Metarank:
Building an open-source LTR engine on top of a feature store
This is us

- This is **NOT** a sales talk: we want feedback
- Working on personalization for almost 10 years
Personalization?

- same items
- different visitors
- different item ordering
Offline vs Online

- offline: ranking is affected by previous session
- online: ranking is affected by past actions within session:
  - Mobile/desktop
  - Traffic source / Referer
  - Landing page
  - Previous clicks & searches
e-commerce
content
social
Personalization works!

- Fashion: 6.86%
- Home & Garden: 8.99%
- Food: 7.80%
- Cosmetics: 6.24%
- Hardware & Automotive: 6.26%
- Consumer Goods: 15.29%
Déjà vu

- different companies
- different contexts
- different goals
same problems
Grebennikow's hierarchy of needs

- Data collection
- Data processing
- Feature extraction
- Stateful features
- Feature store
- Personalization
Airbnb experience

https://medium.com/airbnb-engineering/
machine-learning-powered-search-ranking-of-airbnb-experiences-110b4b1a0789
What are the options?

- ElasticSearch + ES-LTR + Spark + Python + ...
- Random shady SaaS from the internet
- Something else?
A tool to automate common parts

- data model: clicks, impressions, metadata
- feature extraction: UA, Referer, GeoIP, customer profiling
- feature store: replay, bootstrap
- typical LTR ML models: LambdaMART
Metarank

a swiss army knife of personalization
Short path

- implements parts of all levels
- only what's needed
Metarank

How would you rank items A, B, C, D for a visitor 817438?

C, D, A, B will result in best predicted CTR
Metarank
Inside Metarank
Feature store pattern

- **online**: low latency, low throughput
- **offline**: whatever latency, high throughput
• **online**: last version of values
• **offline**: time travel, point-in-time join
Feature store: Offline part
Feature store: Offline part
Point-in-time join

- join event with last value in the past - easy
- join all events to all features - 😞

Findify:
- 10M searches per day
- 24 products in search
- 50 features
Palette Feature Store

Uber-specific curated and crowd-sourced feature database that is easy to use with machine learning projects.

One stop shop

- Search for features in single catalog/spec: rider, driver, restaurant, trip, eaters, etc.
- Define new features + create production pipelines from spec
- Share features across Uber: cut redundancy, use consistent data
- Enable tooling: Data Drift Detection, Auto Feature Selection, etc.
2021
Grebennikov's law

Any sufficiently complicated ML system contains an ad hoc informally-specified bug-ridden implementation of feature storage
Hops-feast-splice

- Python API
- Online/offline mode
- Versioning, time travel
Feature store and Findify

- Simplicity & no extra dependencies
- Most features have similar high-level types
- Multi-tenancy
- Performance
Feature types

*We need not just strings and numbers*

- **Counter** - # of clicks made by a customer
- **Periodic counter** - # of clicks per day
- **Frequency** - estimate % of US in the whole traffic
- **Statistics** - estimate percentiles, min & max
- **Bounded list** - last N customer clicks
HOW Feature stores PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)

SITUATION:
THERE ARE 14 COMPETING Feature stores

14?! RIDICULOUS!
WE NEED TO DEVELOP ONE UNIVERSAL Feature store
THAT COVERS EVERYONE’S USE CASES. YEAH!

SOON:

SITUATION:
THERE ARE 15 COMPETING Feature stores
Feature store and Findify

- Cover just our needs
- Tighter integration: Flink & Scala
- FUN!
Apache Flink

- Unified stream & batch processing
- Stateful
stateful processing

- persistent state: easy to upgrade
- low-latency: no microbatches
- rich DSL: windowing, aggregations
Unified processing

- Same API for online/offline
- different runtime semantics
Metarank
Open Source

- Apache2 licensed, no strings attached
- Single jar file, can run locally
1. Import historical events: S3, HDFS, files
2. Export: state, latest features, training dataset
3. Train: XGBoost and LightGBM are supported
4. Inference: Apache Flink & Redis as backends

Taking off
Event example

```json
{
  "event": "metadata",
  "id": "81f46c34-a4bb-469c-8708-f8127cd67d27",
  "item": "product1",
  "timestamp": "1599391467000",
  "fields": [
    {"name": "title", "value": "Nice jeans"},
    {"name": "price", "value": 25.0},
    {"name": "color", "value": ["blue", "black"]},
    {"name": "availability", "value": true}
  ]
}
```

- **Metadata**: what prior data we have?
- **Impression**: what was displayed to visitor?
- **Interaction**: which actions were performed?
No-code YAML feature setup

Goal: cover 90% most common ML features

- **feature extractors**: compute ML feature value
- **feature store**: add to changelog if changed
- **online serving**: cache latest value for inference
Feature extractors: basic

// take a value from metadata
- name: vote_avg
type: number
scope: item
source: metadata.vote_avg
ttl: 60 days
Feature extractors: basic

// one-hot encode a string
- name: genre
  type: string
  scope: item
  source: metadata.genres
  values:
  - drama
  - comedy
  - thriller
Transformations

// length of the title field
- name: title_length
  type: word_count
  source: metadata.title
  scope: item
Special transformations

// one-hot encode mobile/desktop/tablet category
// from User-Agent field

- name: platform
type: ua_platform
source: impression.ua
Counters

// count how many clicks were done in current session

- name: click_count
type: interaction_count
scope: session
interaction: click
// A sliding window count of interaction events
// for a particular item

- name: item_click_count
  type: window_count
  interaction: click
  bucket_size: 24h // make a counter for each 24h rolling window
  windows: [7, 14, 30, 60] // on each refresh, aggregate to 1-2-4-8 week counts
  refresh: 1h
Profiling

// Does this user had an interaction before
// with other item with the same field value?

- name: clicked_color
type: interacted_with
interaction: click
field: metadata.color
scope: user
Rates: CTR & Conversion

// Click-through rate
- name: CTR
  type: rate
  top: click  // divide number of clicks
  bottom: examine  // to number of examine events
  scope: item
  bucket: 24h  // aggregate over 24-hour buckets
  periods: [7, 14, 30, 60]  // sum buckets for multiple time ranges
Normalization

// histogram sampled number normalization for price
- name: price
  type: relative_number
  method:
    type: estimate_histogram
    pool_size: 100  // for a pool size of 100
    sample_rate: 10  // we sample every 10th event in the pool
    bucket_count: 5  // so value will be mapped to 0-20-40-60-80-100 percentiles
  field: price
  source: item
Current status

https://demo.metarank.ai

- MVP, not all feature extractors are implemented
- Distributed mode is broken
- A long backlog of ML tasks: click models, LTR, de-biasing
Future

We built Metarank to solve our problem.

But it can be useful for others!

- Describe your use-case
- Report problems
Metarank

- github.com/metarank/metarank
- metarank.slack.com
- linkedin.com/in/romangrebennikov/
  linkedin.com/in/vgoloviznin/