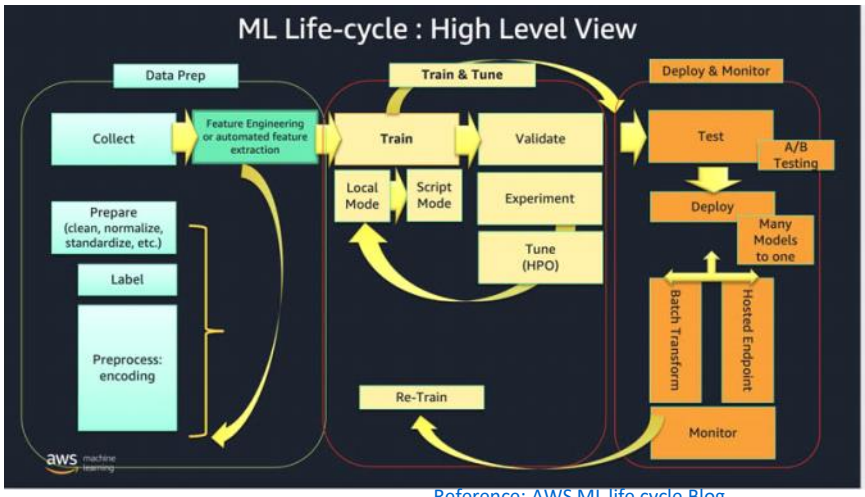
Recommender System: [github link]

predicts which items a user will like by learning patterns from their past behavior and similar users or items



Reference: AWS ML life cycle Blog

What all I have covered as part of ML life cycle:

- Data Acquisition
- Data Preparation
- ETL
- Model Selection
- Model Training & Evaluation
- Deployment (local APIs)
- Post deployment monitoring
- Created Pipelines for future model improvement (incremental training)

About data:

Amazon review 2023 by University of California San Diego

This is a massive dataset collected by McAuley Lab, containing **~571.5 million reviews** and metadata for **~48 million items** across **33 categories**, with data spanning May 1996 to September 2023. [amazon-reviews-2023.github.io+1]

It includes rich features such as review text, ratings, helpfulness votes, item metadata (descriptions, images, price), and user-item interaction graphs — designed to support recommendation, retrieval, and other ML/NLP tasks.

Choosen Category: Appliances. Since it had the most density and challenging size of 1.7 Million users and 90K products (distinct)

Category	#User	#Item	#Rating	Density Score ∝ #Rating/(#User×#Item)
Appliances	1.8M	94.3K	2.1M	Highest (approx. 12.35)
Musical_Instruments	1.8M	213.6K	3.0M	High (approx. 7.79)
CDs_and_Vinyl	1.8M	701.7K	4.8M	Medium (approx. 3.80)
Amazon_Fashion	2.0M	825.9K	2.5M	Lowest (approx. 1.51)

DB design Choices:
Foreign key constraint on parent_asin/product_id ensures when a product is removed from catalog, any models trained in future will not recommend products that do not exists in catalog, since the corresponding foreign records from user-item-rating table will also be deleted in cascading fashion.

Data Preparation: since its source was coming from reputed university its was pretty much clean, only needed to transform raw list of JSON objects into tables/dataframes

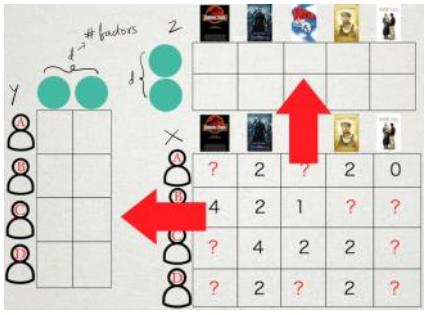
ETL: loaded the transformed tables into SQL database, created pipelines which can accept future incoming data for retraining

Model Selection:

For recommender systems one of popular technique is Matrix factorization, as mentioned in the research paper written by the winners of Netflix price competition

Types:

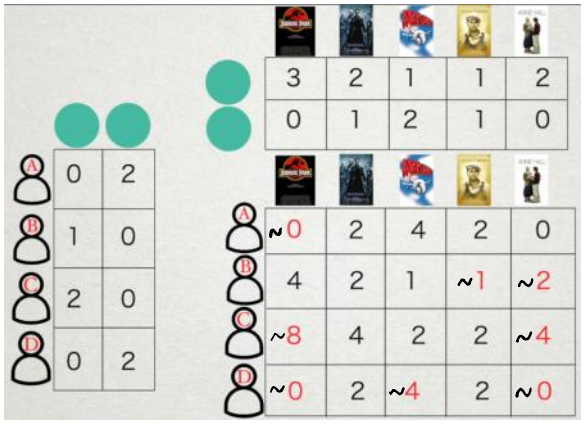
1) Content Based Filtering: "recommends top K similar items based on feature embedding" (title embeddings, cosine similarity)
2) Collaborative Filtering: "recommends top K similar items based on User & Item interaction"



optimization problem

$$\underset{Y,Z}{\operatorname{argmin}} \sum_{i,j} (X_{i,j} - Y_i^T Z_j)^2$$

After finding optimal User & Item matrices, we get:



The filled ratings are approx/predicted, which tells us what item an user is likely to choose

Model Training: from implicit library

ALS, or Alternating Least Squares, is a matrix-factorization method used to predict missing user–item interactions. It's stable, parallelizable, and deterministic, which makes it popular in industry recommender systems. This solves the above optimization problem (modified one)

Model Evaluation: MAP@K, It checks the first K recommendations you showed a user and measures how many of those were actually useful or correct, and how early they appeared in the list. Higher MAP@K means users are more likely to quickly find something they like without scrolling.

Post deployment monitoring: unlike MAP@K metric which can be used with ground truth, in real time we won't have that luxury, so how to determine if our model is performing well or not?, that is by custom indicators (below are not the only but few possible intuitive ones)

Good Recommendation Indicator:

- clicking and being re-directed to a particular product page from landing page
- If they purchases/buys the product
- If they go from one product page to another product page via one of the recommended products

Bad Recommendation Indicator:

- for a search query or product title we get a list of recommendations, and if user scrolls till the end and does not click on any of the recommendations and ends the session or goes back, if the user despite going till the end but maybe comes back and clicks on any recommendations then that is not "bad" (need to track user scrolling till the end)

Further testing can be done using A/B testing in case there are multiple models, user behaviour is monitored to determine models performance.

Incremental training: once we filter out the Good recommendations we can export the user-item-rating records from real time which adds to existing pool of data and if there are any addition or deletion of products from catalog update them accordingly and trigger retraining scripts