Faculty of Information Technology
Ho Chi Minh City University of Education

Recommendation Systems

Retrieval, Scoring and Re-ranking

Lecturer: MSc. Tran Thanh Nha

TA: Ta Cong Phi

Outline

1. Retrieval
2. Scoring
3. Re-ranking

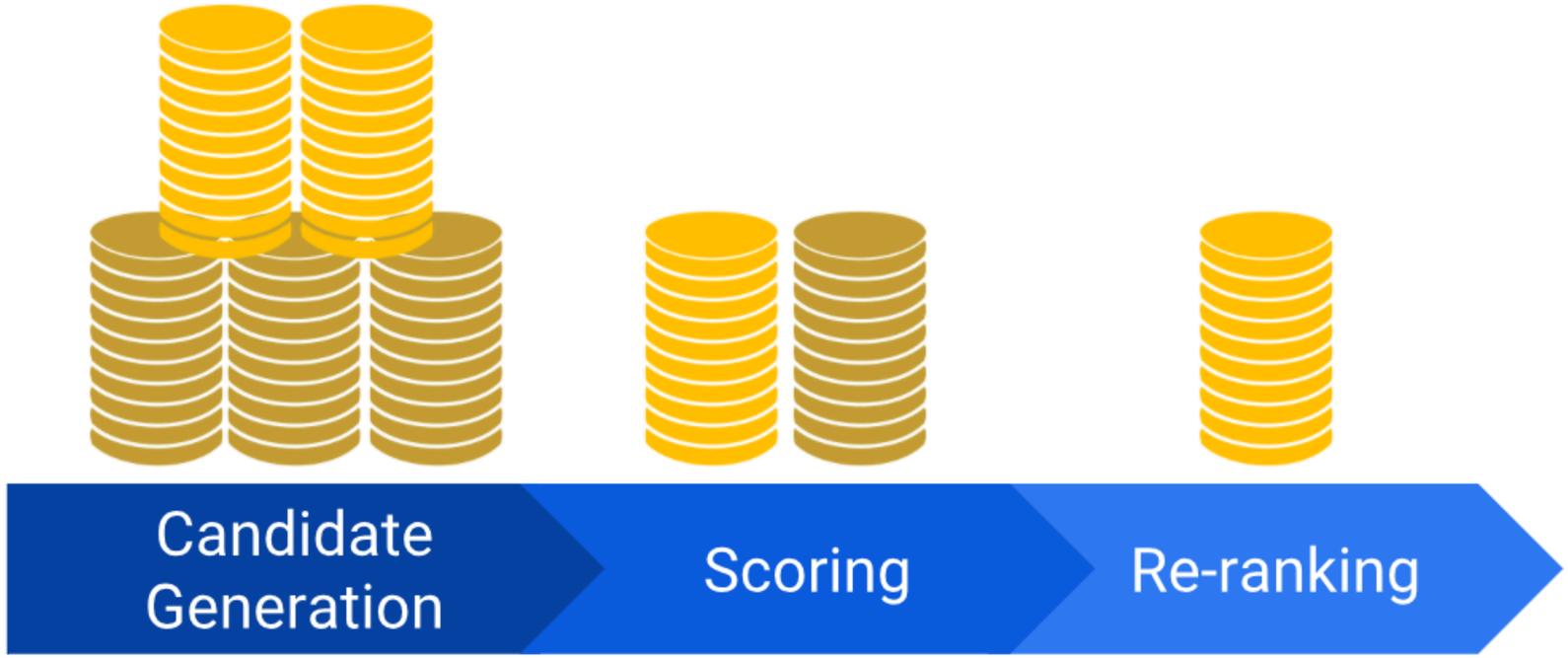
Retrieval

Overview

Introduction

➢ One common architecture for recommendation systems consists of the following components:

- Candidate generation
- Scoring
- Re-ranking



Overview

Introduction

➢ Candidate generation:

- In this first stage, the system starts from a potentially huge corpus and generates a much smaller subset of candidates.

➢ Scoring:

- Next, another model scores and ranks the candidates in order to select the set of items (on the order of 10) to display to the user.

➢ Re-ranking:

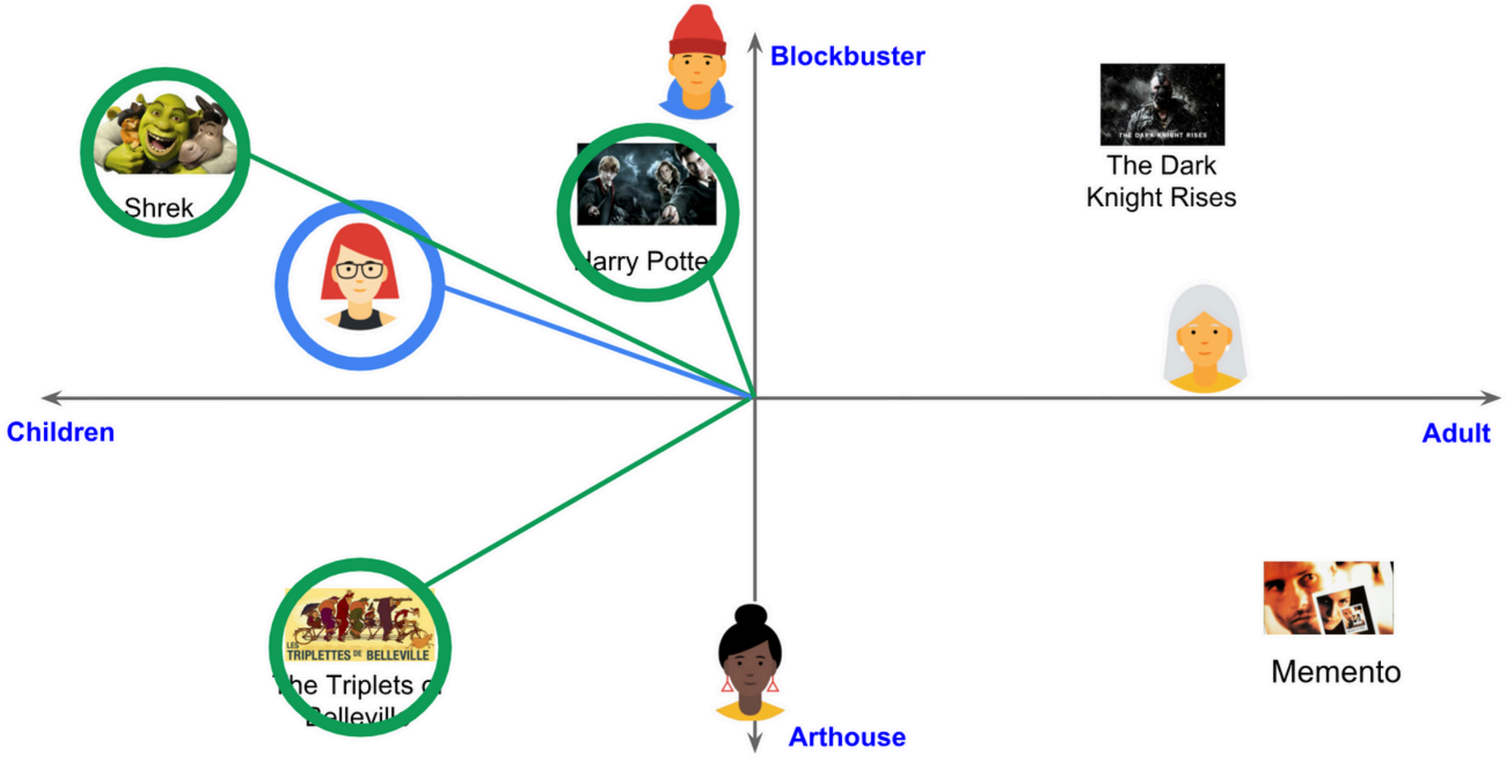
- Finally, the system must take into account additional constraints for the final ranking.

Introduction

- At serve time, given a query, you start by doing one of the following:
  - For a matrix factorization model, the query (or user) embedding is known statically, and the system can simply look it up from the user embedding matrix.
  - For a DNN model, the system computes the query embedding at serve time by running the network on the feature vector x.
- Once you have the query embedding ***q***, search for item embeddings hat are close to ***q*** in the embedding space.

Introduction

- You can use a similar approach in related-item recommendations.
- When the user is watching a YouTube video, the system can first look up the embedding of that item, and then look for embeddings of other items that are close in the embedding space.



Large-scale retrieval

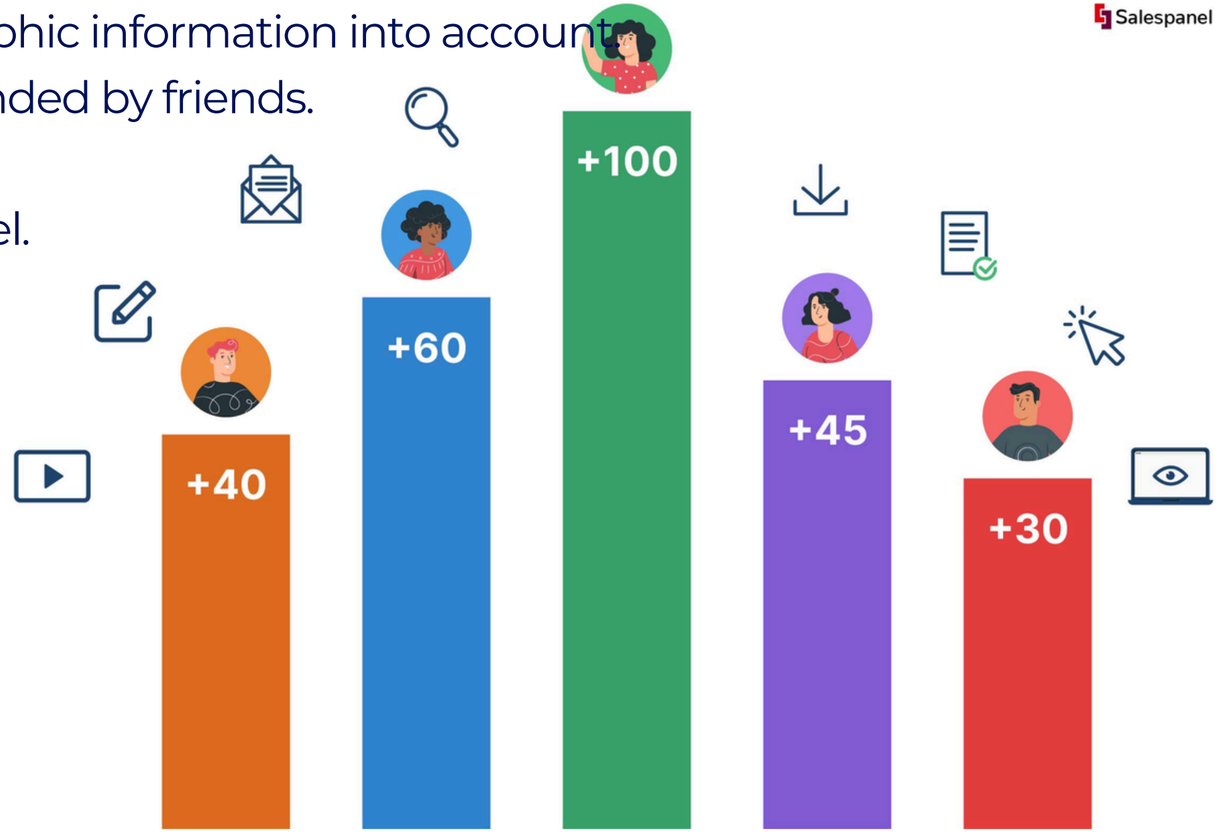
- To compute the nearest neighbors in the embedding space, the system can exhaustively score every potential candidate.
- Exhaustive scoring can be expensive for very large corpora.
  - If the query embedding is known statically, the system can perform exhaustive scoring offline, precomputing and storing a list of the top candidates for each query. This is a common practice for related-item recommendation.
  - Use approximate nearest neighbors. Google provides an open-source tool on GitHub called ScaNN (Scalable Nearest Neighbors).

Scoring

Scoring

Introduction

- After candidate generation, another model scores and ranks the generated candidates to select the set of items to display.
- Such as:
  - "Local" vs "distant" items; that is, taking geographic information into account.
  - A social graph; that is, items liked or recommended by friends.
  - Popular or trending items.
  - Related items from a matrix factorization model.
  - User features that account for personalization.



Introduction

➢ The system combines these different sources into a common pool of candidates.

- Score each candidate: Based on specific features (e.g. user features and item features).
- Score ranking: The items with the highest scores will be selected for display.

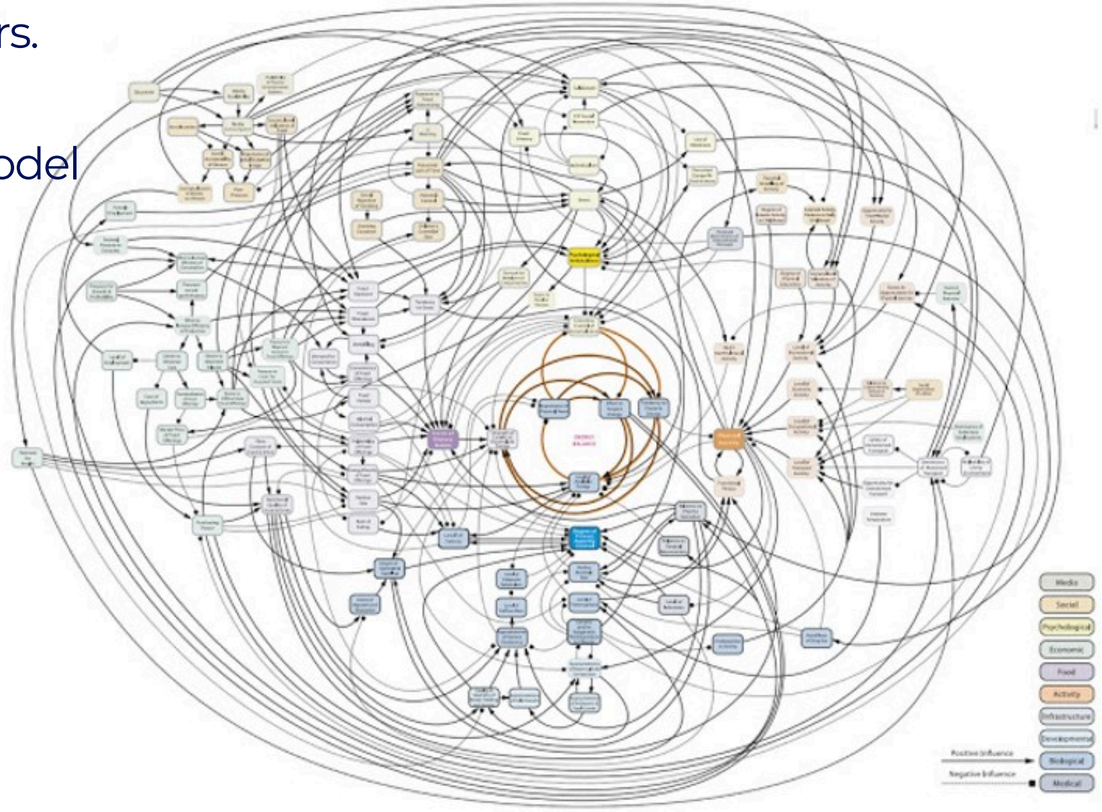
➢ For example, the system can train a model to predict the probability of a user watching a video on YouTube given the following:

- Query features (for example, user watch history, language, country, time)
- Video features (for example, title, tags, video embedding)

➢ The system can then rank the videos in the pool of candidates according to the prediction of the model.

Why not let the candidate generator score?

- Since candidate generators compute a score (such as the similarity measure in the embedding space), you might be tempted to use them to do ranking as well.
- However, you should avoid this practice for the following reasons:
  - Some systems rely on multiple candidate generators.
  - With a smaller pool of candidates, the system can afford to use more features and a more complex model that may better capture context.

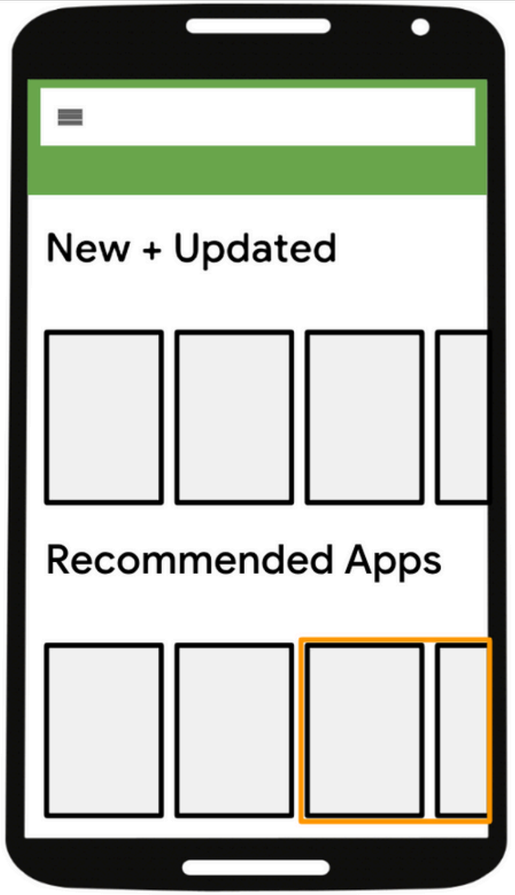


Choosing an objective function for scoring

- The choice of scoring function can dramatically affect the ranking of items, and ultimately the quality of the recommendations.
- Example:
  - Maximize click rate: If the scoring function optimizes for clicks, the systems may recommend click-bait videos.
  - Maximize watch time: If the scoring function optimizes for watch time, the system might recommend very long videos, which might lead to a poor user experience.
  - Increase diversity and maximize session watch time: Recommend shorter videos, but ones that are more likely to keep the user engaged.

Positional bias in scoring

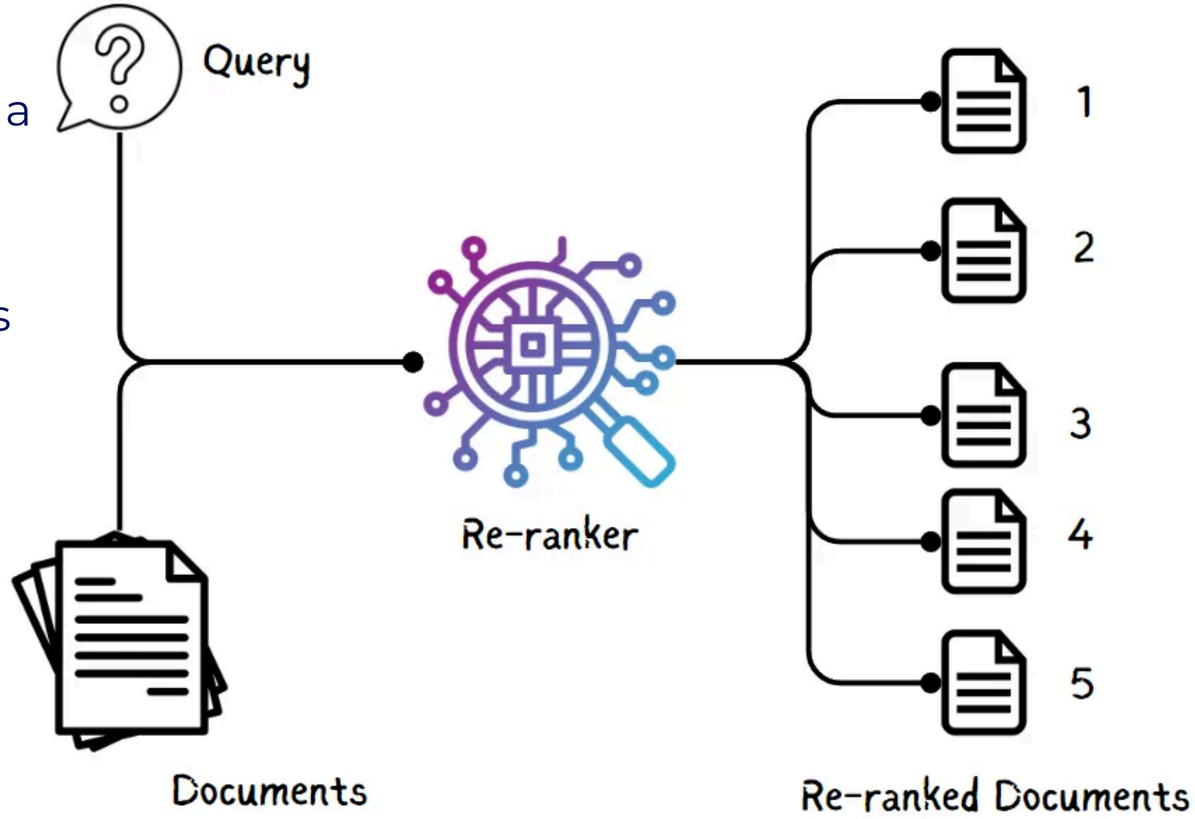
- Items that appear lower on the screen are less likely to be clicked than items appearing higher on the screen.
- However, when scoring videos, the system usually doesn't know where on the screen a link to that video will ultimately appear.
- Querying the model with all possible positions is too expensive.
- Solutions:
  - Create position-independent rankings.
  - Rank all the candidates as if they are in the top position on the screen.



Re-ranking

Introduction

- In the final stage of a recommendation system, the system can re-rank the candidates to consider additional criteria or constraints.
- One re-ranking approach is to use filters that remove some candidates.
- Example:
  1. Training a separate model that detects whether a video is click-bait.
  2. Running this model on the candidate list.
  3. Removing the videos that the model classifies as click-bait.



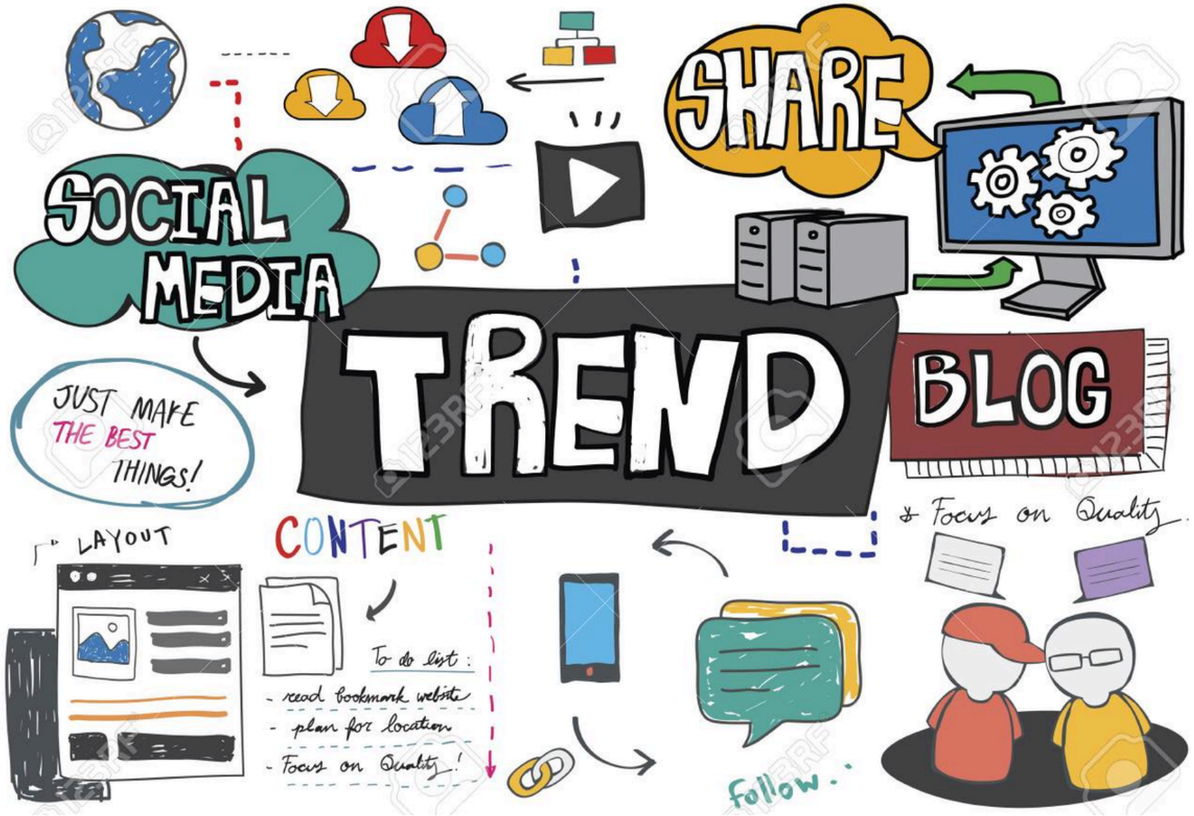
Introduction

- Another re-ranking approach is to manually transform the score returned by the ranker.
- Example: The system re-ranks videos by modifying the score as a function of:
  1. Video age (perhaps to promote fresher content).
  2. Video length.

Re-ranking

Freshness

- Most recommendation systems aim to incorporate the latest usage information, such as current user history and the newest items.
- Keeping the model fresh helps the model make good recommendations.



Re-ranking

Freshness

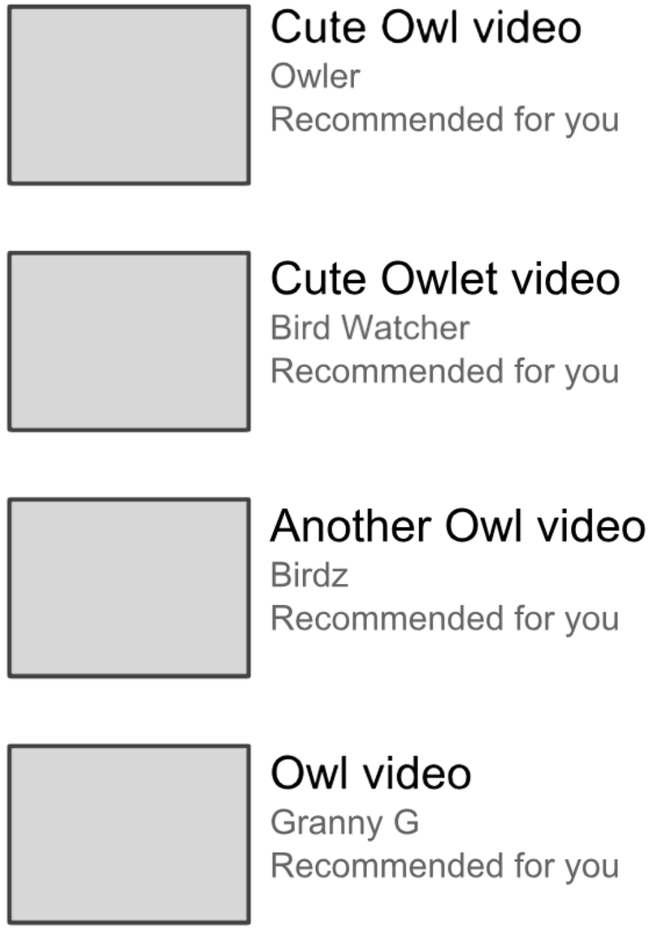
➢ Solutions:

- Re-run training as often as possible to learn on the latest training data.
- Create an "average" user to represent new users in matrix factorization models.
- Add document age as a feature.

Re-ranking

Diversity

- If the system always recommend items that are "closest" to the query embedding, the candidates tend to be very similar to each other.
- This lack of diversity can cause a bad or boring user experience.
- For example, if YouTube just recommends videos very similar to the video the user is currently watching, such as nothing but owl videos (as shown in the illustration), the user will likely lose interest quickly.



Re-ranking

Diversity

➢ Solutions:

- Train multiple candidate generators using different sources.
- Train multiple rankers using different objective functions.
- Re-rank items based on genre or other metadata to ensure diversity.

Re-ranking

Fairness

- Your model should treat all users fairly.
- Therefore, make sure your model isn't learning unconscious biases from the training data.

Re-ranking

Fairness

➢ Solutions:

- Include diverse perspectives in design and development.
- Train ML models on comprehensive data sets.
- Track metrics (for example, accuracy and absolute error) on each demographic to watch for biases.
- Make separate models for underserved groups.

Thanks for your attention!!