MLOps World 2022

Jim Dowling, CEO, Hopsworks

Personalized Recommendations and Search with **Retrieval and Ranking** at scale on Hopsworks

HOPSWORKS

H&M Personalized Fashion Recommendations

Personalized product recommendations based on previous purchases

- **articles.csv** - detailed metadata for each `article_id` available for purchase
- **customers.csv** - metadata for each `customer_id`
- **transactions_train.csv** - purchases for each customer with date

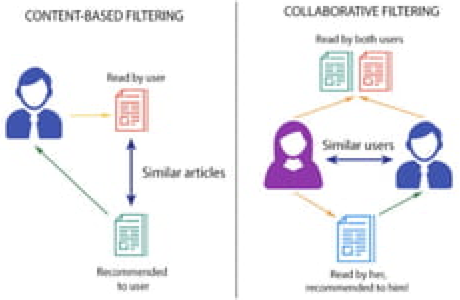
Predict the `article_ids` each customer will purchase.

https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations

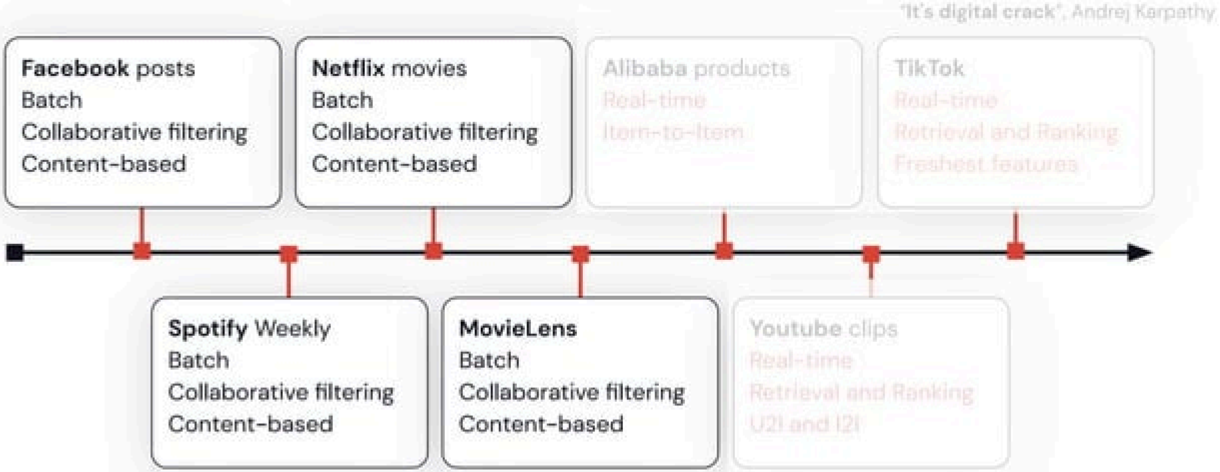
Recommender systems are ***information filtering systems*** that personalize the information coming to a user based on her historical interests, the relevance of the information, and the current context (e.g., what is trending).

Some paradigms for recommender systems:

- Collaborative filtering
- Content-based filtering
- Social and demographic recommenders
- Contextual recommendation algorithms

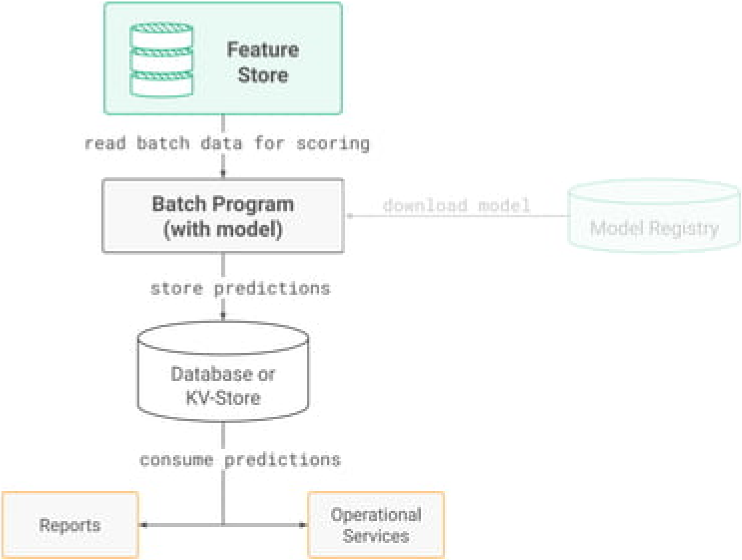


Analytical/Batch Recommendation Systems



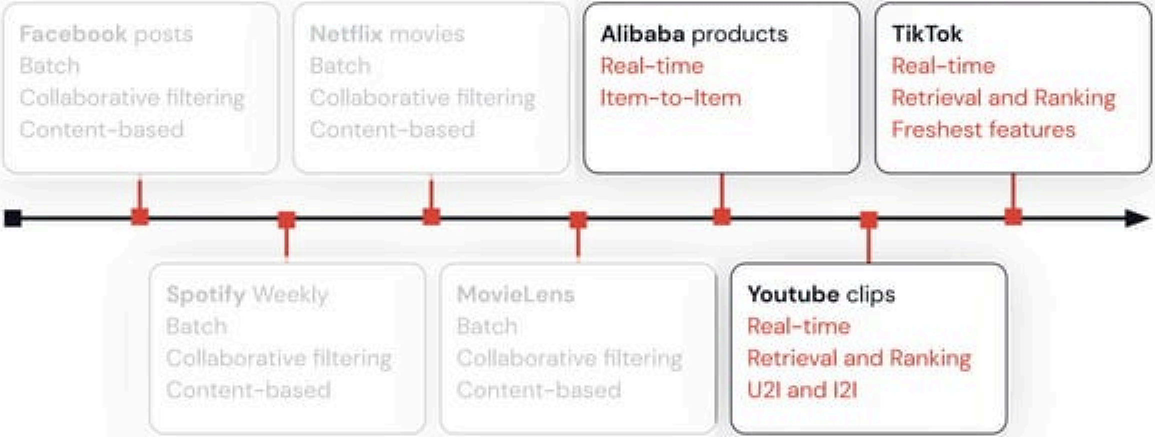
HOPSWORKS

Analytical (Batch) Recommendation Service

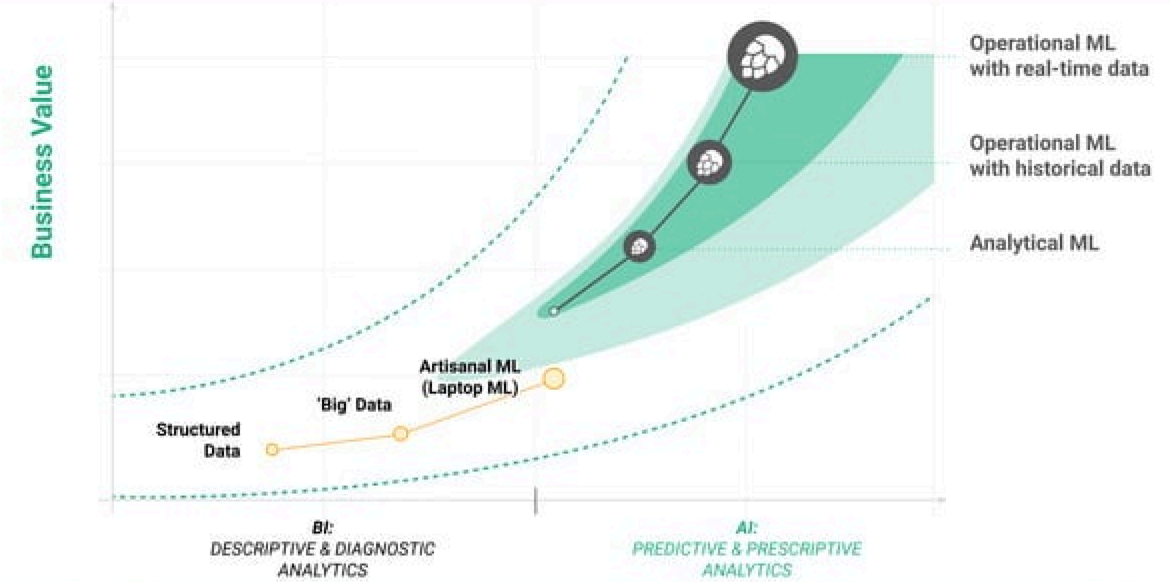


HOPSWORKS *Operational/Online Recommendation Systems*

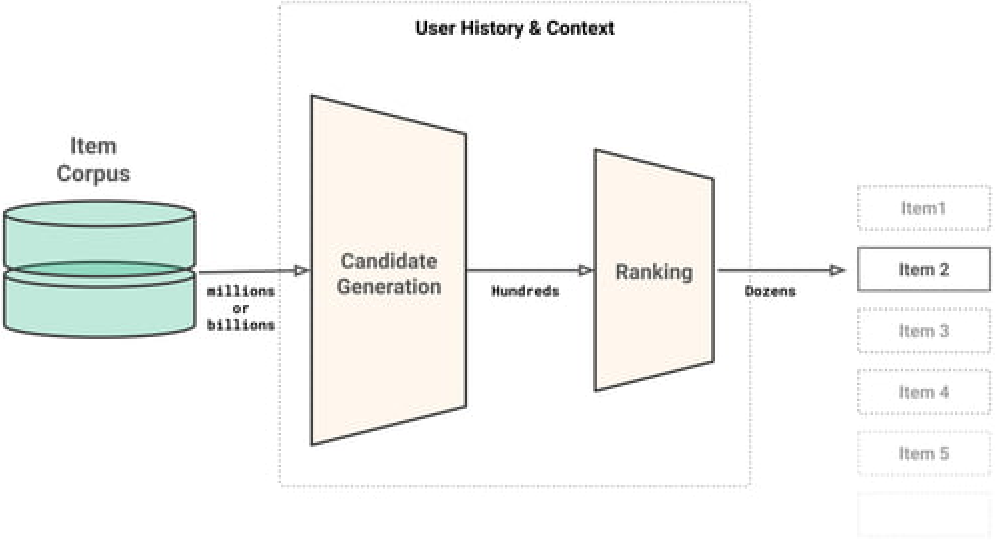
"**It's digital crack**", Andrej Karpathy



HOPSWORKS Increase Business Value with more Real-Time Recommendations



Online Recommendations: Retrieval and Ranking Architecture



3-Stage Retrieval, Filtering, Ranking

Personalized Music Discovery Playlist:

- *Retrieval:* Nearest neighbor search in item embedding space
- *Filtering*: Remove tracks user has heard before (e.g. w/ Bloom Filters)
- *Ranking*: Re-use embedding space distance from and tade off between score, similarity and BPm to reduce jarring track transitions

Social Media Feed:

- *Retrieval:* Random walks on social graph to find new items in user's network
- *Filtering*: Remove posts from muted or blocked users
- *Ranking*: Predict user's likelihood of interacting with posts, but "twiddle" the list so adjacent posts are from different authors

Examples by Nvidia - https://www.youtube.com/watch?v=5qjiY-kLwFY

3-Stage Retrieval, Filtering, Ranking

eCommerce "Add to Cart":

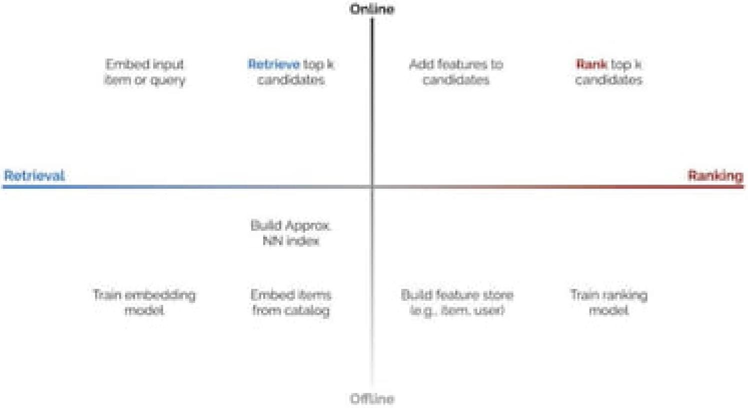
- *Retrieval:* Look up items commonly co-purchased with cart contents
- *Filtering*: Remove candidate items that are currently out of stock (or already in your cart)
- *Ranking*: Predict how likely to buy each candidate item the user is, but reorder to maximize expected revenue

Social Media Feed:

- *Retrieval:* Many candidate sources for the various rows/shelves/banners
- *Filtering*: Remove items that aren't licensed for streaming in user's country
- *Ranking*: Predict user's stream time for each item, but arrange a set of shelves that trade off between predicted relevance and matching the genre distribution of the user's previous consumption

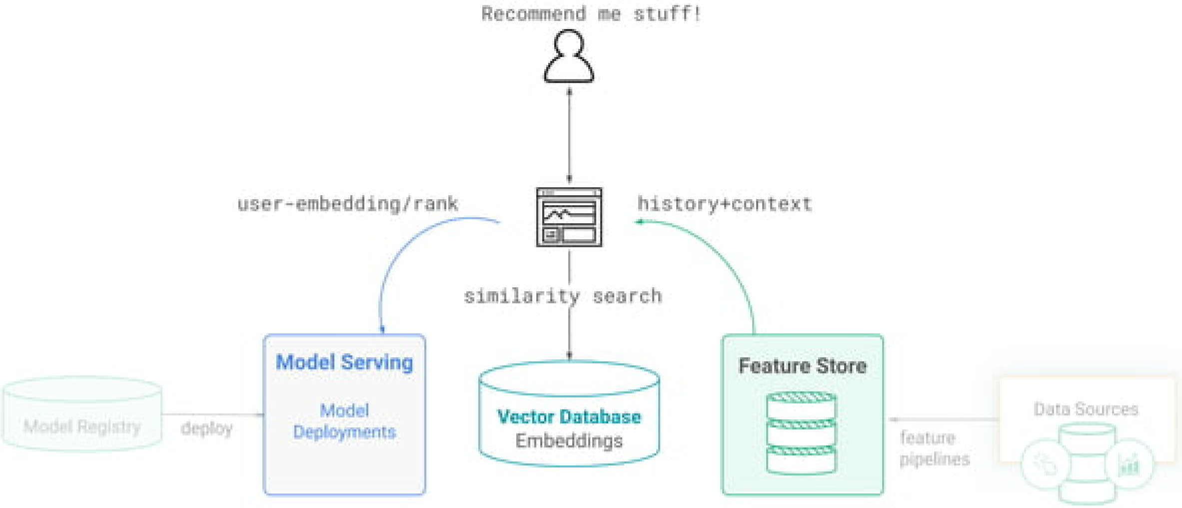
Examples by Nvidia - https://www.youtube.com/watch?v=5qjiY-kLwFY

Online and Offline Infrastructure for Retrieval/Ranking



[Image credit: https://eugeneyan.com/writing/system-design-for-discovery/]

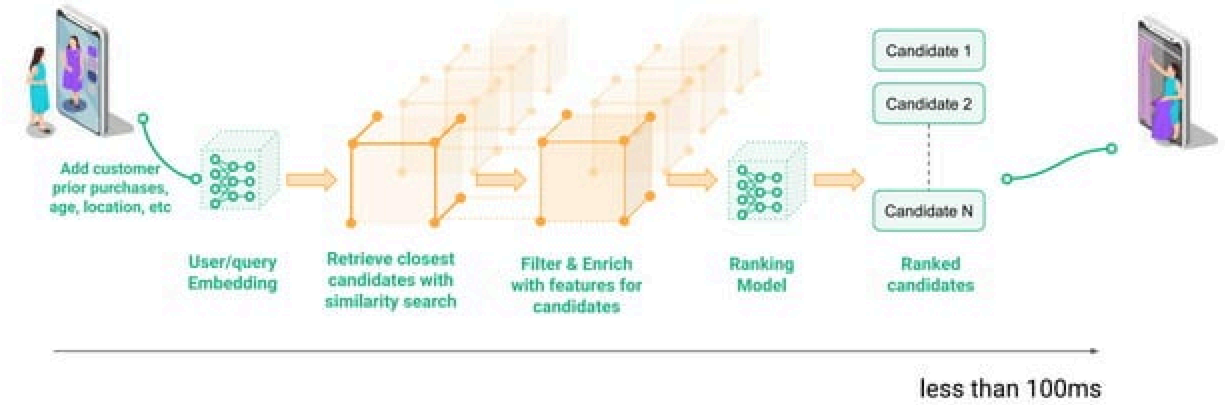
(Two-Tower) Operational Recommendation Service



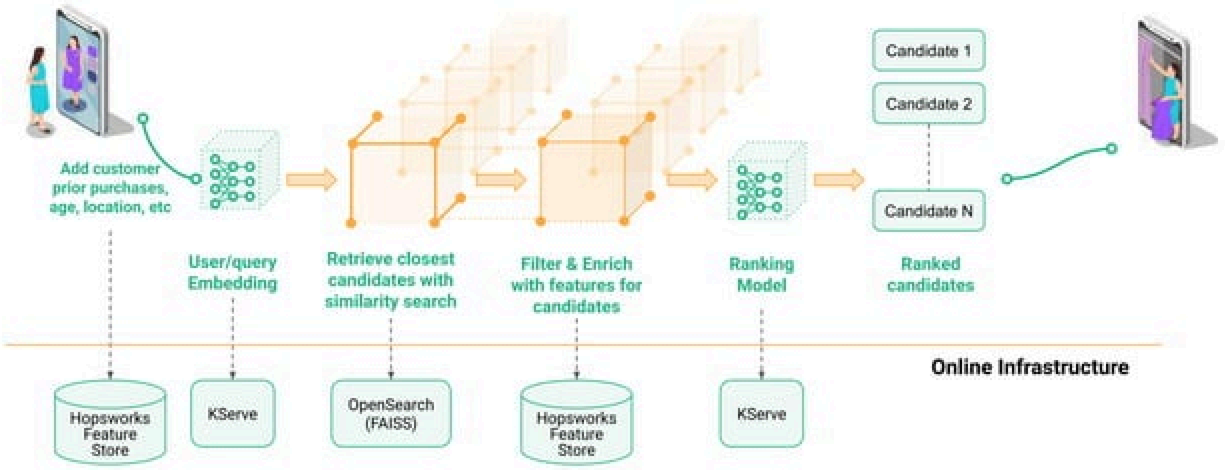
Create Embeddings - then Retrieval, Filtering, Ranking

User/Query and Item Embeddings →	Retrieval →	Filtering →	Ranking
User/query embedding model - user history and context. Item embedding model Built Approx Nearest Neighbor (ANN) Index with items and item embedding model.	Retrieve candidate items based on the user embedding from the ANN Index	Remove candidate items for various reasons: • underage user • item sold out • item bought before • item not available in user's region	With the ranking model, score all the candidate items with both user and item features, ensuring, candidate diversity.

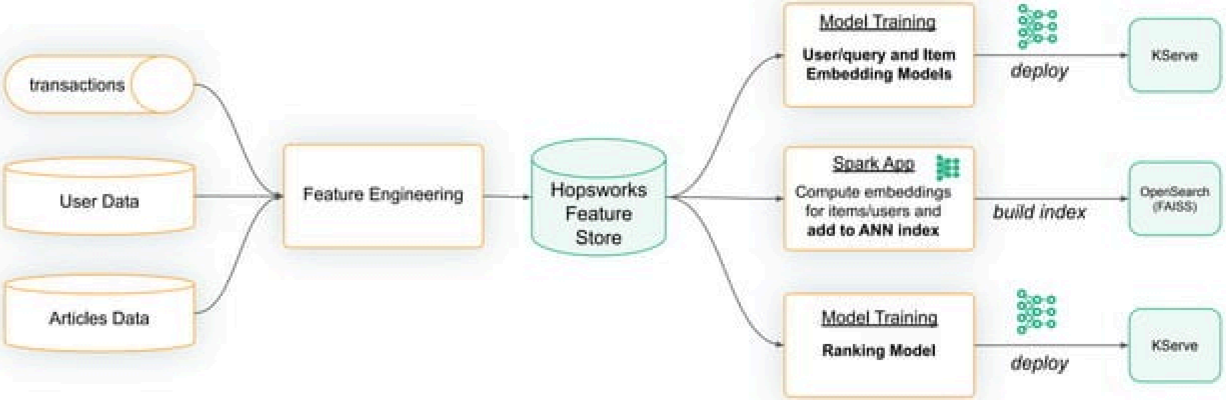
Retrieval and Ranking for Fashion Recommendations



Online Infrastructure for Retrieval and Ranking



Offline Infrastructure for Ranking and Retrieval



01.
Embeddings

HOPSWORKS *Embeddings*

- An embedding is a mapping of a discrete — categorical — variable to a vector of continuous numbers: [1.119, 4.341, ...,1.334]
- Create a denser representation of the categories and maintain some of the implicit relationship information between items

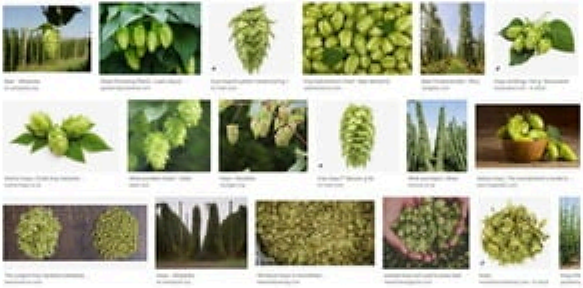
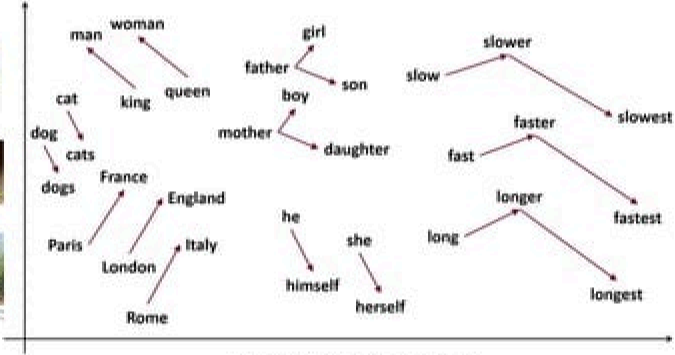


Image Embeddings enable Similarity Search



Word2Vec Embeddings

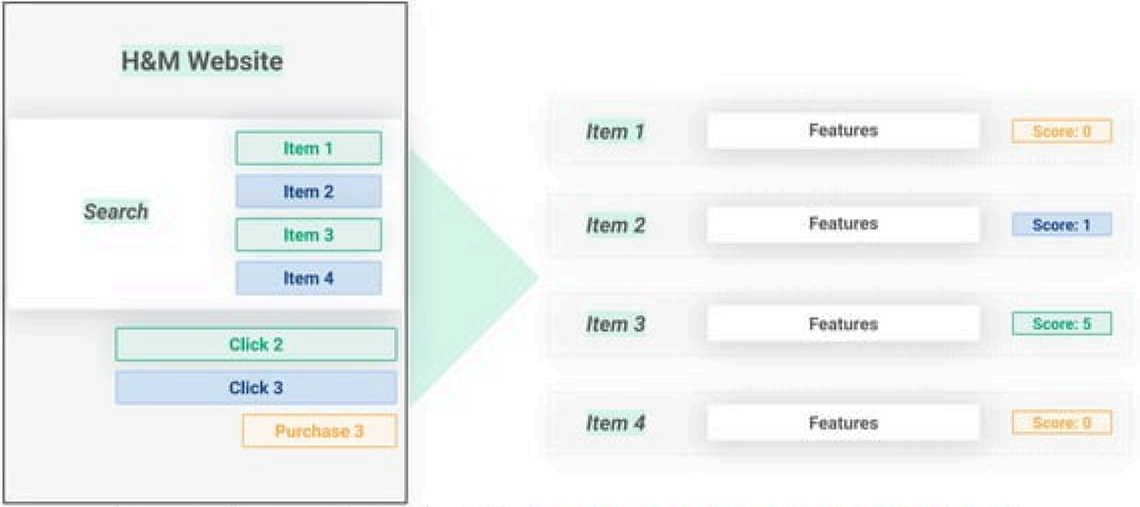
Can a "`user query`" find "`items`" with similarity search?

Yes, by mapping the "`user query`" embedding into the "`item`" embedding space.

Representation learning for retrieval usually involves supervised learning with labeled or pseudo-labeled data from user-item interactions.

Feature Logging - Generate Training Data for Embedding Models

Log user-item interactions as training data for our two-tower model.

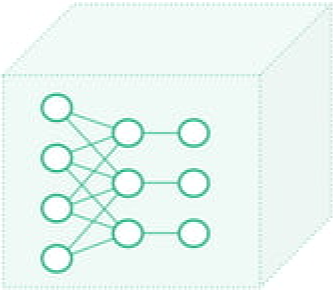


[Image Credit: Roman Grebennikov, Findify - https://www.youtube.com/watch?v=BskjQPkrYec]

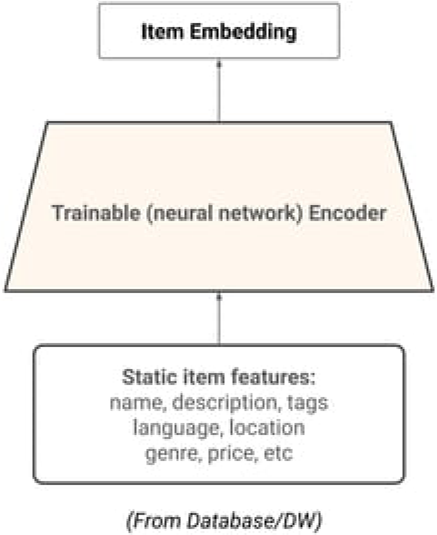
How do we create Training Data for our Embedding Models?

Implicit Feedback vs Explicit Feedback
Imagine the user sees a feed and selects the fifth item in the list. This shows us that the first four items should have been ranked lower in the list. We can train our ranker model with this sort of data.
You can also train a model with explicit feedback by introducing a user feedback/rating system in your application.

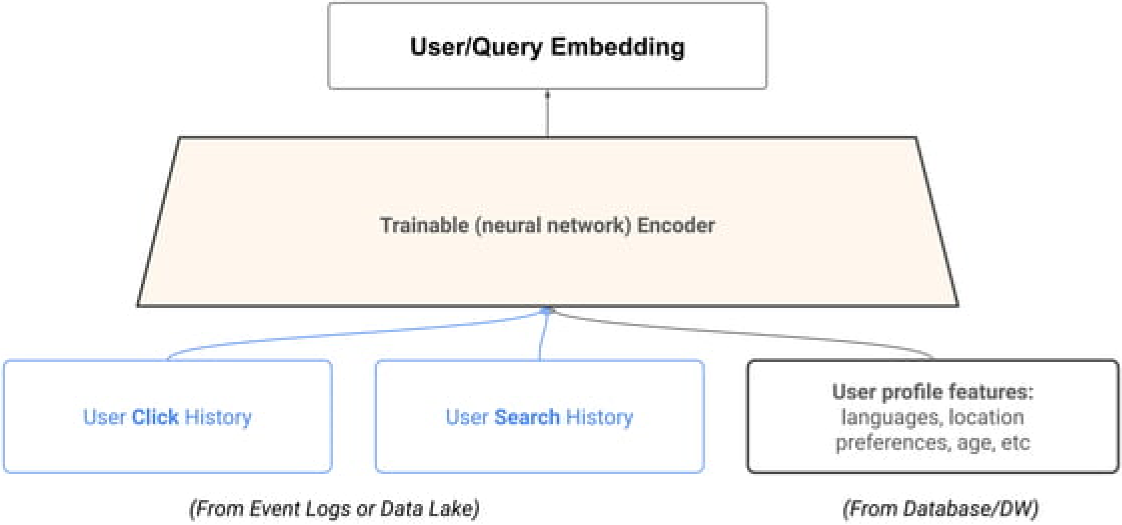
Negative examples should also be used when training - items labeled "irrelevant" to a given query.



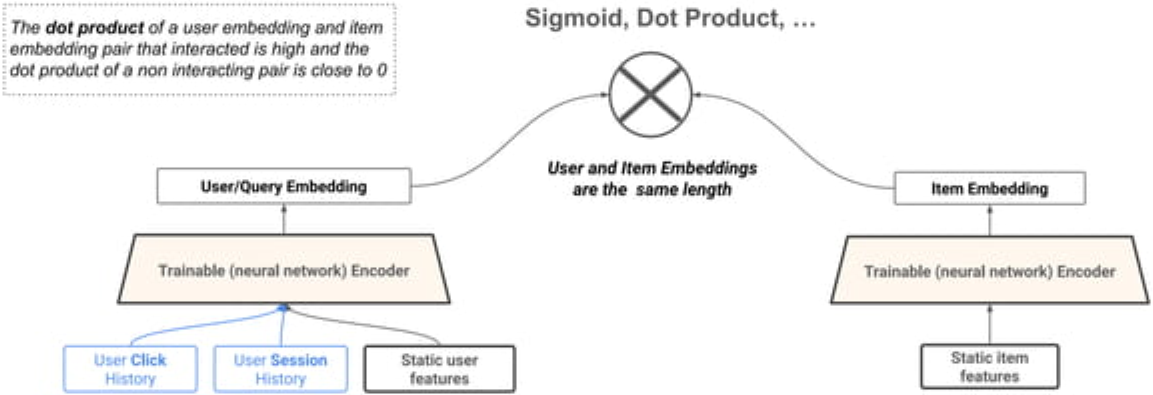
The Item Embedding Model



The User/Query Embedding Model



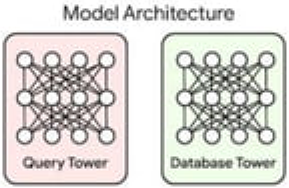
Training the Query and Item Embedding Models with Two Towers



TensorFlow has the tensorflow-recommenders library to train two-tower embedding models.
Our training data, transactions.csv, consists of customer and article pairs. You need to provide only positive pairs, where the customer purchased an article. Training produces 2 models: an item encoder model and a user encoder model.

Create embeddings for Items, insert them into embedding space

(aka build an Approximate Nearest Neighbor (ANN) Index)



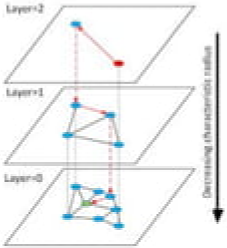
[Source: https://cloud.google.com/blog/products/ai-machine-learning/vertex-matching-engine-blazing-fast-and-massively-scalable-nearest-neighbor-search]

02.
Vector Database - Similarity Search with Embeddings

A vector database (or embedding store): a database for durably storing embeddings and supporting similarity search (nearest neighbor search)

Approximate nearest neighbor (ANN) algorithms provide fast nearest neighbor search, **O(log(n))**

- Facebook's FAISS uses Hierarchical Navigable Small World Graphs (HNSW)
- Google ScaNN



HNSW Graph Search

HOPSWORKS OpenSearch k-NN: Create Index

```
PUT my-knn-index-1
{
  "settings": {
    "index": {
      "knn": true,
      "knn.space_type": "cosinesimil"
    }
  },
  "mappings": {
    "properties": {
      "my_vector1": {
        "type": "knn_vector",
        "dimension": 2
      },
      "my_vector2": {
        "type": "knn_vector",
        "dimension": 4 }
}}}
```

Create a k-NN nearest neighbour Index in OpenSearch. In the following slides, we will add entries to the index. Then query it with approximate nearest neighbor search. Then query it with additional filters.

https://opensearch.org/docs/latest/search-plugins/knn

HOPSWORKS

OpenSearch k-NN: Insert Embeddings into Index

```
POST _bulk
{ "index": { "_index": "my-knn-index-1", "_id": "1" } }
{ "my_vector1": [1.5, 2.5], "price": 12.2 }
{ "index": { "_index": "my-knn-index-1", "_id": "2" } }
{ "my_vector1": [2.5, 3.5], "price": 7.1 }
{ "index": { "_index": "my-knn-index-1", "_id": "3" } }
{ "my_vector1": [3.5, 4.5], "price": 12.9 }
{ "index": { "_index": "my-knn-index-1", "_id": "4" } }
{ "my_vector1": [5.5, 6.5], "price": 1.2 }
{ "index": { "_index": "my-knn-index-1", "_id": "5" } }
{ "my_vector1": [4.5, 5.5], "price": 3.7 }
{ "index": { "_index": "my-knn-index-1", "_id": "6" } }
{ "my_vector2": [1.5, 5.5, 4.5, 6.4], "price": 10.3 }
{ "index": { "_index": "my-knn-index-1", "_id": "7" } }
{ "my_vector2": [2.5, 3.5, 5.6, 6.7], "price": 5.5 }
{ "index": { "_index": "my-knn-index-1", "_id": "8" } }
{ "my_vector2": [4.5, 5.5, 6.7, 3.7], "price": 4.4 }
{ "index": { "_index": "my-knn-index-1", "_id": "9" } }
{ "my_vector2": [1.5, 5.5, 4.5, 6.4], "price": 8.9 }
```

Note that each entry has a unique "_id" and an additional "price" attribute.
my_vector1 has an embedding of length 2.
my_vector2 has an embedding of length 4.

OpenSearch k-NN: Similarity search using Index

```
GET my-knn-index-1/_search
{
  "size": 2,
  "query": {
    "knn": {
      "my_vector2": {
        "vector": [2, 3, 5, 6],
        "k": 2
      }
    }
  }
}
```

`k` is the number of neighbors the search of each graph will return. The `size` option indicates how many results the query actually returns.

OpenSearch k-NN: Similarity search using Index with filtering

```
GET my-knn-index-1/_search
{
  "size": 2,
  "query": {
    "knn": {
      "my_vector2": {
        "vector": [2, 3, 5, 6],
        "k": 2
      }
    }
  },
  "post_filter": {
    "range": {
      "price": {
        "gte": 5,
        "lte": 10
      }
    }
  }
}
```

We are filtering results so that we only get candidates with a price between 5-10.

03.

Ranking and Refining Recommenations

What does a Ranking Model do?

Input: a set of instances

$$X = \{x_1, x_2, \ldots, x_n\}$$

Output: a rank list of these instances

$$\hat{Y} = \{x_{r_1}, x_{r_2}, \ldots, x_{r_n}\}$$

Ground truth: a correct ranking of these instances

$$Y = \{x_{y_1}, x_{y_2}, \ldots, x_{y_n}\}$$

Each instance (user-item pair) is represented with a list of features, retrieved from the feature store.

Training data is the user-item features and the label is the relevance ratings.

Ranking models should be fast - low latency to rank 100s of candidates, so decision trees are popular.

Ranking Models - reordering and the feature store

The ranking stage should consider additional criteria or constraints.

If the system always recommend items that are "closest" to the user/query embedding, the candidates tend to be very similar to each other.

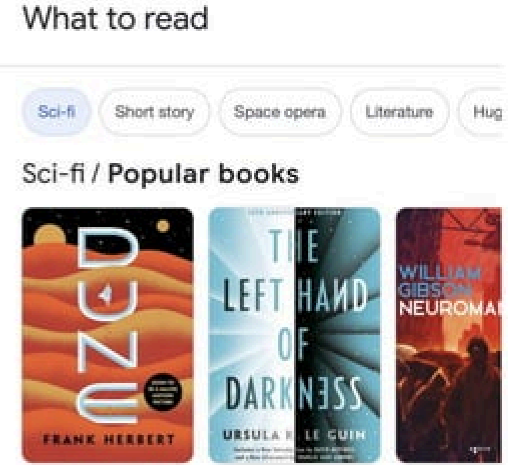
Re-rank items based on genre or other metadata to ensure diversity.

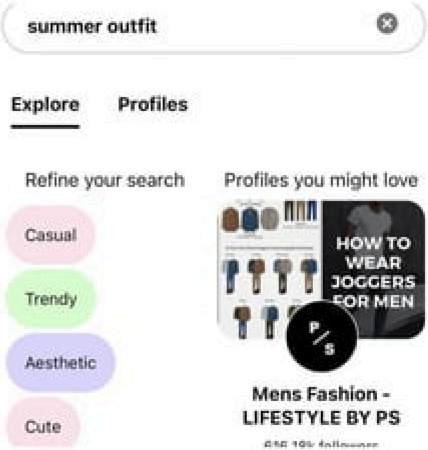
When ranking, we can include features that were not feasible during candidate generation.

Use the feature store to retrieve features such as user persona (e.g., demographics, price propensity), item metadata (e.g., attributes, engagement statistics), cross features (e.g., interaction between each feature pair), and media embeddings.

Building a ranking model with TF Recommenders: https://www.youtube.com/watch?v=ZkwJo5HRjiQ
https://www.tensorflow.org/recommenders/examples/basic_ranking

Tags and Metadata - Refine your Recommendations/Search

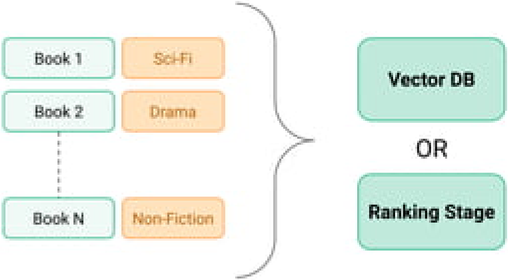




Tags, Metadata - Refining your Search by Filtering

Book 1	Feature1	Feature2	...	TAG

Feature helps generate candidates in ANN search



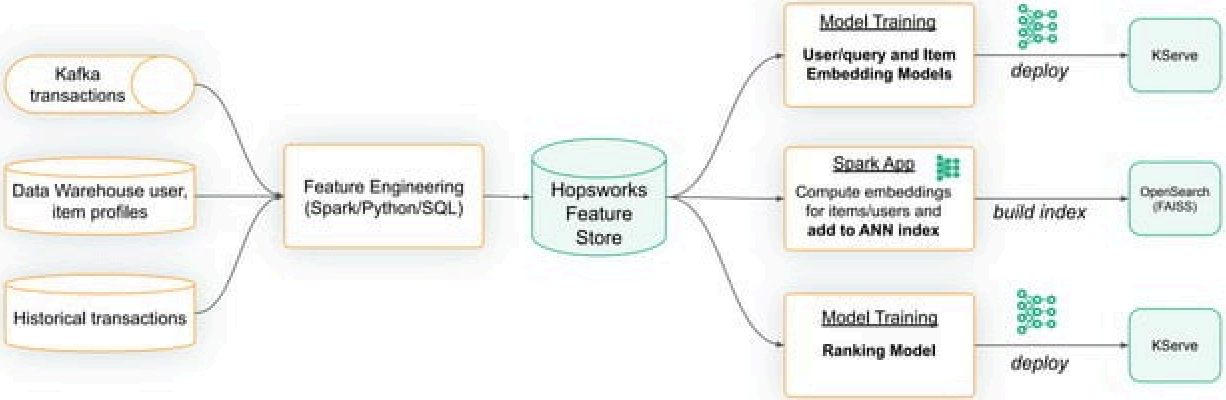
Static attributes. Filter ANN results in OpenSearch

Static/Dynamic Filter Candidates.

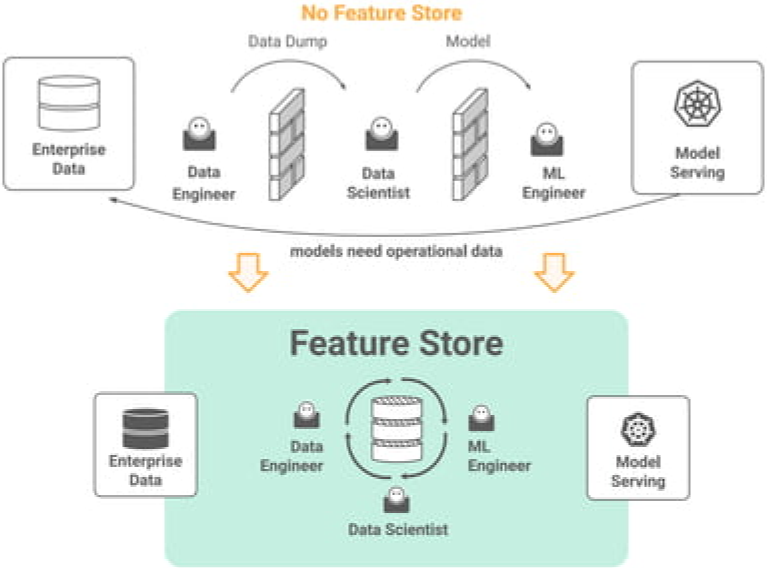
04.

Offline Infrastructure for Ranking and Retrieval

HOPSWORKS *Offline Infrastructure for Ranking and Retrieval*

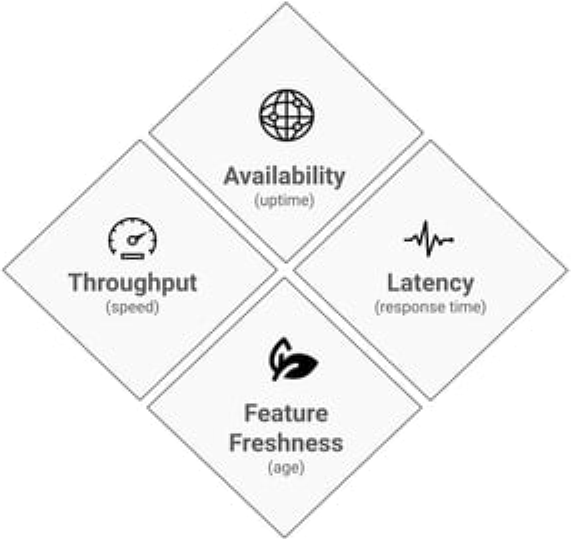


HOPSWORKS *What is a Feature Store for Machine Learning?*



What are the System Properties of Recommendation Systems?

Apart from the accuracy of your ML models, you have to worry about…



HOPSWORKS

Why should I care about Performance and Availability ?

Availability: Embedded in-memory DB crashes on Black Friday at e-retailer, produces random recommendations

Throughput: Online retail store produces increasing number of sales, needs to switch feature pipeline from Pandas to PySpark.

Freshness: Video streaming service switches from Spark Streaming to Flink, improving feature freshness from 10s down to 2s, improving the engagement rate.

Latency: A real-time recommendation service for a music streaming company returns 250 candidates during retrieval, then it needs 250 lookups from the feature store in < 30ms for ranking.

Why should I care about Performance and Availability ?

Availability: Embedded in-memory DB crashes on Black Friday at e-retailer, produces random recommendations.

Throughput: Online retail store produces increasing number of sales, needs to switch feature pipeline from Pandas to PySpark.

Freshness: Video streaming service switches from Spark Streaming to Flink, improving feature freshness from 10s down to 2s, improving the engagement rate.

Latency: A real-time recommendation service for a music streaming company returns 250 candidates during retrieval, then it needs 250 lookups from the feature store in < 30ms for ranking.

HOPSWORKS Why should I care about Performance and Availability ?

Availability: Embedded in-memory DB crashes on Black Friday at e-retailer, produces random recommendations.

Throughput: Online retail store produces increasing number of sales, needs to switch feature pipeline from Pandas to PySpark.

Freshness: Video streaming service switches from Spark Streaming to Flink, improving feature freshness from 10s down to 2s, improving the engagement rate.

Latency: A real-time recommendation service for a music streaming company returns 250 candidates during retrieval, then it needs 250 lookups from the feature store in < 30ms for ranking.

HOPSWORKS Why should I care about Performance and Availability ?

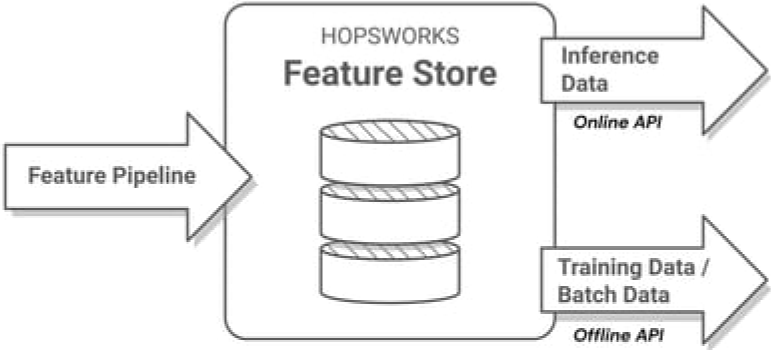
Availability: Embedded in-memory DB crashes on Black Friday at e-retailer, produces random recommendations.

Throughput: Online retail store produces increasing number of sales, needs to switch feature pipeline from Pandas to PySpark.

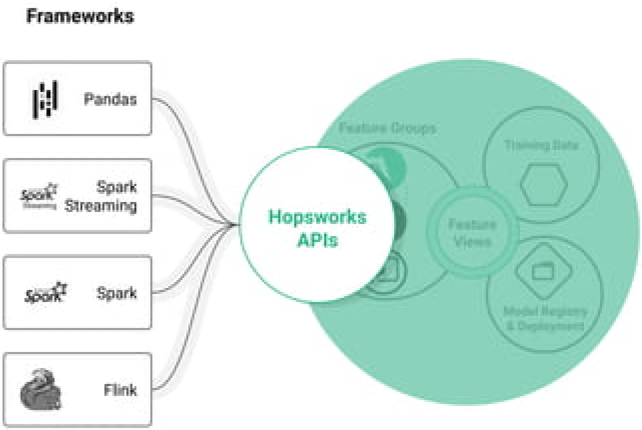
Freshness: Video streaming service switches from Spark Streaming to Flink, improving feature freshness from 10s down to 2s, improving the engagement rate.

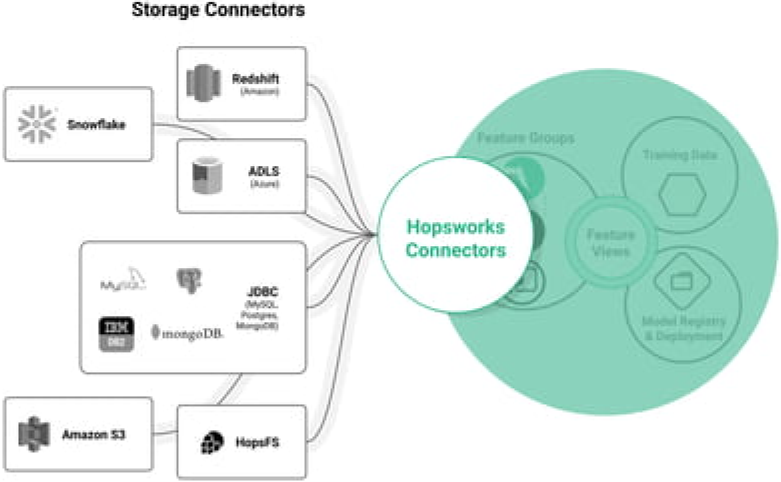
Latency: A real-time recommendation service for a music streaming company returns 250 candidates during retrieval, then it needs 250 lookups from the feature store in < 30ms for ranking.

HOPSWORKS *Write API, Online/Offline Read APIs*

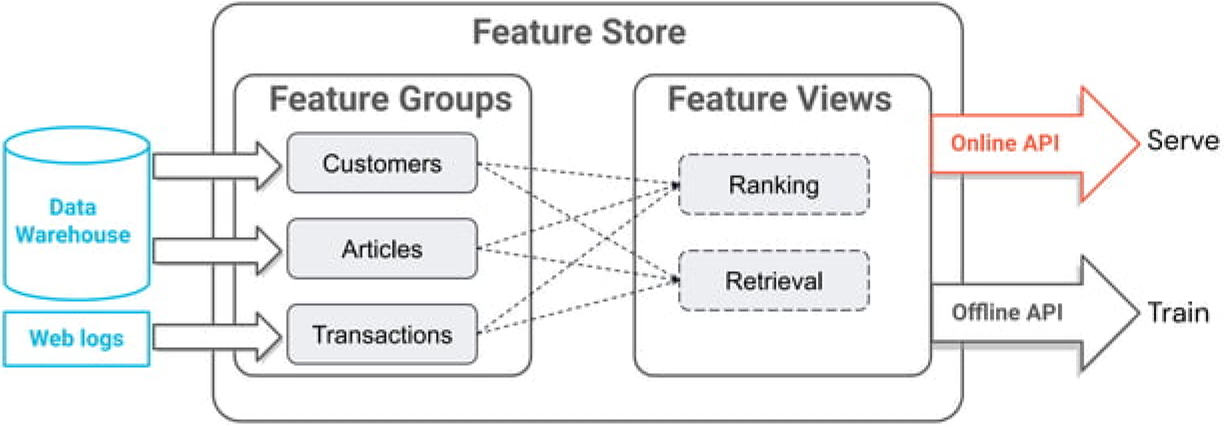


HOPSWORKS Feature Engineering: APIs & Connectors

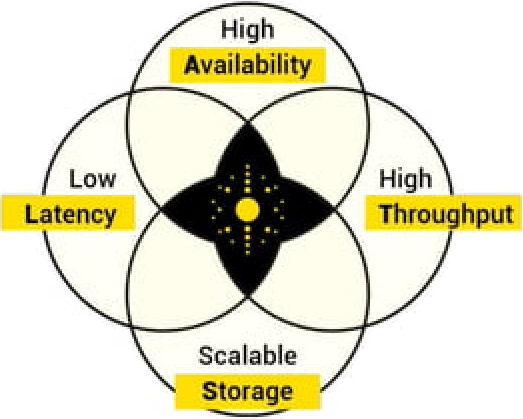



Storage Connectors
Redshift
(Amazon)
Snowflake
ADLS
(Azure)
JDBC
(MySQL, Postgres, MongoDB)
mongoDB
Amazon S3
HopsFS
Hopsworks Connectors
Feature Groups
Training Data
Feature Views
Model Registry & Deployment

Write to Feature Groups, Read from Feature Views



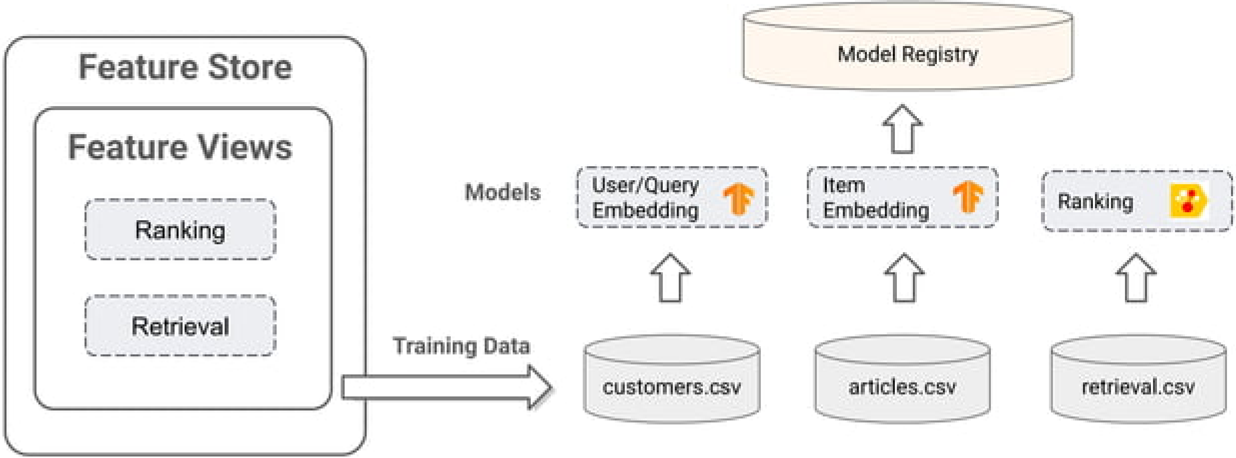
RonDB - Hopsworks' Online Feature Store



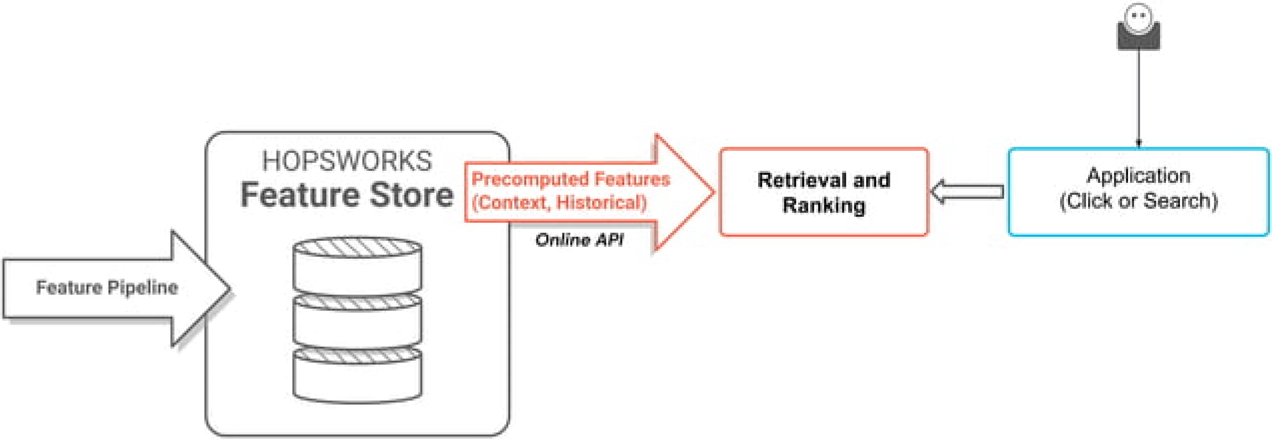
https://www.rondb.com

< 1ms KV lookup
>100M KV Lookups/sec
>99.999% availability
>1 PB storage

HOPSWORKS *Training Models*



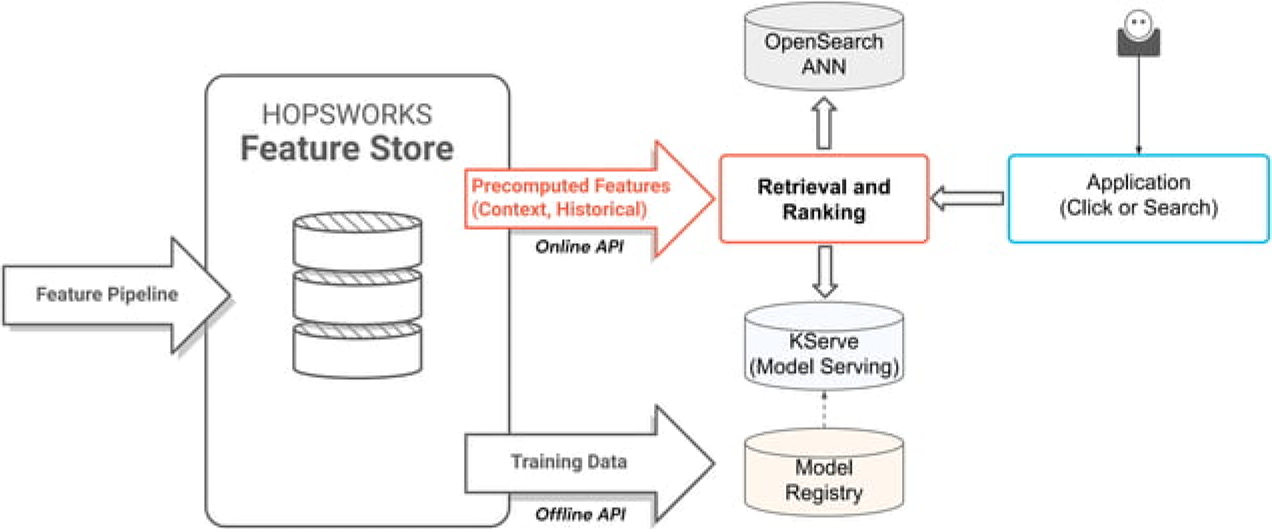
Feature Store used for Online for Retrieval and Ranking



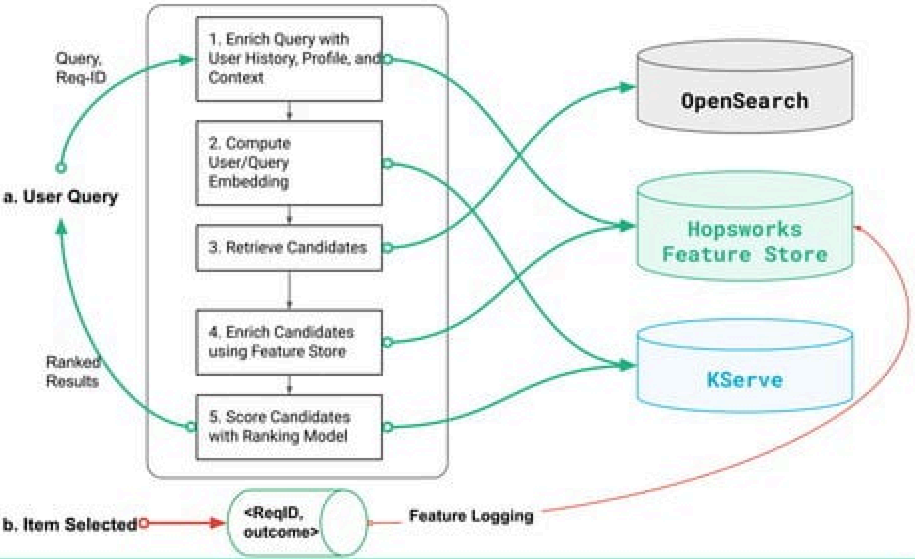
05.

Hopsworks = Feature Store + VectorDB + Model Serving

Hopsworks for Retrieval and Ranking



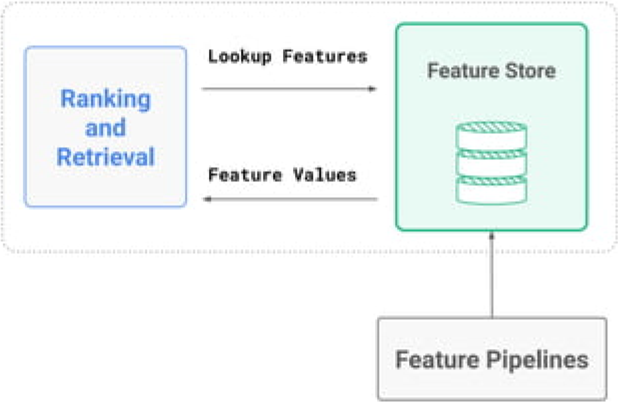
Hopsworks Ranking and Retrieval Service



06.
Case Study:
Spotify Music Search

Goal:
Support Spotify Personalized Search in a Retrieval and Ranking Architecture.
Benchmark the highest throughput, lowest latency key-value stores to identify one that could scale to handle millions of concurrent lookups per second on Spotify's workloads.

Systems:
Aerospike and **RonDB** were identified as the only systems capable of meeting the triple goals of High Throughput, Low Latency, and High Availability. Other databases such as Redis, Cassandra, BigTable were not considered for availability or latency or throughput reasons.

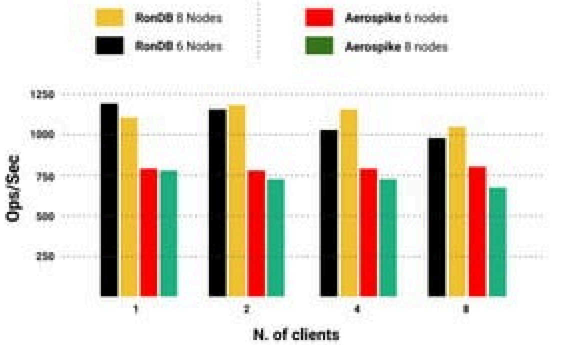


Benchmark Setup

Node Type	GCP Instance Type	Virtual CPUs	Memory	Disk Size	Disk Type
MySQL Servers	n1-standard-2	2	7.5GB	120GB	pd-ssd
NDB Management Node	e2-standard-16	16	64GB	120GB	pd-ssd
NDB Data Nodes	n1-highmem-32	32	208GB	408GB	pd-ssd
Aerospike Nodes	n1-highmem-32	32	208GB	408GB	pd-ssd
Java Client Nodes	e2-standard-16	16	64GB	120GB	pd-ssd

Hardware Benchmark Setup on GCP: RonDB (NDB) vs Aerospike. The Java Client nodes are the clients performing the reads/writes on the Data Nodes. When the cluster is provisioned with 8 RonDB (NDB) data nodes, it has 832GB of usable in-memory storage, when a replication factor of 2 is used.

Benchmark Results - Throughput

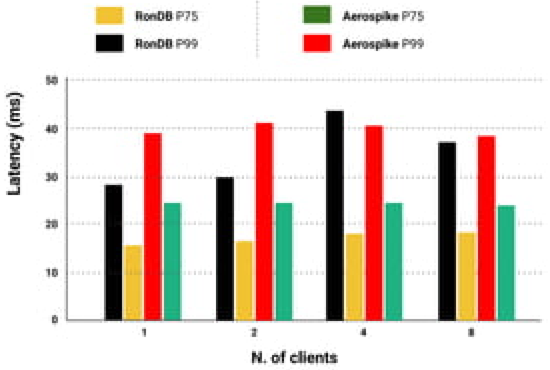


Average throughput of the clients in both 6 and 8 node setup of RonDB and Aerospike

Throughput: higher is better. Each feature store operation was a batch of 250 key-value lookups, meaning with 8 clients for a 8-node RonDB cluster, there are >2m lookups/sec.

HOPSWORKS

Benchmark Results - Latency



Average latency of the clients in both 6 node setup of RonDB and Aerospike

Latency: lower is better. Each feature store operation was a batch of 250 key-value lookups. So, for RonDB, the P99 when performing 250 primary operations in a single transaction is under 30ms.

RonDB **35% Higher Throughput**
RonDB **30% Better Latency**

Based on Public Report from Spotify
comparing Aerospike and RonDB (NDB Cluster) as Feature Stores

http://kth.diva-portal.org/smash/get/diva2:1556387/FULLTEXT01.pdf

07.

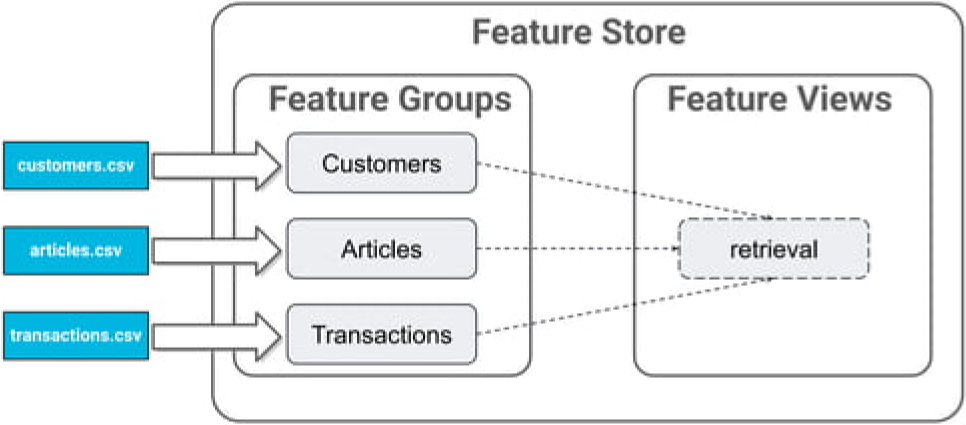
Ranking and Retrieval with Hopsworks

H&M Recommendation Service

Today's Workshop: Retrieval and Ranking Platform in Python

- articles.csv
- customers.csv
- transactions_train.csv

1_feature_engineering.ipynb	create feature groups for articles, customers, transactions
2a_create_retrieval_dataset.ipynb	create feature view for retrieval model (training data)
2b_train_retrieval_model.ipynb	train two-tower model - user and article embedding models
3_build_index.ipynb	build opensearch KNN index with embeddings for all articles
4a_create_ranking_dataset.ipynb	create feature view for retrieval model (training data)
4b_train_ranking_model.ipynb	train ranking model
5_create_deployment.ipynb	deploy models to KServe + glue code for Hopsworks, OpenSearch

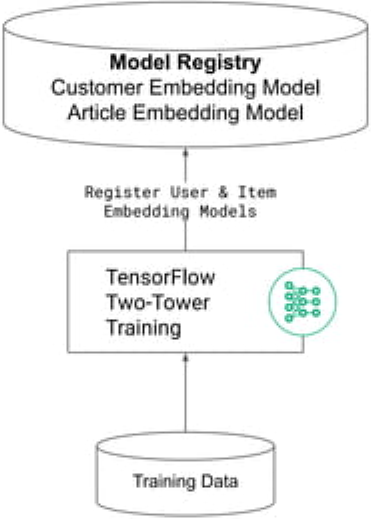


Transaction = (customer_id, article_id, timestamp, channel)

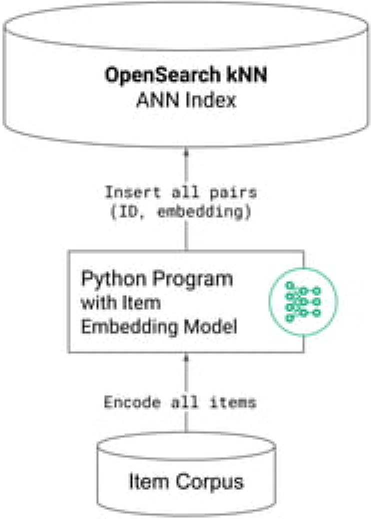
1_feature_engineering.ipynb

2a_create_retrieval_feature_views.ipynb

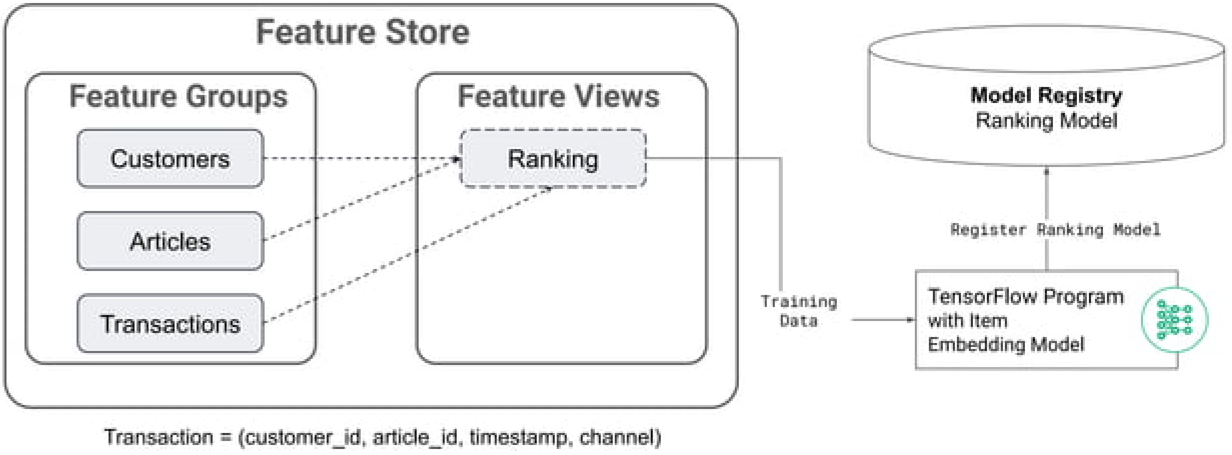
HOPSWORKS



2b_train_retrieval_model.ipynb

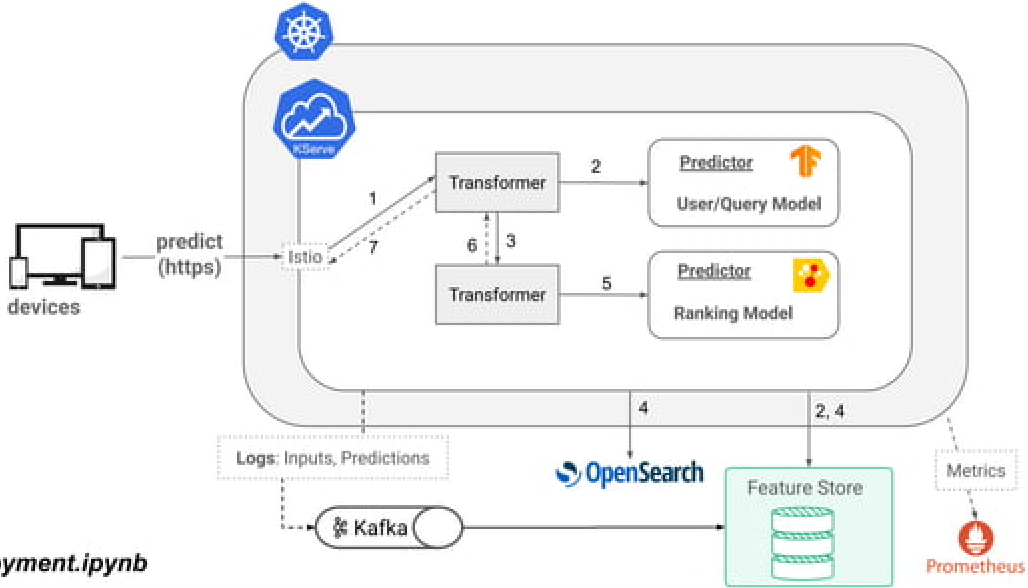


3_build_index.ipynb



4a_create_ranking_feature_views.ipynb

4b_train_ranking_model.ipynb



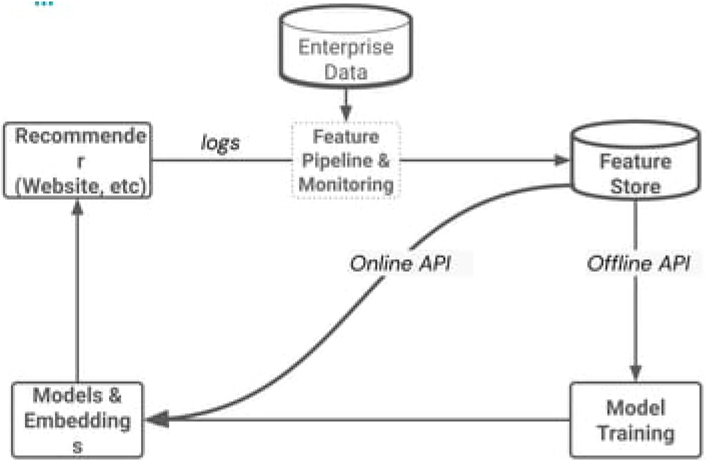
5_create_deployment.ipynb

08.

Where next for Retrieval and Ranking Architectures?

Build out your Data-for-AI Flywheel

more data -> better models -> more users -> more data-> ...



Democratize the development of Recommendation Systems

```
user
.let(seed_id=user_id)
.liked(max_num_to_retrieve=30)
.account_nn(embedding_config=default)
.posted_media(max_media_per_account=10)
.filter(non_recommendable_model_threshold=0.2)
.rank(ranking_model=default)
.diversify_by(seed_id, method=round_robin)
```

https://ai.facebook.com/blog/powered-by-ai-instagrams-explore-recommender-system/

HOPSWORKS

Hopsworks is the platform for **ML collaboration,**
powered by the leading
Python-centric Enterprise Feature Store.

Silicon Valley

470 Ramona St
Palo Alto
California
USA

⭐ Stockholm

Hopsworks AB
Medborgarplatsen 25
118 72 Stockholm
Sweden

London

IDEALondon,
69 Wilson St.
London
UK

HOPSWORKS

www.hopsworks.ai

Compliance
Governance

Efficiency
At Scale

Open &
modular