

# SWE404/DMT413

## BIG DATA ANALYTICS

Lecture 11: Recommender Systems & Collaborative Filtering

Lecturer: Dr. Yang Lu

Email: [luyang@xmu.edu.my](mailto:luyang@xmu.edu.my)

Office: A1-432

Office hour: 2pm-4pm Mon & Thur

# Outlines

- Recommender Systems
- Content-based Methods
- Collaborative Filtering
- Related Issues



# RECOMMENDER SYSTEMS

# Recommender Systems

- During the last few decades, with the rise of YouTube, Amazon, Netflix and many other such web services, recommender systems have played an important role in our lives.
- From e-commerce, to entertainment, to online advertisement, recommender systems are today unavoidable in our daily online journeys.
  - We can't even escape from it today.

# Recommender Systems

- In a very general way, recommender systems are nothing but *algorithms* and *data*, aimed at suggesting relevant items to users.
  - The word “item” here is an abstraction, which can be movies to watch, text to read, products to buy, advertisements to click, or anything else that a user will interact with.
- Recommender systems are really critical in some industries as they can generate a huge amount of income when they are accurate.
  - Now, it becomes a way to stand out significantly from competitors.

# Amazon

- 35% of Amazon.com's revenue is generated by its recommendation engine.
- It is based on the analysis and modeling of a large number of user behaviors.

## Frequently Bought Together

Total price: **\$83.09**

[Add both to Cart](#)

[Add both to List](#)

This item: Structure and Interpretation of Computer Programs - 2nd Edition (MIT Electrical Engineering and... by Harold Abelson Paperback **\$50.50**

The Pragmatic Programmer: From Journeyman to Master by Andrew Hunt Paperback **\$32.59**

## Customers Who Bought This Item Also Bought

Page 1 of 13

Book Title	Author	Format	Price	Prime
The Little Schemer - 4th Edition	Daniel P. Friedman	Paperback	\$36.00	✓
Instructor's Manual I/a Structure and Interpretation of Computer Programs...	Gerald Jay Sussman	Paperback	\$28.70	✓
The Pragmatic Programmer: From Journeyman to Master	Andrew Hunt	Paperback	\$32.59	✓
Introduction to Algorithms, 3rd Edition (MIT Press)	Thomas H. Cormen	Hardcover	\$66.32	✓
An Introduction to Functional Programming Through Lambda Calculus	Greg Michaelson	Paperback	\$20.70	✓
Purely Functional Data Structures	Chris Okasaki	Paperback	\$40.74	✓
Code: The Hidden Language of Computer Hardware and Software	Charles Petzold	Paperback	\$17.99	✓
The Little Prover (MIT Press)	Daniel P. Friedman	Paperback	\$31.78	✓

# ByteDance

- ByteDance's operating revenue has shown exponential growth in the past years:
  - \$900 million in 2016;
  - \$2.4 billion in 2017;
  - \$7.4 billion in 2018;
  - \$17 billion in 2019.
- Recommender system is one of the most important techniques adopted for its products.





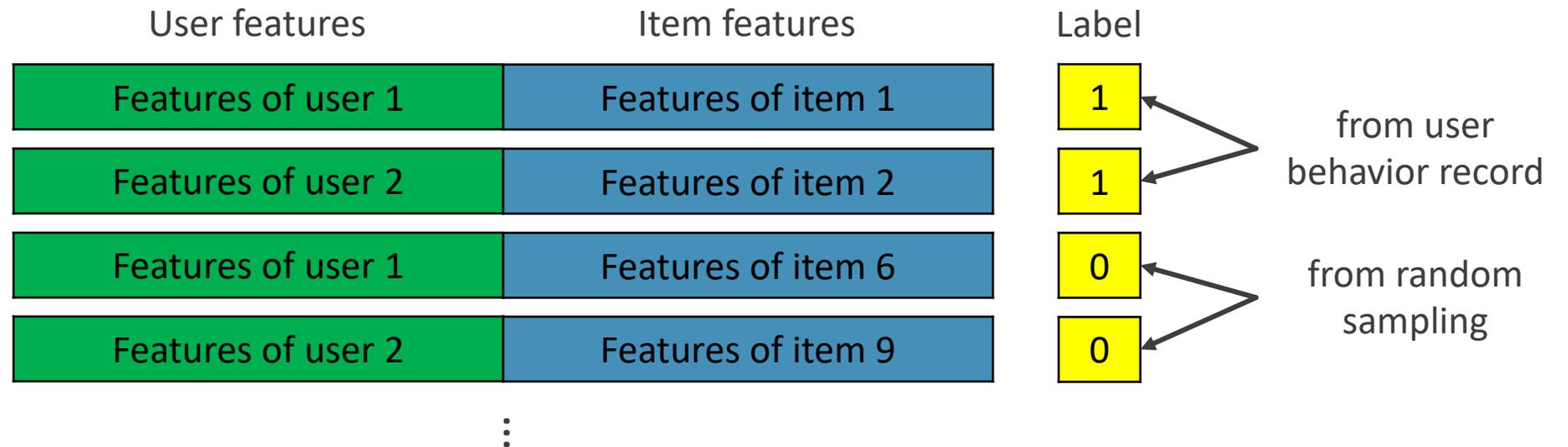
# CONTENT-BASED METHODS

# Content-Based Methods

- Content-based methods first extract user features and item feature, and then build classification or regression models on them.
- Take movie recommendation as an example:
  - User features may include age, gender, occupation, income, country, speaking language, etc.
  - Item features may include category, cast, rating, duration, language, company, director, etc.
- Some feature correlations can be captured by the model:
  - Teenagers tend to watch Marvel movies / young ladies tend to watch love movies.
  - People tend to watch the movie whose language is same as the speaking language.

# Content-Based Methods

- As we usually do for supervised learning, we can construct training and test datasets:
  - For example, user 1 clicked item 1, user 2 clicked item 2.
- Recommend by ranking the predicted probabilities of all unclicked items for a given user.

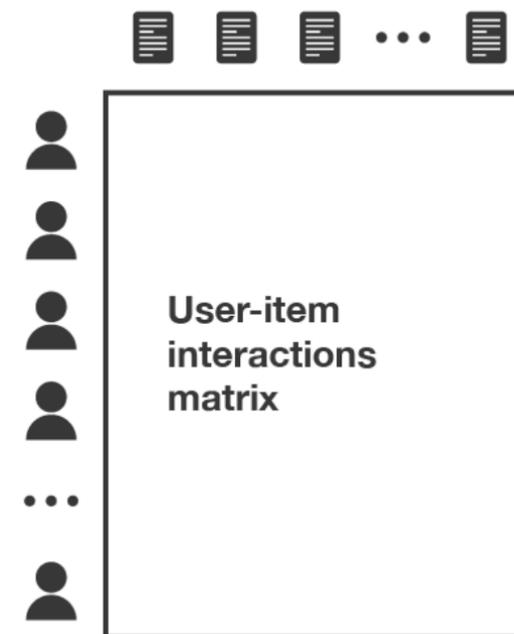




# COLLABORATIVE FILTERING

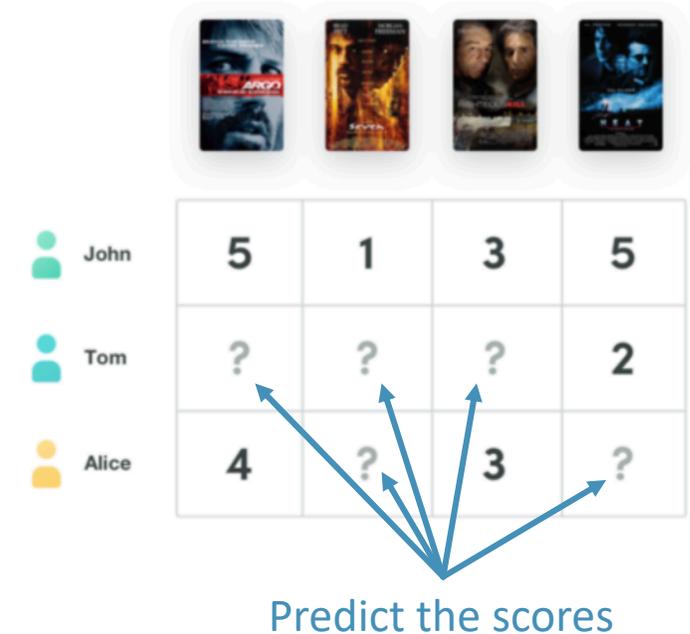
# Collaborative Filtering

- Collaborative methods for recommender systems are methods that are based **solely** on the past interactions recorded between users and items in order to produce new recommendations.
  - No user feature and item feature is needed.
- These interactions are stored in the so-called *user-item interaction matrix*.
- The values stored in the user-item interaction matrix can be:
  - 0/1 for product click or purchase.
  - Score or rating for product review.
  - 1/-1 for like/dislike.



# User-Item Interaction Matrix

- Now, the task is simply to fill in the blank in the user-item interaction matrix.
  - It becomes a matrix completion problem, i.e. estimate the value of missing entries for a matrix.
  - It is a general problem in mathematics and computer science and has numerous applications (recommendation is one of the most important applications).
- After that, sort the predicted value in the blank cell for each user and show the top ones to them.



# Collaborative Filtering

Generally, there are two types of collaborative filtering methods:

- **Memory-based methods (aka Neighborhood-based).**
  - No latent model is assumed.
  - The algorithms directly works with the user-item interaction matrix.
- **Model-based methods.**
  - Some latent interaction model is assumed.
  - A model is trained to reconstruct the missing values in user-item interaction matrix.

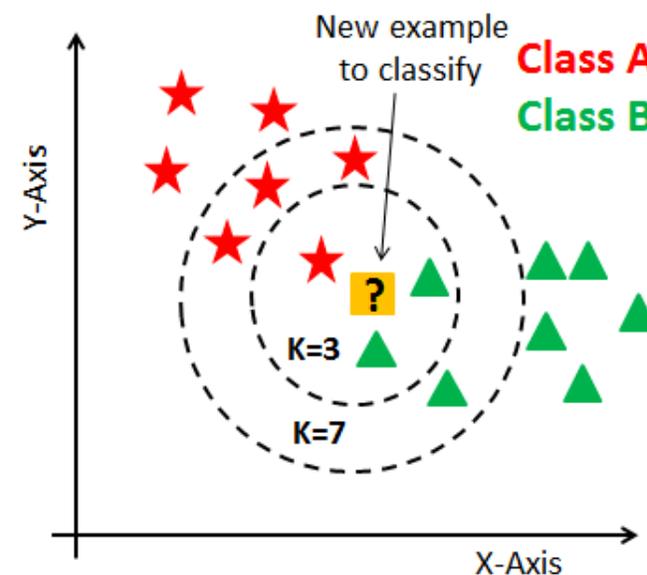
# Memory-Based Methods

Two dimensions can be explored:

- User-based methods: Given a user, search for similar users, recommend items that are interacted with similar users to the user.
- Item-based methods: Given a user, find the user's preferred items, search for similar items and recommend them to the user.

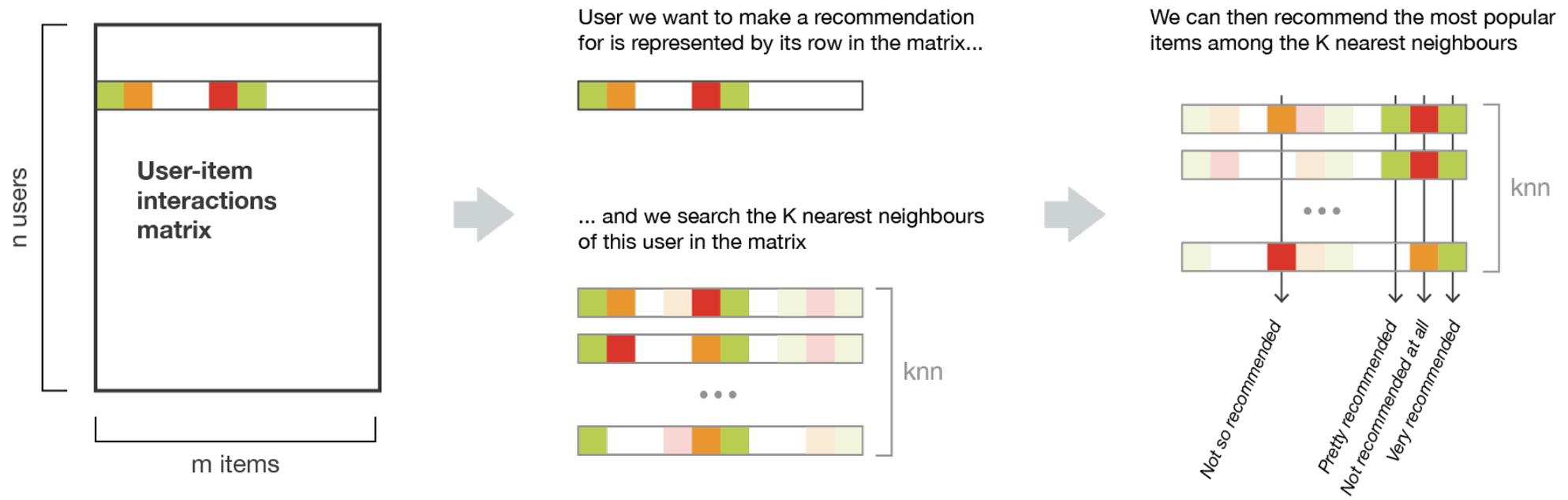
# $k$ -Nearest Neighbor

- $k$ -Nearest Neighbor ( $k$ NN) is a non-parametric and lazy learning algorithm.
  - Non-parametric means there is no assumption for underlying data distribution.
  - Lazy algorithm means it does not need to have a training process. Actually, it doesn't have a model.
- Given a test sample,  $k$ NN has the following basic steps:
  - Calculate distance to all training samples.
  - Find the nearest  $k$  neighbors.
  - Vote for labels (required only for classification).

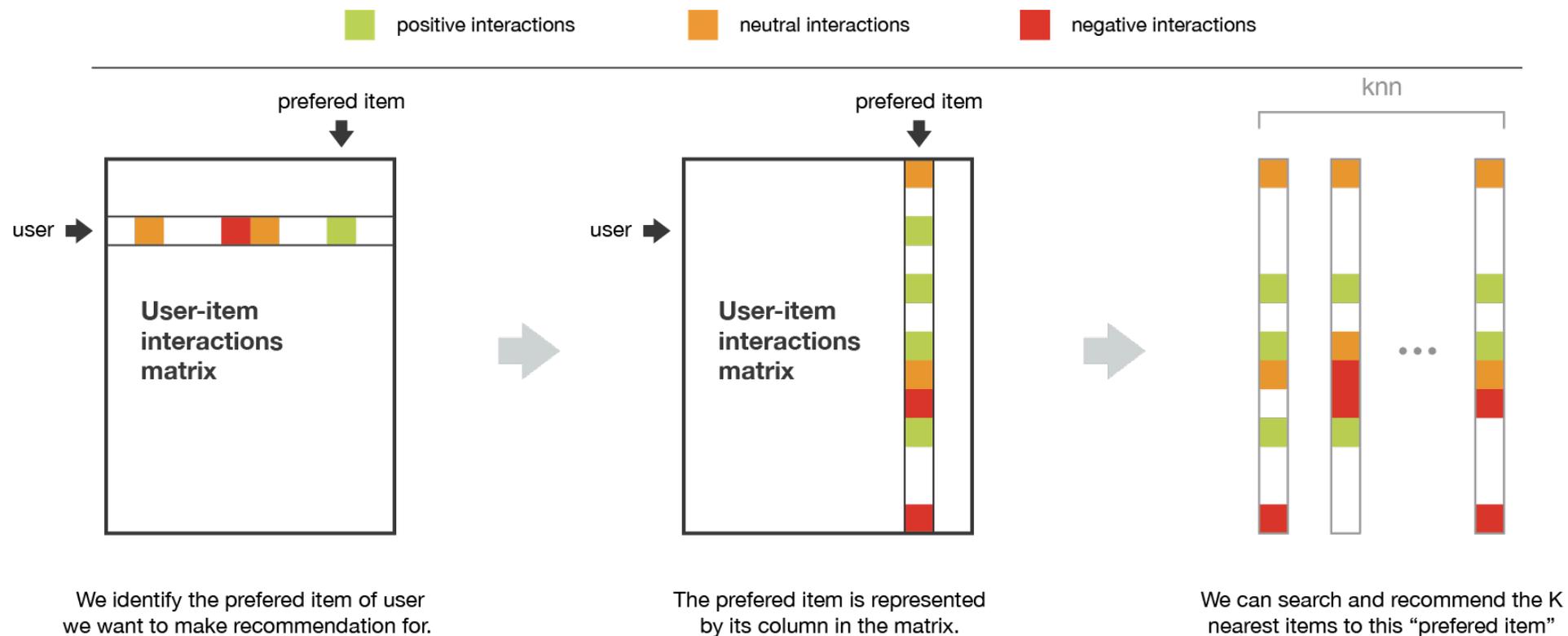


# User-Based Methods

■ positive interactions      ■ neutral interactions      ■ negative interactions



# Item-Based Methods



# User-Based Methods vs. Item-Based Methods

- The user-based methods are based on the search of similar users.
  - In general, every user have only interacted with a few items, it makes the method pretty sensitive to any recorded interactions (**high variance**).
  - On the other hand, as the final recommendation is only based on interactions recorded for users similar to our user of interest, we obtain more personalized results (**low bias**).
- The item-based methods are based on the search of similar items.
  - In general, a lot of users have interacted with an item, the neighborhood search is far less sensitive to single interactions (**low variance**).
  - As a counterpart, interactions coming from every kind of users are considered in the recommendation, making the method less personalized (**high bias**).
  - Thus, this approach is less personalized than the user-based approach but more robust.

# Complexity

- One of the biggest flaw of memory-based collaborative filtering is that they do not scale easily.
  - For systems with millions of users and millions of items, the nearest neighbors search step can become intractable.
  - That is also why Spark and MLlib do not support  $k$ NN, and thus do not support memory-based collaborative filtering.
    - You can think about whether you can implement  $k$ NN in Spark.

# Model-Based Methods

- Model-based methods assume a latent model supposed to explain these user-item interactions.
- A representative model-based method is called *matrix factorization*.
- The main assumption behind matrix factorization is that there exists a pretty **low dimensional latent space of features** in which we can represent both users and items.
- Then, the interaction between a user and an item can be obtained by computing the **dot product** of corresponding dense vectors in that space.

# Motivation of Matrix Factorization

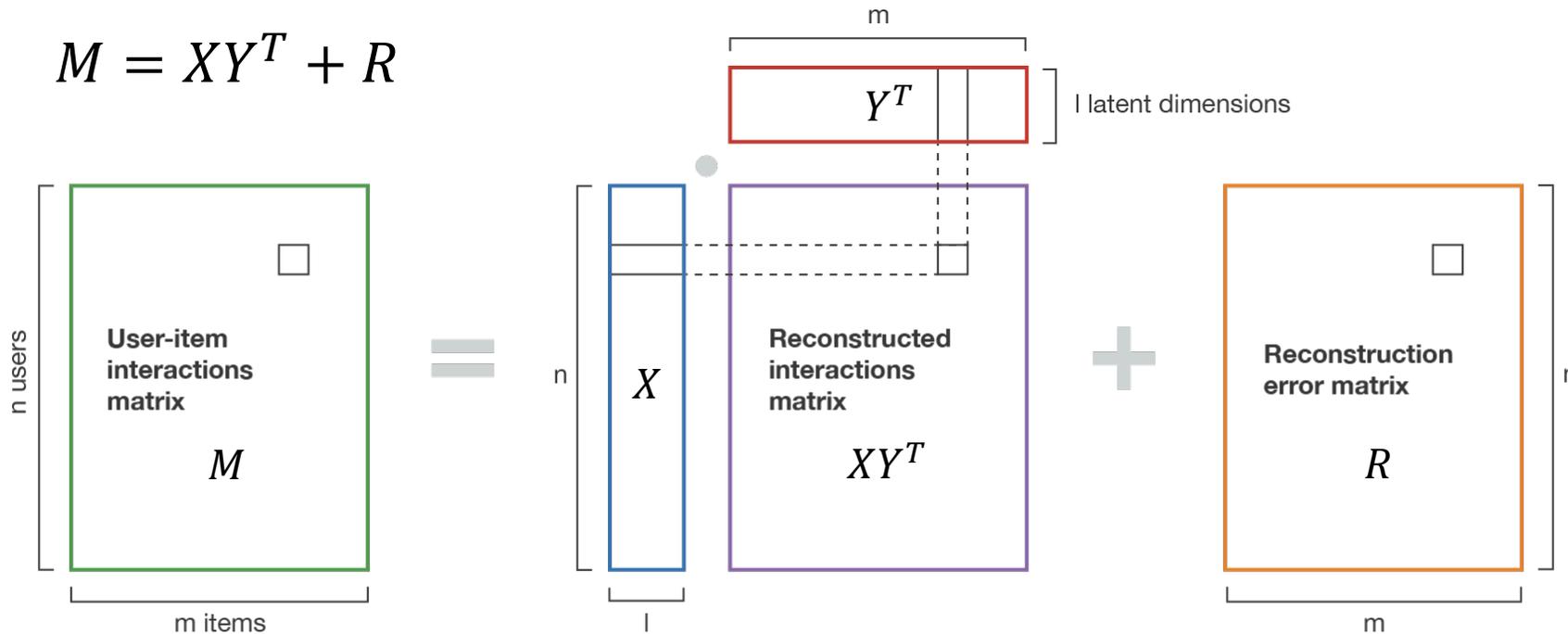
- For example, consider that we have a user-movie rating matrix. In order to model the interactions between users and movies, we can assume:
  - There exists some features describing pretty well movies.
  - These features can also be used to describe user preferences or popularity.
- However we don't want to give explicitly these features to our model like content-based methods.
- Instead, we prefer to let the system discover these useful features by itself and make its own representations of both users and items.
  - Design an algorithm to learn useful features.

# Motivation of Matrix Factorization

- These learned features are also called *embeddings*.
  - Each dimension in the embeddings is a real number.
- As they are learned and not given, embeddings have a mathematical meaning for model prediction but no intuitive interpretation.
  - It is almost impossible know the meaning of each dimension of embeddings.
- Indeed, the consequence of such factorization is that similar users in terms of preferences as well as similar items in terms of characteristics ends up having close embedding representations in the latent space.

# Matrix Factorization

$$M = XY^T + R$$



The **user-item interactions matrix** is assumed to be equal to...

... the **dot product** of a **user matrix** and a **transposed item matrix**...

... plus some **reconstruction error**

# Mathematics of Matrix Factorization

- Let's consider an  $n \times m$  user-item interaction matrix  $M$  of ratings where only some items have been rated by each user.
  - Most of the entries in  $M$  are set to *None* to express the lack of rating.
- We want to factorize that matrix such that

$$M \approx XY^T.$$

- $X$  is an  $n \times l$  “user latent matrix” whose  $i$ th row  $X_i$  represents the  $i$ th user.
- $Y$  is an  $m \times l$  “item latent matrix” whose  $j$ th row  $Y_j$  represents the  $j$ th item.
- $l$  is the embedding dimension in the latent space in which users and item will be represented.

# Mathematics of Matrix Factorization

- So, we search for matrices  $X$  and  $Y$  whose matrix product best approximates the existing interactions in  $M$ .
- Denoting  $E$  the set of pairs  $(i, j)$  such that  $M_{ij}$  is not *None*, we want to find  $X$  and  $Y$  that minimize the *reconstruction error*:

$$(X, Y) = \operatorname{argmin}_{X, Y} \sum_{(i, j) \in E} (X_i Y_j^T - M_{ij})^2.$$

- To prevent overfitting, we can limit the complexity of the latent matrices with a regularization parameter:

$$(X, Y) = \operatorname{argmin}_{X, Y} \sum_{(i, j) \in E} (X_i Y_j^T - M_{ij})^2 + \lambda (\|X\|_F + \|Y\|_F)$$

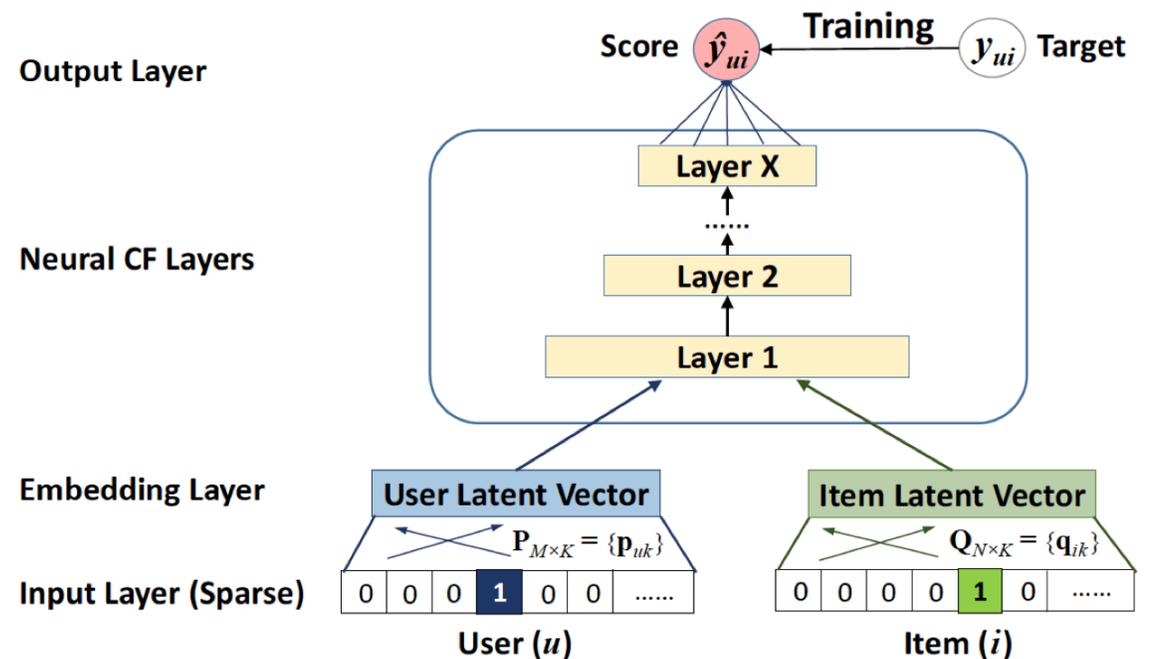
Frobenius norm

# Mathematics of Matrix Factorization

- The matrices  $X$  and  $Y$  can then be obtained following a gradient descent optimization process.
  - In each iteration, we consider only a subset of the pairs in  $E$ , because we can't fill all data in memory.
  - The gradient descent can be done alternatively on  $X$  and  $Y$  at each step (in each iteration, fix one and update another).
- Once the matrix has been factorized, to make a new recommendation is to simply multiply a user vector by any item vector and recommend the items with highest scores.
- Notice that we could also use memory-based methods with these new representations of users and items, because the burden of  $k$ NN search is relieved by low dimensional representation.

# Beyond Matrix Factorization

- Notice that, our goal is to learn latent representation of users and items.
  - Matrix factorization is only one solution.
- We can use neural networks to accomplish the goal.



## Explicit vs. Implicit Feedback

- The standard approach to matrix factorization-based collaborative filtering treats the entries in the user-item interaction matrix as *explicit* preferences given by the user to the item, for example, users giving ratings to movies.
  - However, the ratings may not be available for all recommendation scenarios.
- It is common in many real-world use cases to only have access to *implicit* feedback (e.g. view duration, number of clicks, purchases, likes, shares, etc.).
  - Those numbers are then related to *the level of confidence* in observed user preferences. It can somehow replace explicit ratings given to items.

# Cold Start Problem

- As it only consider past interactions to make recommendations, collaborative filtering suffer from the *cold start problem*.
- It is impossible to recommend anything to new users or to recommend a new item to any users.
  - Initially, there's no interaction for a new user and a new item.
- This drawback can be addressed in different way:
  - **Random strategy**: recommending random items to new users or new items to random users.
  - **Maximum expectation strategy**: recommending popular items to new users or new items to most active users.
  - **Exploratory strategy**: recommending a set of various items to new users or a new item to a set of various users.
  - **Hybrid strategy**: using content-based method for the early stage of the user or the item.

# Content-Based Methods vs. Collaborative Filtering

	Content-Based Methods	Collaborative Filtering
Advantages	<ul style="list-style-type: none"><li>• Don't need any data about other users.</li><li>• Interpretable results.</li></ul>	<ul style="list-style-type: none"><li>• Don't need domain knowledge because the embeddings are automatically learned.</li><li>• Can help users discover new interests.</li></ul>
Disadvantages	<ul style="list-style-type: none"><li>• Highly depend on feature engineering and domain knowledge.</li><li>• Can only make recommendations based on existing interests of the user.</li></ul>	<ul style="list-style-type: none"><li>• Cold start problem.</li><li>• Useful and informative features can't be used.</li></ul>

# Mlib API

- Mlib currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries.
  - Specifically, a matrix factorization method called the alternating least squares (ALS) algorithm is used.
  - You are suggested to read the following papers:
    - Koren, Y., Bell, R. and Volinsky, C., 2009. Matrix factorization techniques for recommender systems. *Computer*, 42(8), pp.30-37.
    - Zhou, Y., Wilkinson, D., Schreiber, R. and Pan, R., 2008, June. Large-scale parallel collaborative filtering for the netflix prize. In *International conference on algorithmic applications in management* (pp. 337-348). Springer, Berlin, Heidelberg.

# Mlib API

```
class pyspark.mllib.recommendation.ALS \[source\]
```

Alternating Least Squares matrix factorization

*New in version 0.9.0.*

```
classmethod train(ratings, rank, iterations=5, lambda_=0.01, blocks=-1, nonnegative=False, seed=None) \[source\]
```

- Train a matrix factorization model given an RDD of ratings by users for a subset of products.
- The ratings matrix is approximated as the product of two lower-rank matrices of a given rank (number of features).
- Input and parameters:
  - **ratings** – RDD of *Rating* or (userID, productID, rating) tuple.
  - **rank** – Number of features to use (also referred to as the number of latent factors or embedding dimension).
  - **iterations** – Number of iterations of ALS. (default: 5)
  - **lambda** – Regularization parameter. (default: 0.01)
  - **blocks** – Number of blocks used to parallelize the computation. A value of -1 will use an auto-configured number of blocks. (default: -1)
  - **nonnegative** – A value of True will solve least-squares with nonnegativity constraints. (default: False)

# Mlib Example with Explicit Feedback

- Training data:

1	1,1,5.0
2	1,2,1.0
3	1,3,5.0
4	2,1,1.0
5	2,2,1.0
6	2,3,5.0
7	3,1,1.0
8	3,3,5.0
9	3,4,1.0
10	4,1,5.0
11	4,2,1.0
12	4,4,5.0

- Test data:

1	1,4,5.0
2	2,4,1.0
3	3,2,1.0
4	4,3,5.0

- User 1 and 4 are similar: like item 1, 3, 4 and dislike item 2.
- User 2 and 3 are similar: like item 3 and dislike item 1, 2, 4.

# MLlib Example

```
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating

# Load and parse the data
train_data = sc.textFile("train.data")
train_ratings = train_data.map(lambda l: l.split(',')\
    .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))
train_interaction = train_ratings.map(lambda p: (p[0], p[1]))

test_data = sc.textFile("test.data")
test_ratings = test_data.map(lambda l: l.split(',')\
    .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))
test_interaction = test_ratings.map(lambda p: (p[0], p[1]))

# Build the recommendation model using Alternating Least Squares
rank = 10
numIterations = 10
model = ALS.train(train_ratings, rank, numIterations)

# Evaluate the model on training data
train_predictions = model.predictAll(train_interaction).map(lambda r: ((r[0], r[1]), r[2]))
train_ratesAndPreds = train_ratings.map(lambda r: ((r[0], r[1]), r[2])).join(train_predictions)
train_MSE = train_ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
print("Training Mean Squared Error = " + str(train_MSE))

test_predictions = model.predictAll(test_interaction).map(lambda r: ((r[0], r[1]), r[2]))
test_ratesAndPreds = test_ratings.map(lambda r: ((r[0], r[1]), r[2])).join(test_predictions)
test_MSE = test_ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
print("Test Mean Squared Error = " + str(test_MSE))

Training Mean Squared Error = 6.58432763311764e-05
Test Mean Squared Error = 0.4504563601333942
```

```
train_ratesAndPreds.collect()
```

```
[((3, 3), (5.0, 4.995795701456707)),
 ((4, 2), (1.0, 1.000255165669839)),
 ((1, 2), (1.0, 0.9997298959341163)),
 ((1, 3), (5.0, 5.000961796554121)),
 ((2, 1), (1.0, 1.0045528987014631)),
 ((2, 3), (5.0, 5.000044785316618)),
 ((1, 1), (5.0, 4.993413981545112)),
 ((2, 2), (1.0, 0.974965734431291)),
 ((3, 1), (1.0, 1.0052396373395827)),
 ((4, 4), (5.0, 4.992929888213817)),
 ((3, 4), (1.0, 0.9982449988413988)),
 ((4, 1), (5.0, 5.000196921930447))]
```

```
test_ratesAndPreds.collect()
```

```
[((1, 4), (5.0, 4.0918731072496275)),
 ((2, 4), (1.0, 0.34882229396288067)),
 ((3, 2), (1.0, 0.8456589518526052)),
 ((4, 3), (5.0, 5.727514551891664))]
```

# MLlib Example

```
model.recommendProductsForUsers(4).collect()
```

```
[(1,
  (Rating(user=1, product=3, rating=4.99495894435074),
    Rating(user=1, product=1, rating=4.9936157901529254),
    Rating(user=1, product=4, rating=1.8202134052011605),
    Rating(user=1, product=2, rating=1.0287746326427167))),
 (2,
  (Rating(user=2, product=3, rating=4.997937494270436),
    Rating(user=2, product=1, rating=1.0045636511354246),
    Rating(user=2, product=2, rating=0.984679588989948),
    Rating(user=2, product=4, rating=-0.08847966689259223))),
 (3,
  (Rating(user=3, product=3, rating=4.993134227993103),
    Rating(user=3, product=2, rating=1.1597402602367057),
    Rating(user=3, product=1, rating=1.010110716722662),
    Rating(user=3, product=4, rating=0.9920657708069707))),
 (4,
  (Rating(user=4, product=1, rating=4.999351567672768),
    Rating(user=4, product=4, rating=4.991966192679747),
    Rating(user=4, product=3, rating=2.4454701827843586),
    Rating(user=4, product=2, rating=1.0023917204206207)))]
```

```
model.recommendUsersForProducts(4).collect()
```

```
[(1,
  (Rating(user=4, product=1, rating=4.999351567672768),
    Rating(user=1, product=1, rating=4.9936157901529254),
    Rating(user=3, product=1, rating=1.010110716722662),
    Rating(user=2, product=1, rating=1.0045636511354246))),
 (2,
  (Rating(user=3, product=2, rating=1.1597402602367057),
    Rating(user=1, product=2, rating=1.0287746326427167),
    Rating(user=4, product=2, rating=1.0023917204206207),
    Rating(user=2, product=2, rating=0.984679588989948))),
 (3,
  (Rating(user=2, product=3, rating=4.997937494270436),
    Rating(user=1, product=3, rating=4.99495894435074),
    Rating(user=3, product=3, rating=4.993134227993103),
    Rating(user=4, product=3, rating=2.4454701827843586))),
 (4,
  (Rating(user=4, product=4, rating=4.991966192679747),
    Rating(user=1, product=4, rating=1.8202134052011605),
    Rating(user=3, product=4, rating=0.9920657708069707),
    Rating(user=2, product=4, rating=-0.08847966689259223)))]
```



## RELATED ISSUES

# Evaluation of a Recommender System

- If the model generates real numeric values such as ratings predictions or matching probabilities, we can evaluate it as a regression problem using an error measurement metric such as MSE.
- If the model generates binary predictions, we can then evaluate the accuracy (as well as precision, recall, F1 score, AUC).
- In this case, the model is trained only on a part of the available interactions and is tested on the remaining ones.

# Click Through Rate

- When we launch our recommender system online, the evaluation becomes different.
- We only show the items with positive predictions to users. Therefore,
  - there will never be false positive (users have no chance to click the negative predictions);
  - the user interaction is not accurate (sometimes miss the item, rather than be not interested).
- In this case, we usually call a user-item interaction as a *click*, and a item that the user has seen as an *impression*.
- *Click Through Rate (CTR)* is usually defined for online evaluation. It is the number of clicks that your items receives divided by the number of items is shown:

$$CTR = clicks / impressions.$$

- For example, if you had 5 clicks and 100 impressions, then your CTR would be 5%.

# Recommender Systems with Different Scenarios

- For movies, books, and news, you may be interested in similar ones.
- However, for some products, you don't need similar ones.
  - E.g. you won't buy similar laptop computers after you buy one from amazon. Instead, you may need keyboard, mouse, headphone and related equipments.
- Therefore, in different scenarios, the focus of a recommender system is different.
  - It is not that easy to simply apply a general recommender system to all scenarios.
  - Analysis and discussion should be made before implementing technical details.
    - Usually done by data analysts and product managers.

# Diversity of a Recommender System

- A direct consequence of recommender system is that you will fall into the small world of your interest.
  - E.g. once you clicked one or two times the news about Donald Trump, all the news shown to you is about Donald Trump, the White House, Republicans, US-China relationship...
  - Someday you will feel bored.
- A good recommender system should mix your interests and something else for exploration and discovery.
  - Diversity is crucial for preventing customer churn.
  - However, this trade-off is hard to handle. It is also called exploration-exploitation dilemma.

# Tutorial

- Try this example as tutorial. Run it by yourself and upload to Attendance Quiz.
  - <https://www.kaggle.com/vchulski/tutorial-collaborative-filtering-with-pyspark>.

# Conclusion

After this lecture, you should know:

- What is a recommender system.
- What are the differences between content-based methods and collaborative filtering.
- What is a user-item interaction matrix.
- What are the differences between memory-based and model-based methods.
- How does matrix factorization work.

# Thank you!

## ■ Reference:

- Tutorial in towardsdatascience.com: <https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada>
- MLlib official site: <https://spark.apache.org/docs/latest/ml-collaborative-filtering.html>
- Koren, Y., Bell, R. and Volinsky, C., 2009. Matrix factorization techniques for recommender systems. *Computer*, 42(8), pp.30-37.