful incremental compute

Neural network example

State is Model weights + Preprocessor state + Optimizer state

Stateful incremental model update is couple of gradient descent steps

Unlike typical aggregations: **Batch size and order matters!**

E.g. 100 online steps vs 1 step with batch of 100?

Stateful incremental compute

Libraries for incremental learning

https://github.com/online-ml/river

https://github.com/VowpalWabbit/vowpal_wabbit

https://github.com/ContinualAI/avalanche

Log and Wait

Training data preparation with feature reuse

- Log features that are being computed for making predictions
- As new labels arrive, match them with their corresponding features. These can be streamed to the training topic/bucket.

Limitations:

- Doesn't work for historical data that might require backfilling

Model deployment

Not a one-off task for most use-cases

Processes around model deployment, like

- model calibration and performance estimation
- model selection
- data quality and drift
- model testing and observability

etc. are continuously improved using fresh data.

E.g. Contextual bandit based model selection where new labels are used to compute metrics/rewards to update the bandit

TurboML at a glance

Declare your components

- Data sources
- Features
  - SQL
  - Dataframe API + Python UDFs/UDAFs
- Models
  - Out of the box algorithms
  - Python to write models
- Metrics
- Model deployment configuration

TurboML at a glance

Choose update method for components

- Data sources
  - Pull-based ingestion vs push-based
- Features
  - Streaming vs batch updates
- Models
  - Online training vs trigger-based vs ad-hoc updates
- Metrics & Model deployment
  - Streaming vs ad-hoc updates

TurboML at a glance

Tech Stack

- Apache Kafka (Real-time), Apache Iceberg (Historical)
- Apache Flink, Risingwave (Stream processing)
- Ibis (High-level DataFrame APIs)
- Apache DataFusion, DuckDB (Batch computation and retrieval)
- Apache Arrow Flight (Data transfer)
- PostgreSQL (Metadata)
- S3 object storage (Model store)
- RocksDB (Log and Wait)
- Starrocks (Real-time Analytics)

Challenges

- **Data**: Non-stationary data makes traditional challenges like data quality more pronounced.
- **Algorithmic**: Catastrophic forgetting, robustness, stability
- **MLOps**: Tough enough for static, batch setting. Much tougher for continuous streaming settings.
- **Infrastructure and Scaling**: Being able to scale with both requests and models, in a cost-effective manner.
- **People**: Intuitive interfaces in familiar environments

Questions?

LinkedIn: in/arjitj

Email: arjit@turboml.com