

# 01- Lecture for Recommendation

## Supervised Machine Learning Recommendation

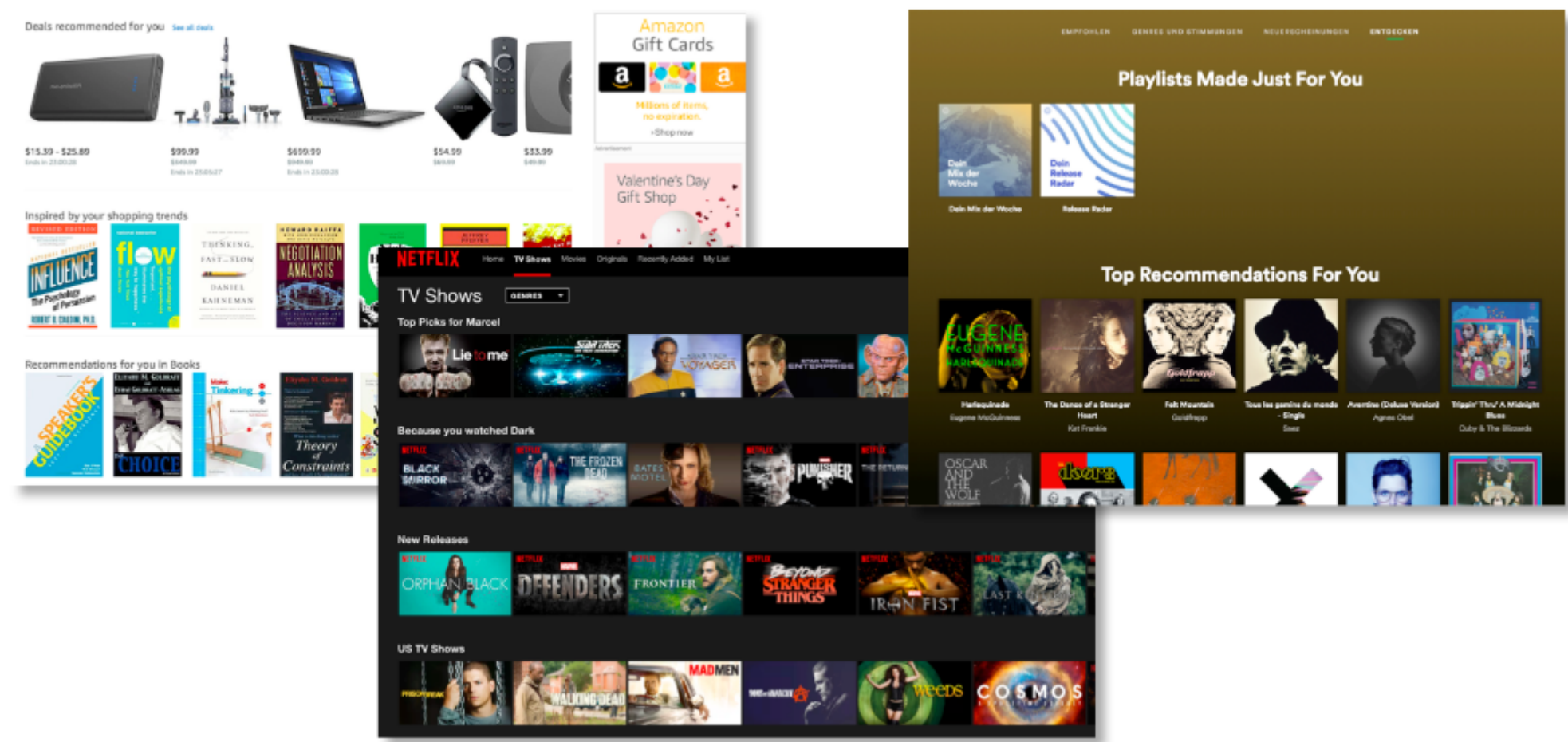


Today we will explore a different powerful application of machine learning: **Recommendation Engines**

### 1. Introduction

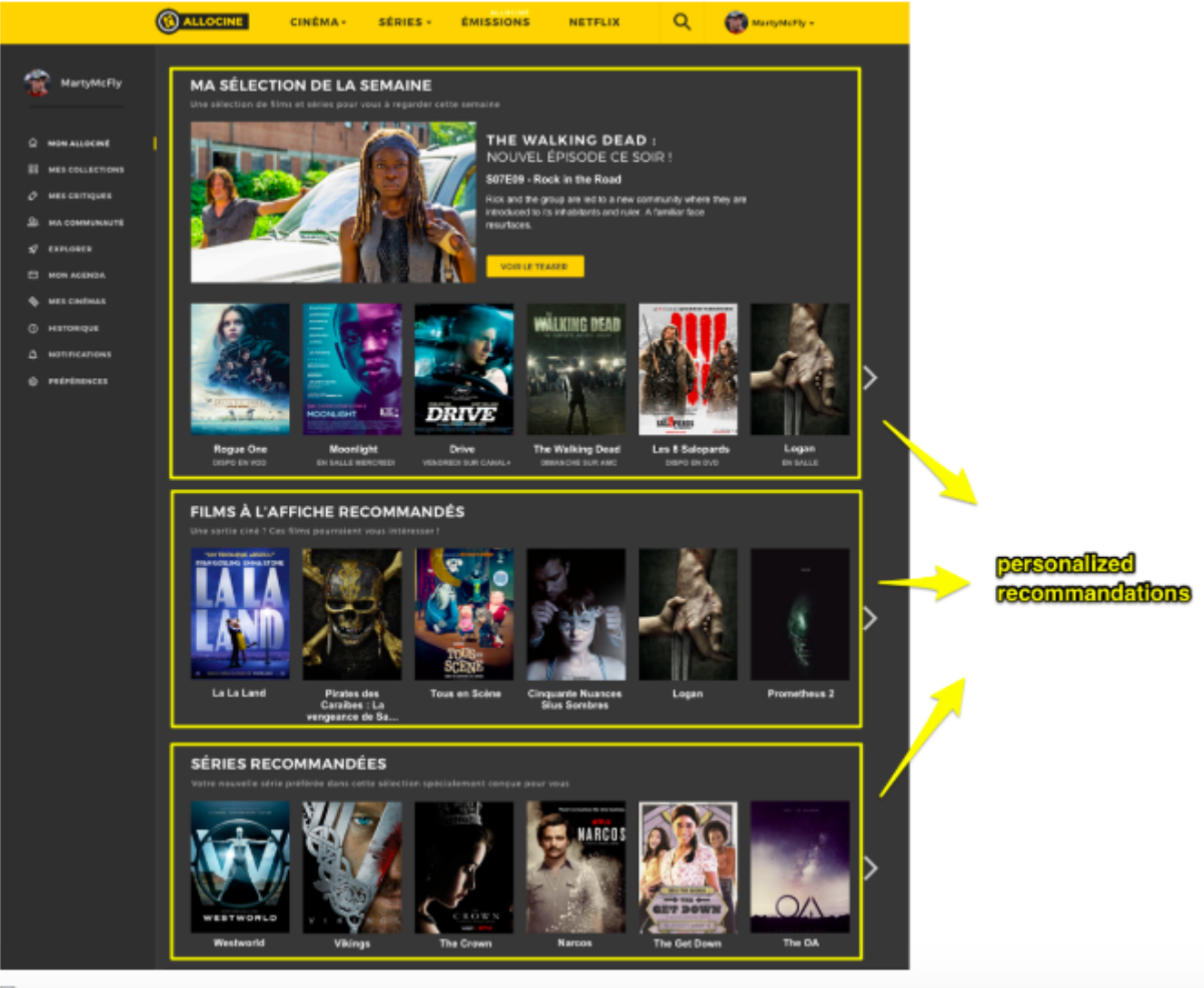
#### 1.1. Recommendation everywhere

Recommendation engines are very popular. Mainly because it successfully drives more revenue/traffic/retention/etc. to your business - in an automated way.



#### 1.2. Classical data scientist mission

Building a recommendation engine is a classical example of a mission for a data scientist.



1.3. Problem formulation: predicting ratings








Unlike precedent problems and algorithms, recommendation systems are part of a category of systems that automatically learn what features to use.

It can be considered as a **supervised machine learning** problem as we feed the targets (ratings) to the model for training.

Let's formulate the problem more precisely. We will consider the example of movie recommendations.

Users rate some movies with a grade (from 0 to 5 for example), and you try to predict what other movies might be interesting to a user.

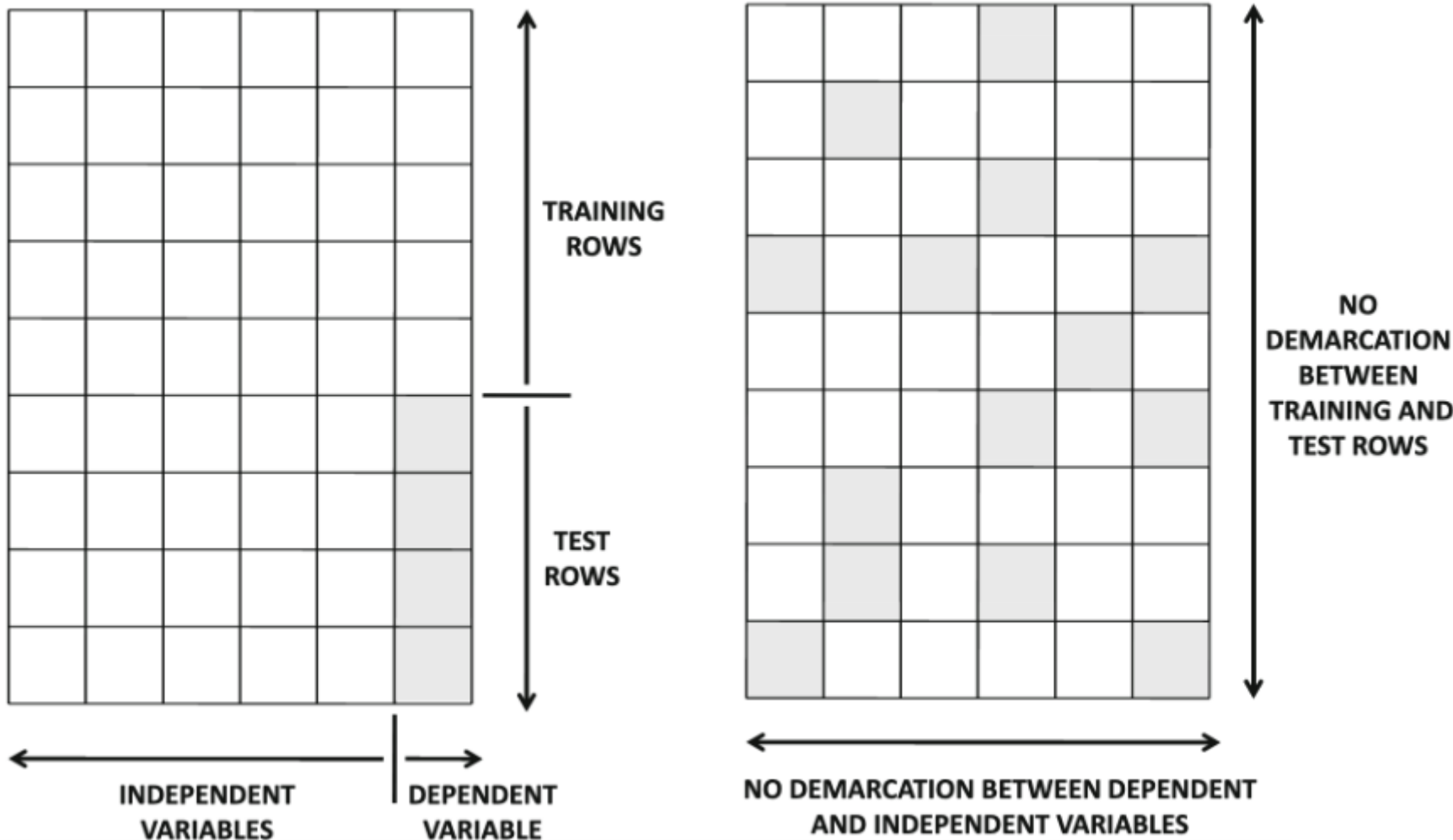
The goal is to **"fill the ratings matrix"**.

			
	5		3
		4	
		4	3
	5	1	3

We call:

- $nu$  = Number of users
- $nm$  = Number of movies
- $ri,j$  = 1 if user  $j$  has rated movie  $i$  (0 otherwise)
- $yi,j$  = rating given by user  $j$  to movie  $i$  (defined only if  $ri,j=1$ )

**Hint:** Recommendation is an application of a **regression** problem but as you can see below, it is still slightly different from what we have worked on before (on the left is a classical supervised ML problem, on the right is a recommendation problem):



1.4. Metrics and evaluation

There are several ways to evaluate your model, the most common is the so-called **MAP@K**, meaning **Mean Average Precision At K**.

What does that mean? Basically, this is a way to compute the ratio of good recommendation, for the **K** first recommended items.

Imagine we recommend **K=5** movies with the following ids: **[7, 3, 11, 5, 1]**, while the target movies to be recommended are the following: **[1, 2, 3, 4, 5]**.

Then we could compute an array of right/wrong with 0s and 1s **[0, 1, 0, 0, 1, 1]**: only the movies **3**, **5** and **1** have been correctly recommended, not the movies **7** and **11**.



So the will be the following:

P@1 P@2 P@3 P@4 P@5  
0/1 1/2 0/3 2/4 3/5

And the will be :

Finally, if you want to compute the MAP@5 on our example (with 3 users), it will be the sum of the AP@5 of each user, divided by the number of users in the dataset.

**Note:** the order matters in the computation of the MAP@K.

## 1.5. Several Approaches

There are several approaches to tackle this problem, that can be divided into three main groups:

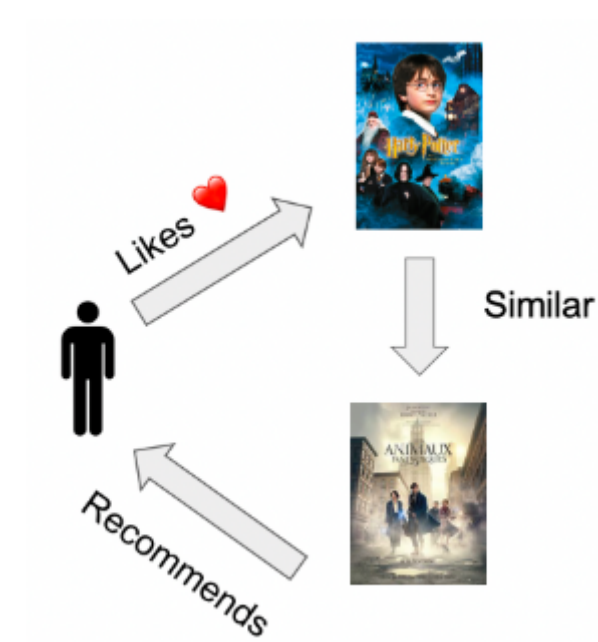
- **Content-based filtering**
- **Collaborative filtering**
- **Hybrid approach** (combining content-based and collaborative filterings)

## 2. Content-based filtering

### 2.1. Intuition








In content-based filtering, the model tries to **recommend items that are similar to those that a user liked in the past**.

For instance, given some attributes of the movies that the user liked (realisator, genre, etc.), we could then recommend similar movies to the user.



Let's suppose that we also have (in addition to our incomplete ratings matrix) a second a matrix containing features associated to our items.








The degree of Fun and the degree of Romance in the movie - between 0 and 1. We chose those specific features, but we could have selected many other features (nationality, genre, box office, etc...).

				Fun	Romance
	5		3	1	0
		4		0.2	1
		4	3	0	0.9
	5	1	3	0.8	0.2

As you can see, we have **features that describe the items**. We will call those features  $X_i$  (for feature vector of movie  $i$ ). Here for example  $X_4=[0.8,0.2]$ .

Now we want to build a **user profile that indicate the type of items this user likes**. Those will be the parameters that the model learns.

Let's call  $W_j$  the parameter vector for user  $j$ . This vector has the same length as  $X_i$  vectors.

				Fun	Romance
	5		3	1	0
		4		0.2	1
		4	3	0	0.9
	5	1	3	0.8	0.2
	$W^1_0$	$W^2_0$	$W^3_0$		
	$W^1_1$	$W^2_1$	$W^3_1$		

The dot product  $X_i \cdot W_j$  will output the **prediction for the rating of the movie j by the user i**.

For example, if we find out (see below how) that the parameters  $W_2=[1,4]$ . Then, the predicted ratings for Nemo would be:

$1 + 4 \cdot 0 = 1$   $y_{1,2}=1 \cdot 1+4 \cdot 0=1$ ... which kind of makes sense.

## 2.2. Gradient descent (again )

Again, in order to learn, we will compute our squared error between our prediction (  $X_i \cdot W_j$  ) and the ratings (  $y_{i,j}$  ). Then try to minimize this error using **gradient descent** algorithm.

The loss function for a single user j is the following:
$$J = \frac{1}{2} \sum_{i:r_{i,j}=1}^{n_m} (W^j \cdot X^i - y_{i,j})^2$$
$$L_j = \sum_{i:r_{i,j}=1}^n (W_j \cdot X_i - y_{i,j})^2$$

So basically we sum over all the movies with a rating (this is what "  $i:r_{i,j}=1$  " means), and we compute the squared error between the prediction and the rating. We then minimize  $W_j$  with gradient descent.

If we want to do so for all the users, we have to add another sum over all users:

$$L = \sum_{i:r_{i,j}=1}^n \sum_{i:r_{i,j}=1}^n (W_j \cdot X_i - y_{i,j})^2$$

## 2.3. The other way around

As you can notice, you could do the same in the other way around: suppose you have features corresponding to your users, you could train an algorithm that will learn the movie profiles (  $W_i$  parameters vector of movie i) that minimize the ratings predictions.

Ratings predictions are obtained with the dot product  $X_j \cdot W_i$  with  $X_j$  features vector of user j.

## 2.4. Pros and cons

Pros:

- No cold-start problem: no prior data needed to run a content based filtering
- Robust to popularity bias: no matter how many people watch a product, it does not get over recommended to other users

Cons:

- Can only recommend items with features close to already watched items: might end up recommending low rating items

# 3. Collaborative Filtering

There are two approaches to collaborative filtering, one based on users, the other on items.

The key idea behind collaborative filterting is that **similar users share the same interest** and that **similar items** (in terms of ratings/interactions) will be liked by a user.

In both approachs, we compute a **similarity score** (we will see how) between each pair of users (for user-based approach) or between each pair of items (for item-based approach).

## 3.1. Similarity

For computing similarity (either between users or between items) we can either use the **cosine similarity** or the **Pearson similarity**. Both are just distances metrics.

The formulas below define how we can compute the similarity between two vectors of attributes A and B:

- **Cosine similarity**: it corresponds to the dot product of A and B divided by the norms of A and B
$$\text{similarity}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$
- **Pearson similarity**: it corresponds to the covariance of A and B divided by the variance of A times the variance of B

$\rho_{A,B} = \frac{\sigma_A \sigma_B \text{Cov}(A,B)}{\sigma_A \sigma_B}$

For example, what is the cosine similarity between user 1 and user 3 in this example? Between 2 and 3?

			
	5		3
		4	
		4	3
	5	1	3

$\text{similarity}(\text{user1}, \text{user3}) = \frac{5 \times 3 + 5 \times 3 + 3 \times 3}{\sqrt{5^2 + 5^2 + 3^2} \sqrt{3^2 + 3^2 + 3^2}} = 1$

$\text{similarity}(\text{user2}, \text{user3}) = \frac{4 \times 3 + 1 \times 3 + 4 \times 3}{\sqrt{4^2 + 1^2 + 4^2} \sqrt{3^2 + 3^2 + 3^2}} = 0.85$

### 3.2. User-based

User based algorithms measure the **similarity between target users and other users**. In fact, you will choose the **k** most similar users (one could say the k nearest neighbors), and predict ratings for movies your user haven't seen yet.

Once we have computed the similarity between users we can retrieve the predicted rating of movie i for user j using the following formula:
$$\hat{y}_{i,j} = \frac{\sum_{k=1}^k \text{similarity}(u_j, u_k) \times y_{i,k}}{\sum_{k=1}^k \text{similarity}(u_j, u_k)}$$

### 3.3. Item-based

Item-based algorithms **measure the similarity between the items that target users rates/interacts with and other items**.

Analogously, we can compute the predictions by replacing  $\text{similarity}(u_j, u_k)$  by  $\text{similarity}(\text{item}_j, \text{item}_k)$  in the formula above.

### 3.4. Pros and cons

Pros:

- Data is self generated, each time a user watches a movie or interacts with an item
- Collaborative filtering may recommend items out the usual ones a user watches

Cons:

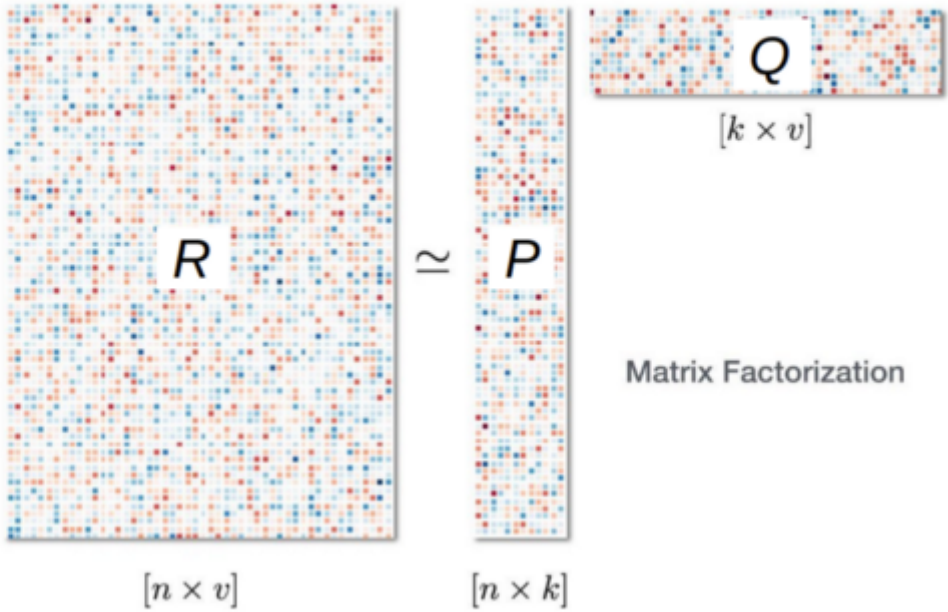
- Collaborative filtering has a major drawback: the cold-start problem. How to begin if you have no data of interactions with your items?
- Collaborative filtering would typically not recommends niche or new items, because there is no prior interaction with such items

### 3.5. [Optional] Matrix factorization

Matrix factorization solves the above problems by reducing the number of parameters to learn, and by fitting these parameters to the data that do exist.

What do we mean by factorization? It is simply finding two smaller matrices P and Q such that the ratings matrix equals to  $P \cdot Q$  (or approximates it)

#### Matrix factorization



This reduces greatly the number of parameters!

The size of those matrices are:

- $(n_u, k)$  for the matrix of users P (sometimes also called U)
- $(k, n_m)$  for the matrix of movies Q (sometimes also called V)

with k a hyperparameter called **number of components**.

Each line of P corresponds to a **user latent vector** (a representation of the user into a  $k$ -dimensional space). Each column of Q corresponds to a **item latent vector** (a representation of the item into a  $k$ -dimensional space).

**Resources:** [Great walkthrough of matrix factorization process](https://lazyprogrammer.me/tutorial-on-collaborative-filtering-and-matrix-factorization-in-python/) [\(https://lazyprogrammer.me/tutorial-on-collaborative-filtering-and-matrix-factorization-in-python/\)](https://lazyprogrammer.me/tutorial-on-collaborative-filtering-and-matrix-factorization-in-python/). In order to learn the representation of the users and of the items (the content of the matrices P and Q) you can proceed with gradient descent. Check the resource above if you want to dig deeper.

## 4. Hybrid System (collaborative + content-based)

### 4.1. LightFM introduction

Top notch tech companies use **hybrid systems** that combine both filtering methods (collaborative + content-based).

It provides more accurate recommendations by balancing both user behavior and attribute preferences.

Presenting... **LightFM**! Hybrid recommendation made easy





Assuming `train` is a (no\_users, no\_items) sparse matrix (with 1s denoting positive, and -1s negative interactions).

LightFM reduces to a traditional **collaborative filtering matrix factorization method**.

```
from lightfm import LightFM

model = LightFM(no_components=30)

model.fit(train)
```

**[BONUS]:** User and item features can be incorporated into training by passing them into the fit method.

Assuming `user_features` is a (no\_users, no\_user\_features) sparse matrix (and similarly for `item_features`), you can call:

```
model = LightFM(no_components=30)

model.fit(train,
          user_features=user_features,
          item_features=item_features,
          epochs=20)
```

## 4.2. Upcoming challenge: Netflix Recommendation



Download the **small** open-source MovieLens dataset: <https://grouplens.org/datasets/movielens/> [\(https://grouplens.org/datasets/movielens/\)](https://grouplens.org/datasets/movielens/)

It consists of **100,000 ratings** of **9,000 movies** by **600 users**.

Users rate movies by putting a **score between 1 and 5**. Each user has rated at least 20 movies.

Your mission is to build a **recommendation engine** that will recommend movies to users based on their likes.