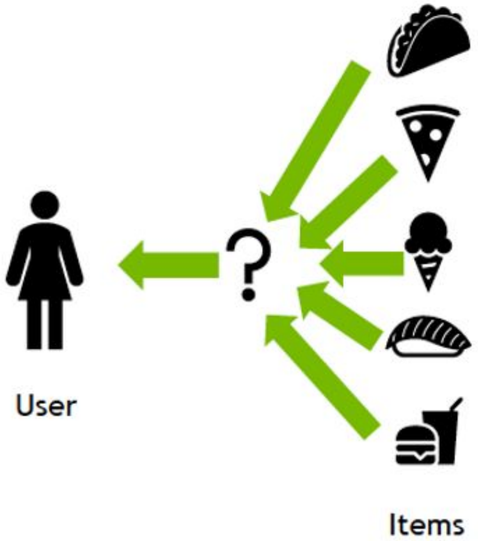
Recommender systems

By Olena Bugaiova

Recommender systems

help solve information overload by providing users with personalized content and showing relevant products from a wide range of selections



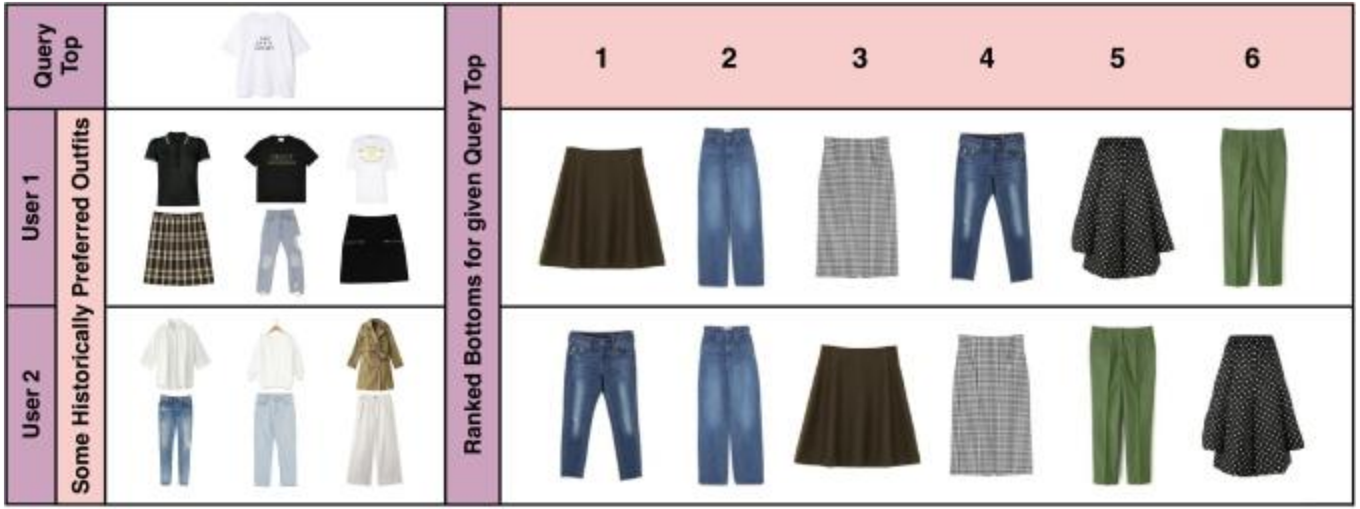
Examples of items to recommend

- songs to play on Spotify
- movies to watch on Netflix
- news to read about your favourite newspaper website
- products to purchase on Amazon

Read more on the nvidia article

Why RecSys are important

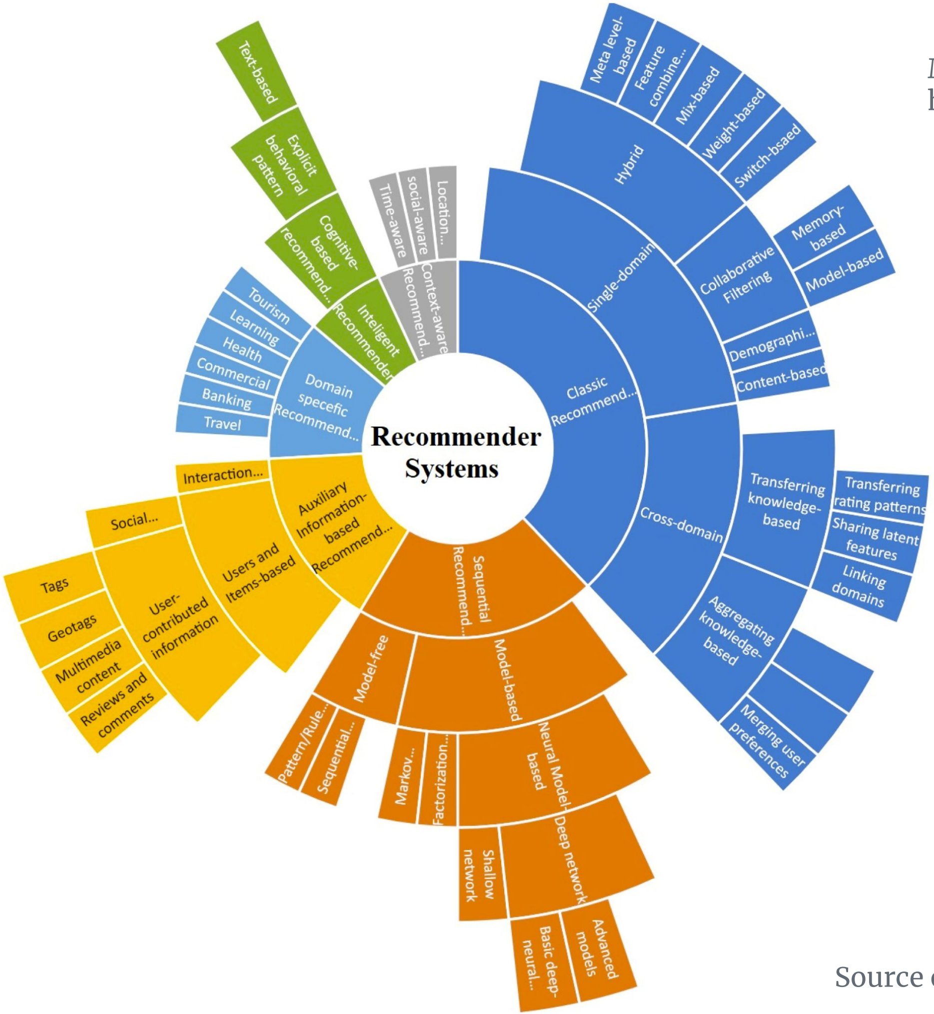
- Increase customer satisfaction
- Beneficial to both service providers and customers



Approaches

- Content based
  - User / item information
- Collaborative filtering
  - User-item interaction
    - Model based
      - Matrix Factorization
      - Deep Learning
    - Memory based
      - User based
      - Item based
- Context-aware
  - Context information
- Hybrid
  - Combining approaches

And more...



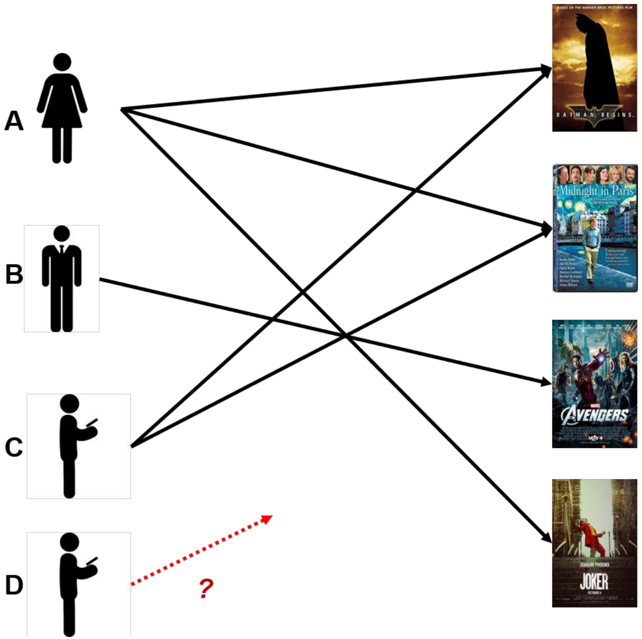
More recently, some variations have been proposed

- sequential recommendation
- session-based recommendation

Source of the image [research paper]

Specifics

Cold start problem



New item problem: when a new item is added to the catalogue and none has rated this item it will never be recommended

New user problem: when a new user has no rating it is impossible to predict his/her rating

Solution: using additional data such as demographic data or better processing existing data

Explicit / implicit feedback

Explicit

Information people provide in response to a specific request from the app

- Request explicit feedback only when necessary
- Always make providing explicit feedback a voluntary task
- Act immediately

Implicit

A wide range of information that arises as people interact with your app's features
You are what you do. This defines you

- Always secure people's information
- Help people control their information
- When possible, use multiple feedback signals to improve suggestions

Read more about explicit / implicit feedback

Data Structures

When number of users and items increases interaction data can have a lot of missing or zero values for instance zero clicks of a user on an item

Sparse matrices are memory efficient data structures that enable us to store large matrices with very few non-zero elements

An example of implementation

```
# row indices of non-zero entries
row_ind = np.array([0, 1, 1, 3, 4])

# column indices of non-zero entries
col_ind = np.array([0, 2, 4, 3, 4])

# values of non-zero entries
data = np.array([1, 2, 3, 4, 5], dtype=float)
```

Dense Matrix

1	2	31	2	9	7	34	22	11	5
11	92	4	3	2	2	3	3	2	1
3	9	13	8	21	17	4	2	1	4
8	32	1	2	34	18	7	78	10	7
9	22	3	9	8	71	12	22	17	3
13	21	21	9	2	47	1	81	21	9
21	12	53	12	91	24	81	8	91	2
61	8	33	82	19	87	16	3	1	55
54	4	78	24	18	11	4	2	99	5
13	22	32	42	9	15	9	22	1	21

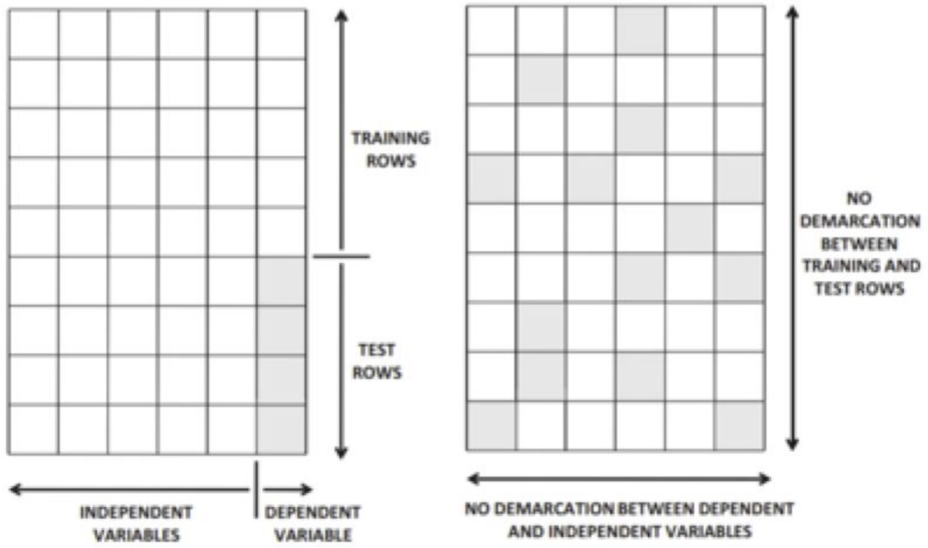
Sparse Matrix

1	.	3	.	9	.	3	.	.	.
11	.	4	.	.	.	.	.	2	1
.	.	1	.	.	.	4	.	1	.
8	.	.	.	3	1	.	.	.	.
.	.	.	9	.	.	1	.	17	.
13	21	.	9	2	47	1	81	21	9
.	.	.	.	.	.	.	.	.	.
.	.	.	.	19	8	16	.	.	55
54	4	.	.	.	11	.	.	.	.
.	.	2	.	.	.	.	22	.	21

Libraries:	Classes:
Scipy.sparse	**bsr_array** – Block Sparse Row array
	coo_array – A sparse array in COOrdinate format
Read more in [this article]()	**csc_array** – Compressed Sparse Column array
	csr_array – Compressed Sparse Row array

Evaluating recommender systems

- The success of the recommender system can be measured through the number of recommendations that are followed
- It is usually done by hiding some of the interactions in historical data in order to simulate the knowledge of which recommendations a user will act upon



(a) Classification (b) Collaborative filtering

In collaborative filtering It is more meaningful to speak of **training and test entries rather than training and test rows**

Split by users: less common evaluation approach. It requires to have the capability to recommend items for new (cold-start) users, which many approaches do not support

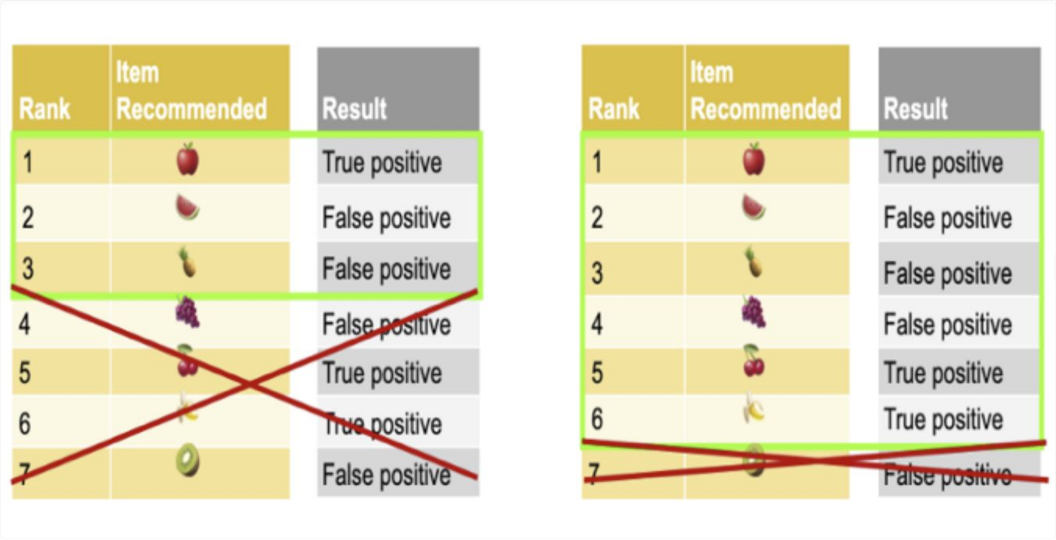
Metric for performance evaluation

Name	Characteristics
MAE and RMSE	Are good for explicit rating, because they compare the exact values of the ratings
Accuracy, precision, Recal	Are good to measure the amount of items that we recommended and a users followed
F1 score	F1 score is the harmonic mean of precision and recall

Order matters (ranking)

Hit Rate	If a user rated one of the top-10 items we recommended, we consider it is a "hit" and the whole hit rate of the system is the count of hits, divided by the number of test users
Average Reciprocal Hit Ranking (ARHR)	We get more credit for recommending an item in which user rated on the top of the rank than on the bottom of the rank

Mean Average Precision @ cutoff (MAP@K)



Characteristics

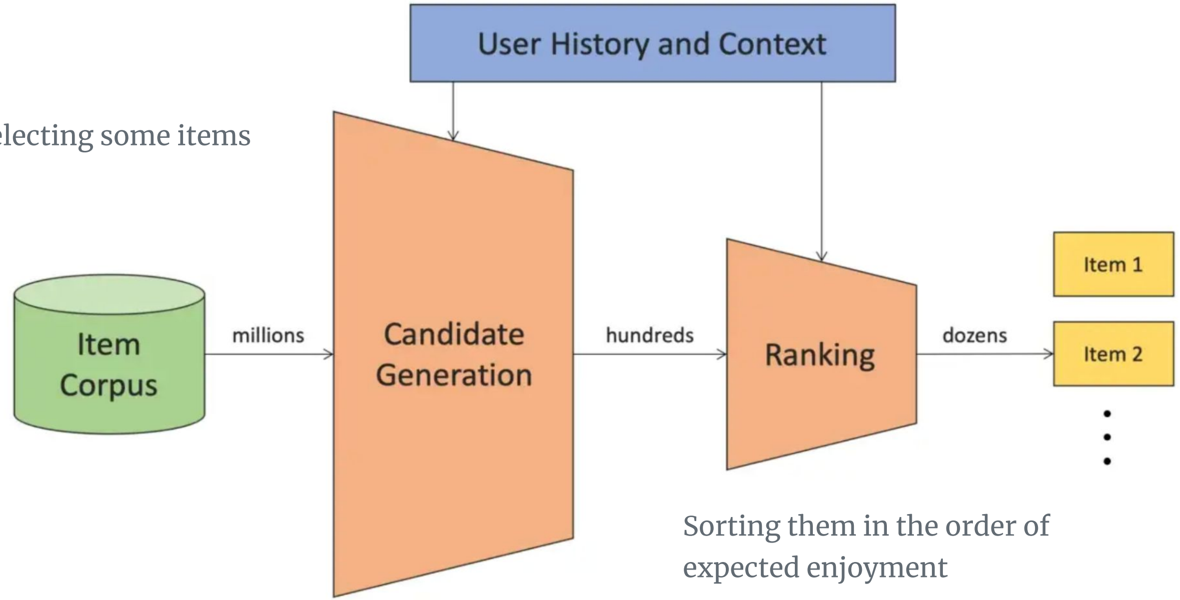
- We are treating the recommendation like a ranking task
- We want the most likely / relevant items to be shown first
- We can calculate the precision at each cutoff

Formula

We average these precisions P(k = i) for every *cutoff* that was correct
For each user, we calculate the AP@N, and then average for all users

Two stages of Recommender Systems

Selecting some items



Sorting them in the order of expected enjoyment

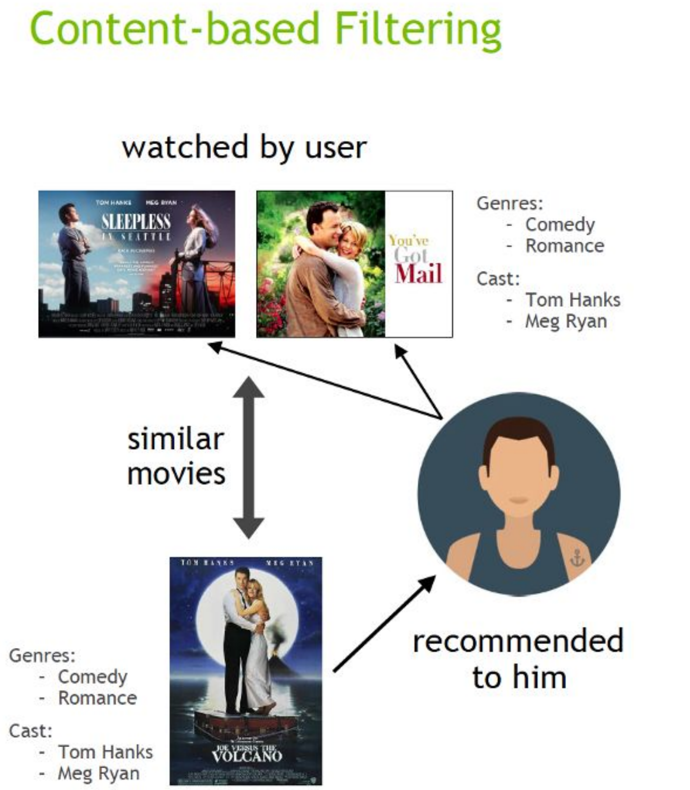
2-stage Recommender System (inspired by YouTube)

Motivation: Item catalogues can grow to millions, hundreds of millions, even billions in extreme cases. Scoring is computationally expensive. Scoring every item for every user just isn't feasible

In practice, you start by quickly selecting a relevant subset of those items

Read more on Towards Data Science and on the nvidia article

Approaches



- The algorithm uses a sequence of discrete, pre-tagged characteristics of an item (this is the content part) to recommend other items with similar properties

- This approach is best suited when there is sufficient information available on the items but insufficient on the user-item interactions

Downsides:

- Defining such similarity function might be tricky and burdensome since many items do not have explicit features that can be easily quantified

- It can require a great amount of computational resources to calculate pairwise similarity scores

Read more on [nvidia article]

- The algorithms recommend items (this is the filtering part) based on preference information from many users (this is the collaborative part)

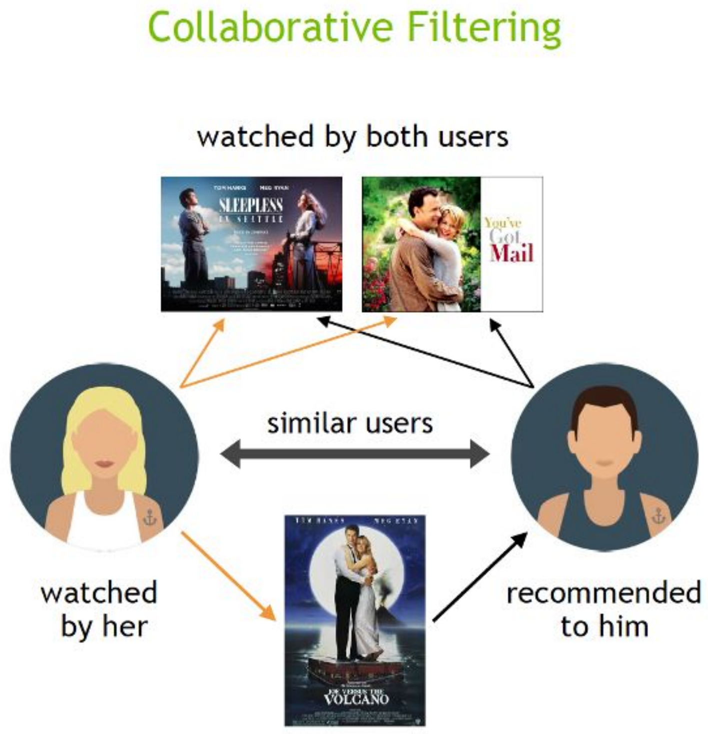
- The idea is that if some people have made similar decisions and purchases in the past, then there is a high probability they will agree on additional future selections

Downsides:

- Not efficient having a small amount of interactions i.e. cold start problem

- Less efficient for recommending less popular items that some users might prefer i.e. in the case of unique tests

- Doesn't encounter that user preferences on items might change over time

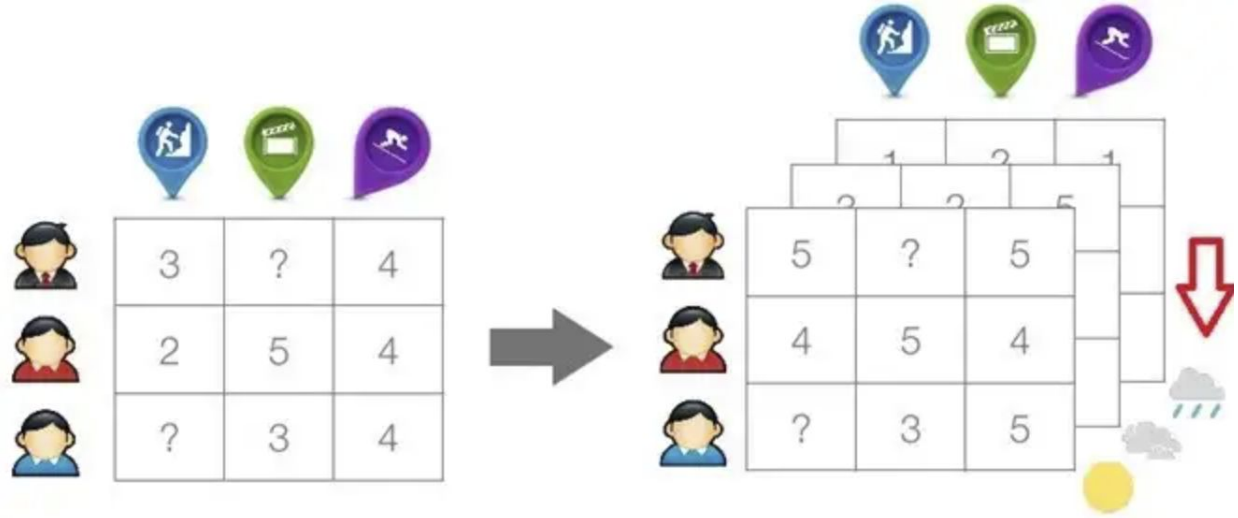


Read more on [nvidia article]

Context-aware Filtering

It takes the context information into account, such as time, location, weather, persona, social media and so on, to provide a better recommendations

In the image below, we add weather as a context information



Memory-Based Collaborative Filtering

- User Based
- Item Based

Step 1: Finding the similarity between all the item pairs for item based / user pairs for user based approach using neighborhood techniques. The most common ones are:

Euclidian distance

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Cosine distance

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$

Pearson correlation coefficient

$$r = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^{n} (y_i - \bar{y})^2}}$$

Step 2: Executing a recommendation system

- We generate predictions of rating of a given user on a given item
  - based on the ratings of this user to similar items for item based approach
  - based of the ratings by similar users on this item for user based approach
- We compute this using weighted sum of the ratings of the other similar items / users

Read more on [the medium article]

Matrix Factorization for Recommendation

- It has become a dominant methodology within the collaborative filtering based recommendations
- MF can be used to calculate similarity in user’s ratings or interactions to provide recommendations

In the simple user-item matrix below, Ted and Carol like movies B and C. Bob likes movie B. To recommend a movie to Bob, matrix factorization calculates that users who liked B also liked C, so C is a possible recommendation for Bob

Latent factors

Tie users and products together. We can see that users with similar tastes exists and items with similar characteristics exist but we don't know what they are. They are related to each other as a group. Users of the same type of group interested in one item indicate that other users of this group might like this item too

In the image, prisoners can see items such as horses but can't see people carrying those items. In this example, horses are observable and people carrying items are unobservable i.e **latent factors**

Read more about [Latent features]

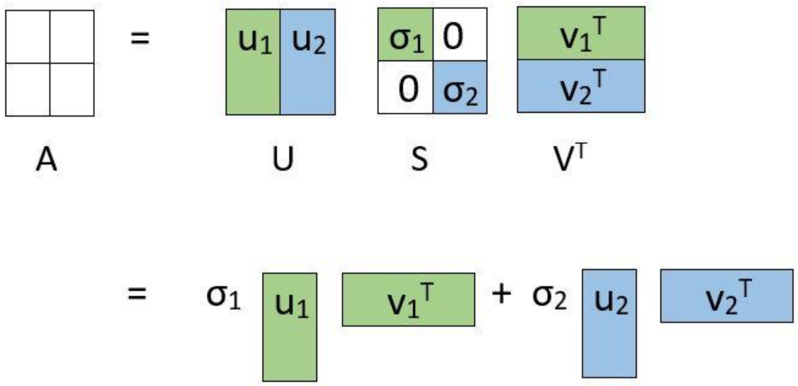
Singular Value Decomposition (SVD)

is the factorization of a matrix into 3 matrices

$$A = U\Sigma V^T$$

- **U** is an (m x m) orthogonal matrix, the left singular vectors
- **Σ** is an (m x n) nonnegative rectangular diagonal matrix, the singular values
- **V** is an (n x n) orthogonal matrix, the right singular vectors

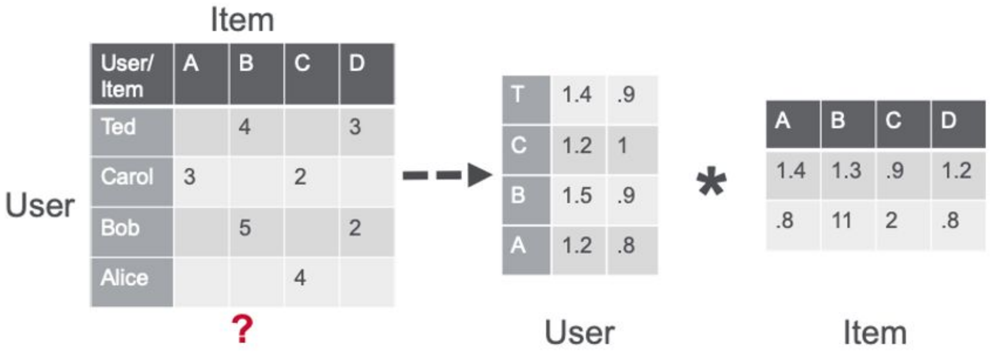
The decomposition allows us to express our original matrix as a linear combination of low-rank matrices



In a practical application, we observe that only the first few, say K, singular values are large. The rest of the singular values approach zero

Alternating Least Square (ALS)

- Algorithm approximates the sparse user item rating matrix u-by-i as the product of two dense matrices, **user** and **item factor matrices** of size u × f and f × i
- The factor matrices represent **latent** or hidden features which the algorithm tries to discover



- For each user and for each item, the ALS algorithm iteratively learns numeric “factors”
- In each iteration, the algorithm alternatively holds one factor matrix fixed and optimizes for the other by minimizing the loss function with respect to the other
- This process continues until it converges

Non-negative Matrix Factorization (NMF)

A method used to factorize a **non-negative matrix**, X, into the product of two **non-negative** lower rank matrices, W and H, such that WH approximates an optimal solution of X

NMF able to automatically extract sparse and **easily interpretable factors**

As an example, we take a gray-level image of a face containing p pixels, and squash the data into a single vector. Let the rows represent the pixels, and the columns each represent one image

In the case of facial images, the basis images are features such as eyes, noses, moustaches, and lips, while the columns of H indicate how much each feature is present in each image

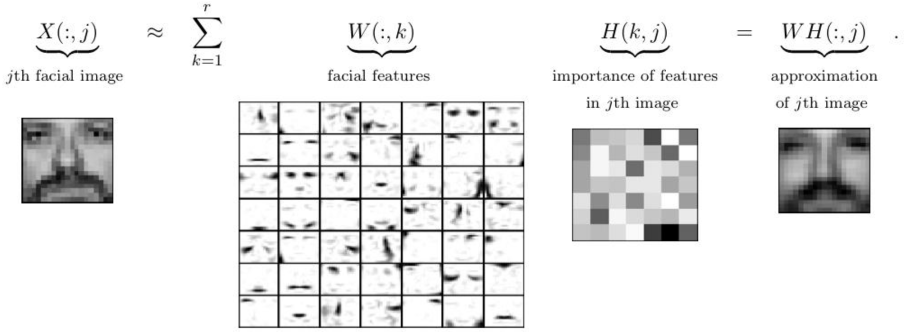


Figure 1: Decomposition of the CBCL face database, MIT Center For Biological and Computation Learning (2429 gray-level 19-by-19 pixels images) using $r = 49$ as in [79].

The appropriate **initialization can be critical** in getting meaningful outputs

Libraries with implementation for Collaborative Filtering

Sklearn for NMF and SVD

```
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import NMF
```

```
svd =  TruncatedSVD(
    n_components = 6,
    algorithm = 'randomized',
    n_iter = 50,
    n_oversamples = 1000,
    power_iteration_normalizer = 'OR',
    random_state = 22
    )
```

```
svd.singular_values_
```

n_components – the number of the first few large singular values

```
nmf = NMF(
      init = 'nndsvda',
      solver = 'cd',
      n_components = 2,
      alpha_W = 1.5,
      alpha_H = 0,
      l1_ratio = 0.92,
      max_iter = 1000,
      tol = 0.01,
      random_state = 22,
      verbose = 10
      )

W = nmf.fit_transform(input_matrix);
H = nmf.components_;
```

n_components – the number of latent factors

Implicit for ALS and LMF

```
from implicit.als import AlternatingLeastSquares
from implicit.lmf import LogisticMatrixFactorization
from implicit.evaluation import precision_at_k
from implicit.evaluation import mean_average_precision_at_k
```

Good for **implicit** feedback

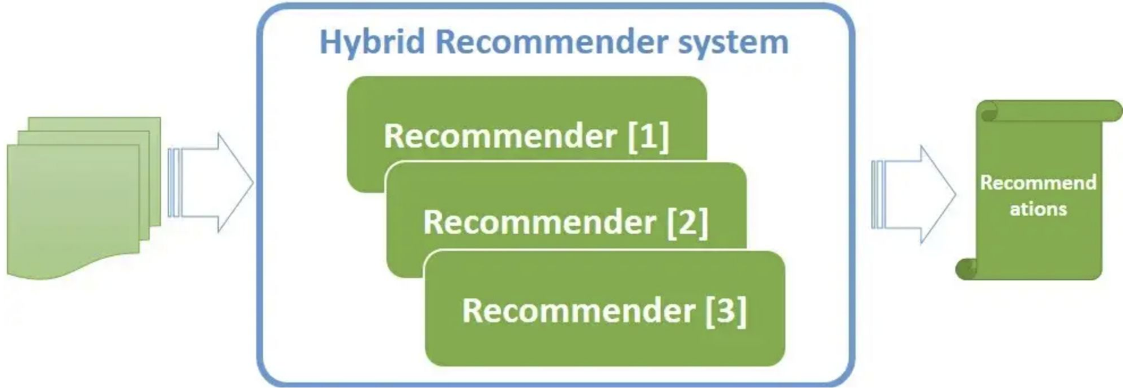
```
model = AlternatingLeastSquares(
     alpha = 1,
     regularization = 30,
     factors = 13,
     iterations = 15,
     random_state = 22
     )

item_factors = model.item_factors
user_factors = model.user_factors
```

factors – the number of latent factors

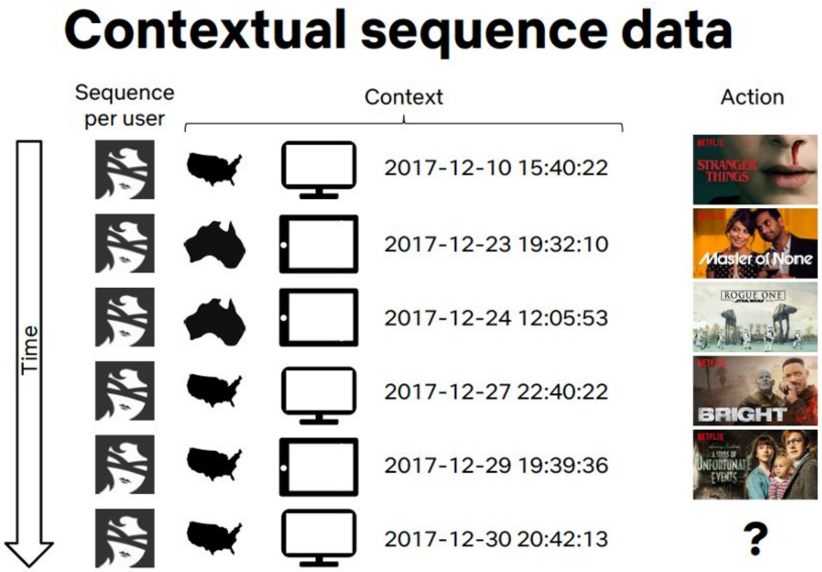
Hybrid Filtering

Combine collaborative filtering and content /context-based methods, to build a more robust RS, like the weighted method, which is a linear combination of weighted RS



Read more about [types of Hybrid Recommendation System]

Sequential and session-based recommendation models



- Use the sequence of user item interactions within a session in the recommendation process

- Examples include predicting the next item in an online shopping cart, the next video to watch, or in the booking.com example, the next travel destination of a traveler

- Library: Transformers4Rec developed by the NVIDIA Merlin

Read more on [this article]

RecSys

In production environment

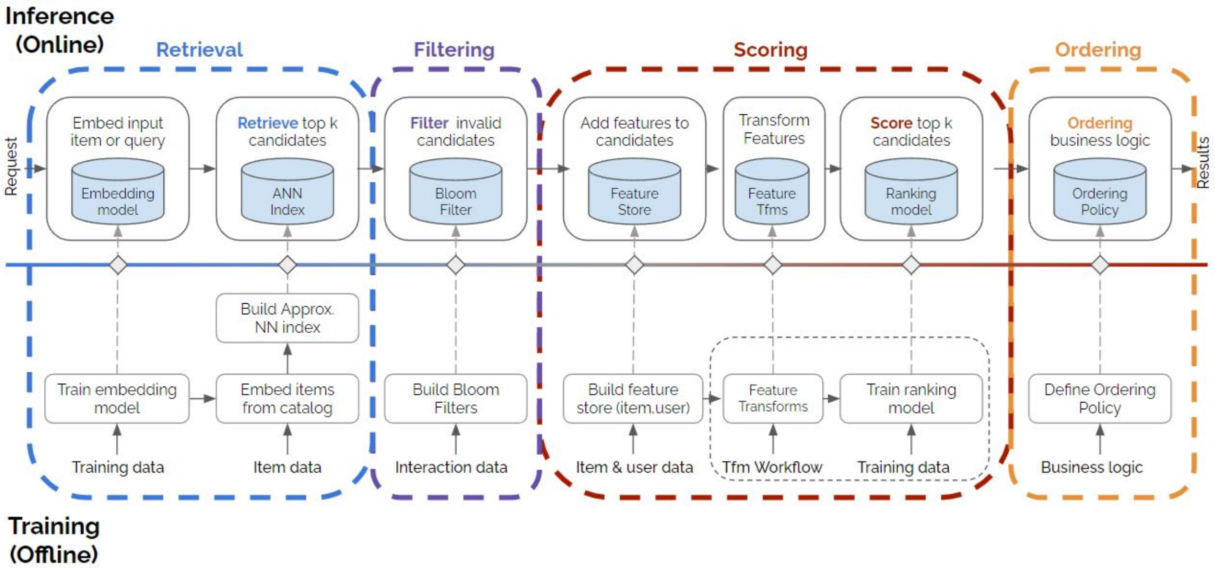
4 stages

We discussed previously these two stages:
1. **Candidate Retrieval**
2. **Ranking**

It worth adding two more:

Filtering stage: allows you to apply business logic rules when there are items that you don't want to show to the user:

- when the item is out of stock
- when it's not age appropriate
- when the user has already consumed the content
- when licencing rights don't allow you to show it in this user's country



Order up! the best list is unlikely to fully align with the individual item scores. Instead we may want to provide a diverse set of items to the user or even show them items outside their normal pool to explore spaces they haven't seen

Read more on nvidia article

4 stages. Examples

	Retrieval	Filtering	Scoring	Ordering
Music Discovery	Find similar songs based on nearest neighbour search	Remove tracks users listened before	Predict likelihood that a user will listen to a song	Trade-Off between score, similarity, BPM, etc
Social Media	Find new posts in user's network	Remove posts from blocked and muted users	Predict likelihood that a user will interact with it	Change order that adjust posts are from different authors
Online Store	Find items which are usually co-purchased	Remove items which are out of stock	Predict likelihood that a user will purchase an item	Reorder items based on price points
Streaming Service	Find items based on different rows/shelves/topics	Remove items which are not available for user's country	Predict user's stream time per item	Organize recommendations to fit genre distributions

RecSys Competitions

OTTO – Multi-Objective Recommender System, the competition is over

Learning Equality - Curriculum Recommendations, the competition is over