Back To Module Home

Machine Learning System Design

0% completed

Search Module

Machine Learning Primer

Video Recommendation

Feed Ranking

Ad Click Prediction

- Problem Statement and Metrics
- Ad Click Prediction Model

Ads Recommendation System Design

Rental Search Ranking

Mark Module as Completed

Ads Recommendation System Design

Learn about the Ads Recommendation system design.

We'll cover the following

4. Calculation and estimation#

Assumptions#

- 40K ad requests per second or 100 billion ad requests per month
- Each observation (record) has hundreds of features, and it takes 500 bytes to store.

Data size#

- Data: historical ad click data includes [user, ads, click_or_not]. With an estimated 1% CTR, it has 1 billion clicked ads. We can start with 1 month of data for training and validation. Within a month we have, $100 * 10^{12} * 500 = 5 * 10^{16}$ bytes or 50 PB. One way to make it more manageable is to downsample the data, i.e., keep only 1%-10% or use 1 week of data for training data and use the next day for validation data.

Scale#

- Supports 100 million users

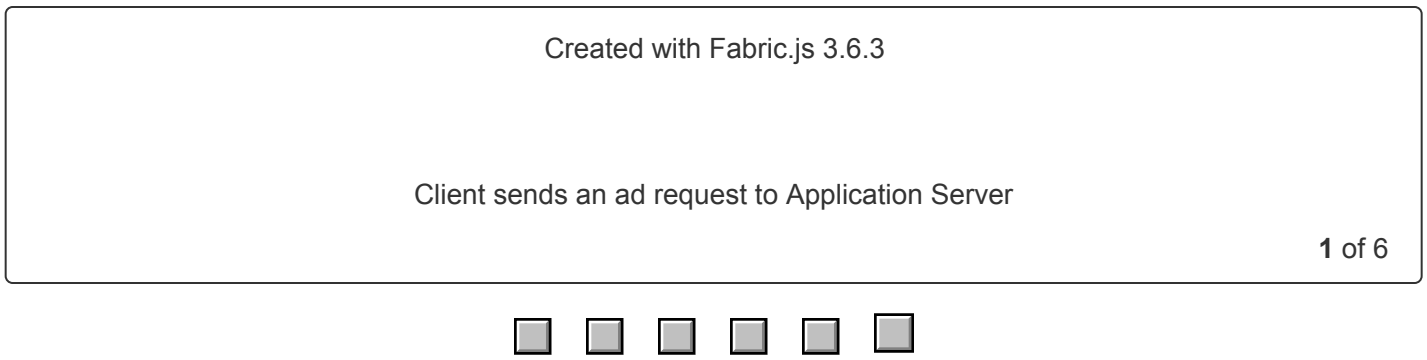
5. High level design#

- Data lake: Store data that is collected from multiple sources, i.e., logs data or event-driven data (Kafka)
- Batch data prep: Collections of ETL (Extract, Transform, and Load) jobs that store data in Training data Store.
- Batch training jobs organize scheduled jobs as well as on-demand jobs to retrain new models based on training data storage.
- Model Store: Distributed storage like S3 to store models.
- Ad Candidates: Set of Ad candidates provided by upstream services (refer back to waterfall model).
- Stream data prep pipeline: Processes online features and stores features in key-

value storage for low latency down-stream processing.

- Model Serving: Standalone service that loads different models and provides Ad Click probability.

Let's examine the flow of the system:



- User visits the homepage and sends an Ad request to the Candidate Generation Service. Candidate Generation Service generates a list of Ads Candidates and sends them to the Aggregator Service.
- The Aggregator Service splits the list of candidates and sends it to the Ad Ranking workers to score.
- Ad Ranking Service gets the latest model from Model Repos, gets the correct features from the Feature Store, produces ad scores, then returns the list of ads with scores to the Aggregator Service.
- The Aggregator Service selects top K ads (For example, k = 10, 100, etc.) and returns to upstream services.

6. Scale the design#

- Given a latency requirement of 50ms-100ms for a large volume of Ad Candidates (50k-100k), if we partition one serving instance per request we might not achieve

Service Level Agreement (SLA). For this, we scale out Model Serving and put Aggregator Service to spread the load for Model Serving components.

> One common pattern is to have the Aggregator Service. It distributes the candidate list to multiple serving instances and collects results. Read more about it here.

7. Follow up questions#

Question	Answer
How do we adapt to user behavior changing over time?	Retrain the model as frequently as possible. One example is to retrain the model every few hours with new data (collected from user clicked data).
How do we handle the Ad Ranking Model being under-explored?	We can introduce randomization in Ranking Service. For example, 2% of requests will get random candidates, and 98% will get sorted candidates from Ad Ranking Service.

8. Summary#

- We first learned to choose Normalize Entropy as the metric for the Ad Click Prediction Model.
- We learn how to apply the Aggregator Service to achieve low latency and overcome

imbalance workloads.

- To scale the system and reduce latency, we can use `kube-flow` so that Ad Generation services can directly communicate with Ad Ranking services.
- We can also learn more about how companies scale there design here.

Ad Click Prediction Model

Next

Problem Statement and Metrics

Mark as Completed

Report an Issue