Introduction to recommendation Systems

Significant industrial & research interest in recommendation systems:

NETFLIX 70-80% from recommendations

YouTube increases its watch times by 60% - 70% per year

amazon 60% - 70% of all sales are generated by recommendations

additional statistics on the use of recommendation systems

- 71% of e-commerce sites offer product recommendations.
- Personalized product recommendations account for more than 35% of purchases on Amazon.
- Product recommendations can account for up to 31% of e-commerce revenues.
- 60% of the clicks on YouTube's home screen are on recommendations.

Dramatic increase of content-delivery services and multimedia content

Wide-range of choices and tight time constraints

Efficient mechanisms required for discovering preferred content

☞ Improvement of user satisfaction, decrease in churn rates

Recommendation systems (RS) address these problems

- ✓ Exploit information about user preferences
- ✓ Accurately & efficiently recommend the appropriate content

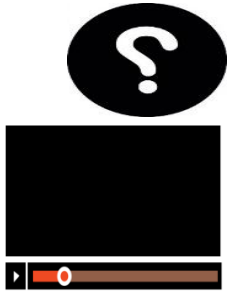
User feedback

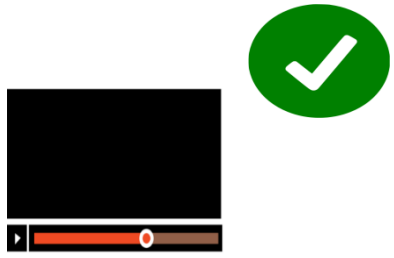
Explicit feedback: Users indicate their preference for items

- Ratings scales (1-5 stars, 1-10 stars)
- Binary values (liked/disliked)

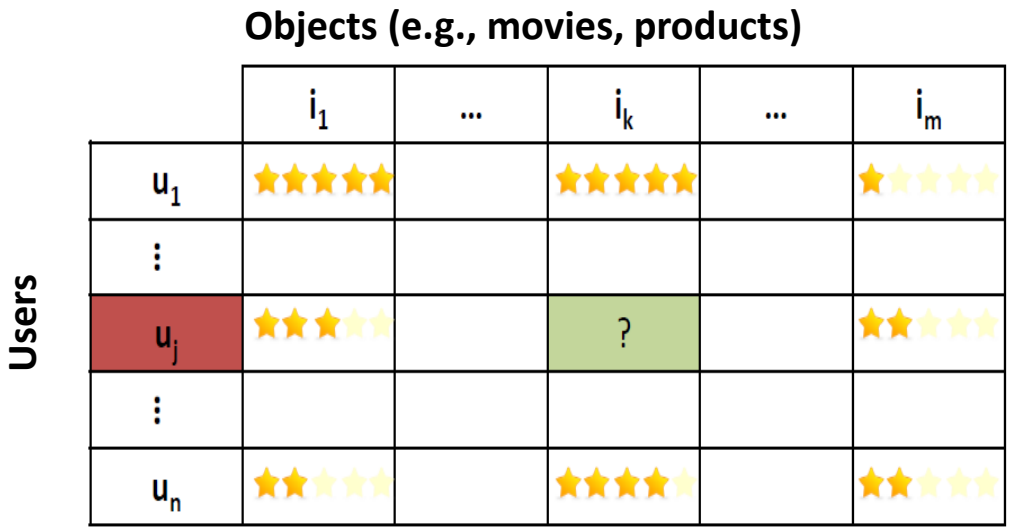
Implicit feedback: Users interact with items

- Purchase history
- Viewing duration
- Click patterns





Ratings matrix



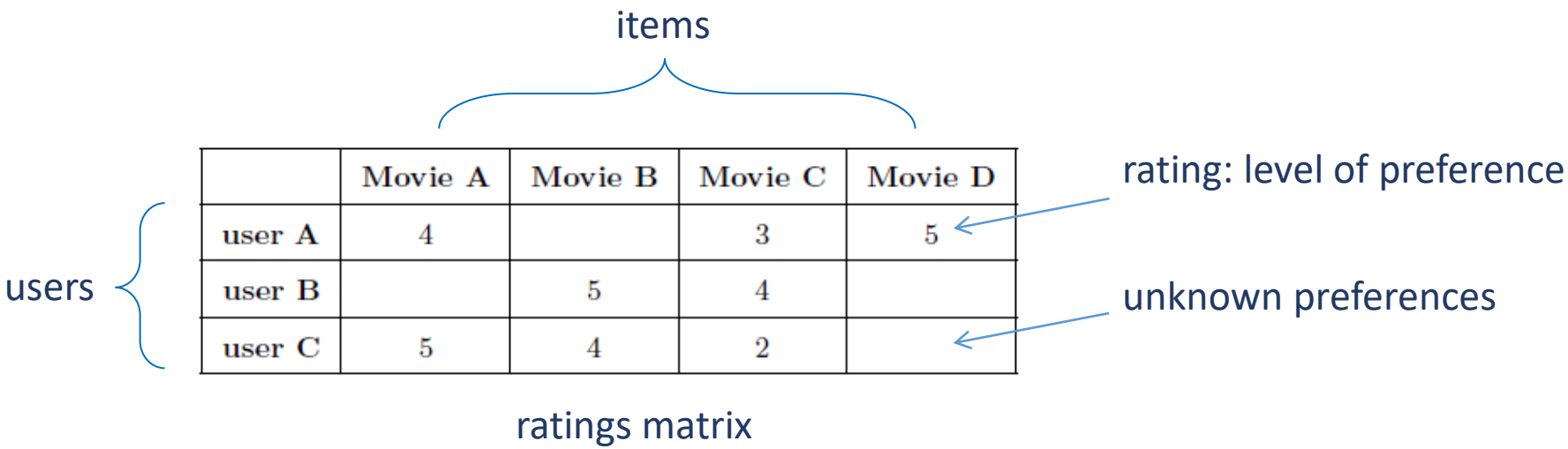
Sparse matrix: Most values are unknown

Predicting: The task of filling the unknown values

Recommendation Systems

Infer user preferences about items

Produce **highly relevant** & **personalized** lists of items



	Movie A	Movie B	Movie C	Movie D
user A	4		3	5
user B		5	4	
user C	5	4	2	

ratings matrix

Main Approaches

Collaborative filtering (CF)

Inspect rating patters to find similar users/items

Content-based (CB)

Analyze attributes of items for building user profiles

In general, CF performs **better** than CB

- CF fail to provide accurate predictions with insufficient ratings
- CB can alleviate the sparsity problem

Popular Recommenders: K-Nearest Neighbors Approach

User-based collaborative filtering (UBCF)

Assumption: Similar users have similar preferences

- **User similarity: agreement on co-rated items**
- Prediction: **weighted sum of similar user's ratings**

Item-based collaborative filtering (IBCF)

Assumption: Users have similar tastes for similar items

- Item similarity: agreement within users rated both items
- Prediction: weighted sum of similar items' ratings

User-based collaborative filtering (UBCF)

Assumption: Similar users have similar preferences

- **User similarity: agreement on co-rated items**
- Prediction: **weighted sum of similar user's ratings**

of agreement between users. A "neighborhood" G is built for a user u, consisting of the set of users who are similar to u (also referred as *neighbors* of u). Once G has been determined, the prediction for the user u about the item i $p(u,i)$ is the weighted average over the ratings of his/her neighbors on this item, using as weights the computed similarities:

$$p_{u,i} = r_u + \frac{\sum_{v \in G(u)} sim(u,v)(r_{v,i} - r_v)}{\sum_{v \in G(u)} |sim(u,v)|}$$

Item-based collaborative filtering (IBCF)

Users have similar tastes for similar items

- Item similarity: **agreement within users rated both item**
- Prediction: weighted sum of similar items' ratings

computing similarities between items, instead of users. Two items are considered similar if the same users tend to rate them similarly. Finally, the expected rating of the user u for the movie i $p(u,i)$ is estimated as follows:

$$p_{u,i} = \frac{\sum_{j \in G(i)} sim(i,j)(r_{u,j})}{\sum_{j \in G(i)} |sim(i,j)|}$$

IBCF

Compute similarity between items

set of users rated both items

rating on item *i*

rating on item j

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_u)(R_{u,j} - \bar{R}_u)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_u)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_u)^2}}$$

deviation from the average rating

Prediction of rating for item *i*

most similar items to *i*

similarity as weight

rating on similar item

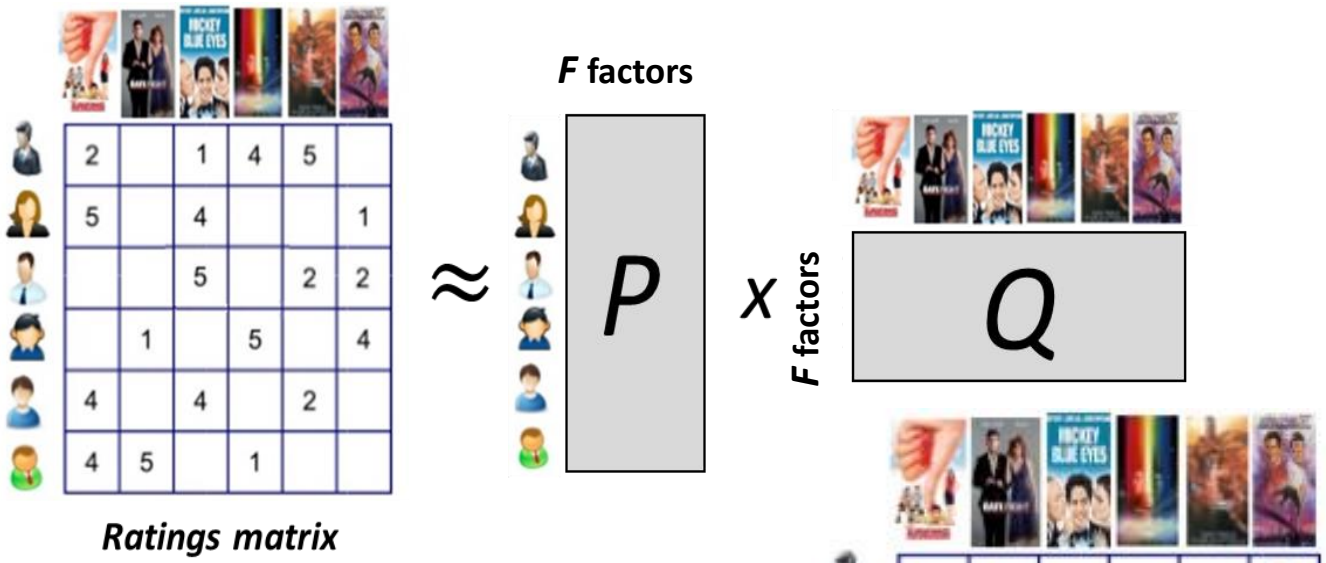
$$p_{u,i} = \frac{\sum_{j \in S} sim(i,j) R_{u,j}}{\sum_{j \in S} |sim(i,j)|}$$

SVD: Matrix Factorization Approach

Decomposition of ratings matrix ***R*** into the product of ***P*** & ***Q***

Each user & item is described with ***F*** latent features

- ***P*** : user factors
- ***Q*** : item factors



Prediction for user u about item i :

$$p_{u,i} = q_i^T p_u = \sum_{f=1}^{F} q_{i,f} \cdot p_{u,f}$$

Matrix - Factorization

- **Factorize a matrix:** to find out two (or more) matrices such that when you multiply them you will get back the original matrix

- **Application perspective**: can be used to discover **latent features** underlying the interactions between two different kinds of entities

Latent Features or Factors

Λανθάνοντα χαρακτηριστικά ή παράγοντες

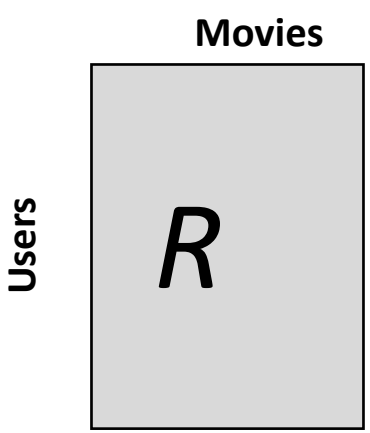
- Latent features in matrix factorization are the hidden patterns and relationships in the data that are not explicitly represented in the original data matrix.

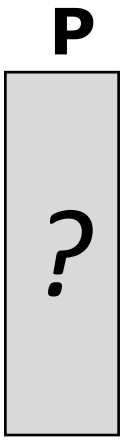
- Λανθάνοντα χαρακτηριστικά στην παραγοντοποίηση μήτρας είναι τα κρυφά μοτίβα και οι σχέσεις στα δεδομένα που δεν αντιπροσωπεύονται ρητά στον αρχικό πίνακα δεδομένων.

- Matrix factorization is a technique for decomposing a large and sparse matrix into two smaller, denser matrices that capture these latent features.
- Examples of latent features include:
  - In a movie recommender system: Latent features could represent genre, rating, popularity, or actors.
  - In a product recommendation system: Latent features could represent product category, brand, price, or features.
  - In a text analysis system: Latent features could represent topics, keywords, or sentiment.

Matrix - Factorization

P: *a |U| x K matrix*



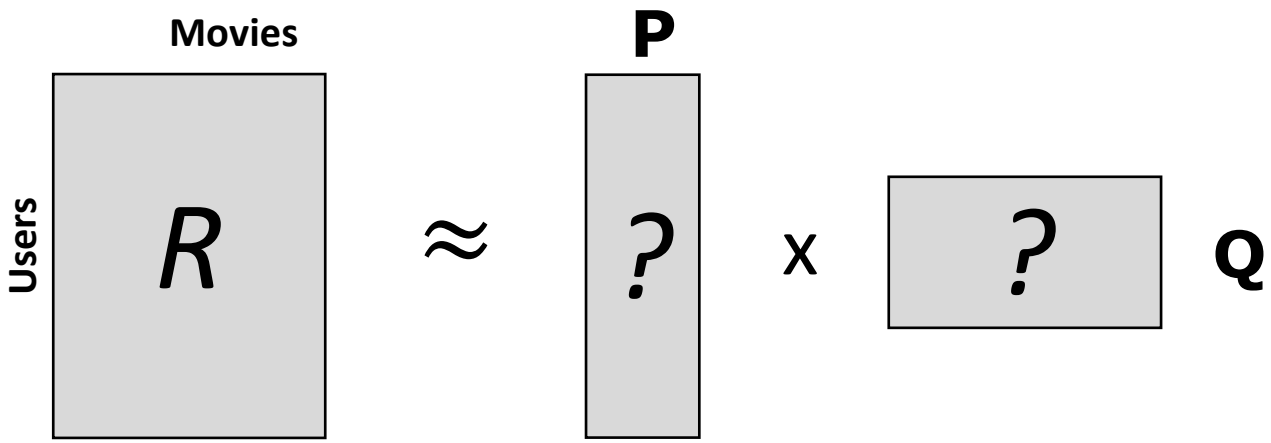


Q: *a |D| x K matrix*

R*: Matrix of size |U| x |D| that contains all the ratings that the users have assigned to the items*

K: The number of factors – This is a parameter that user specifies

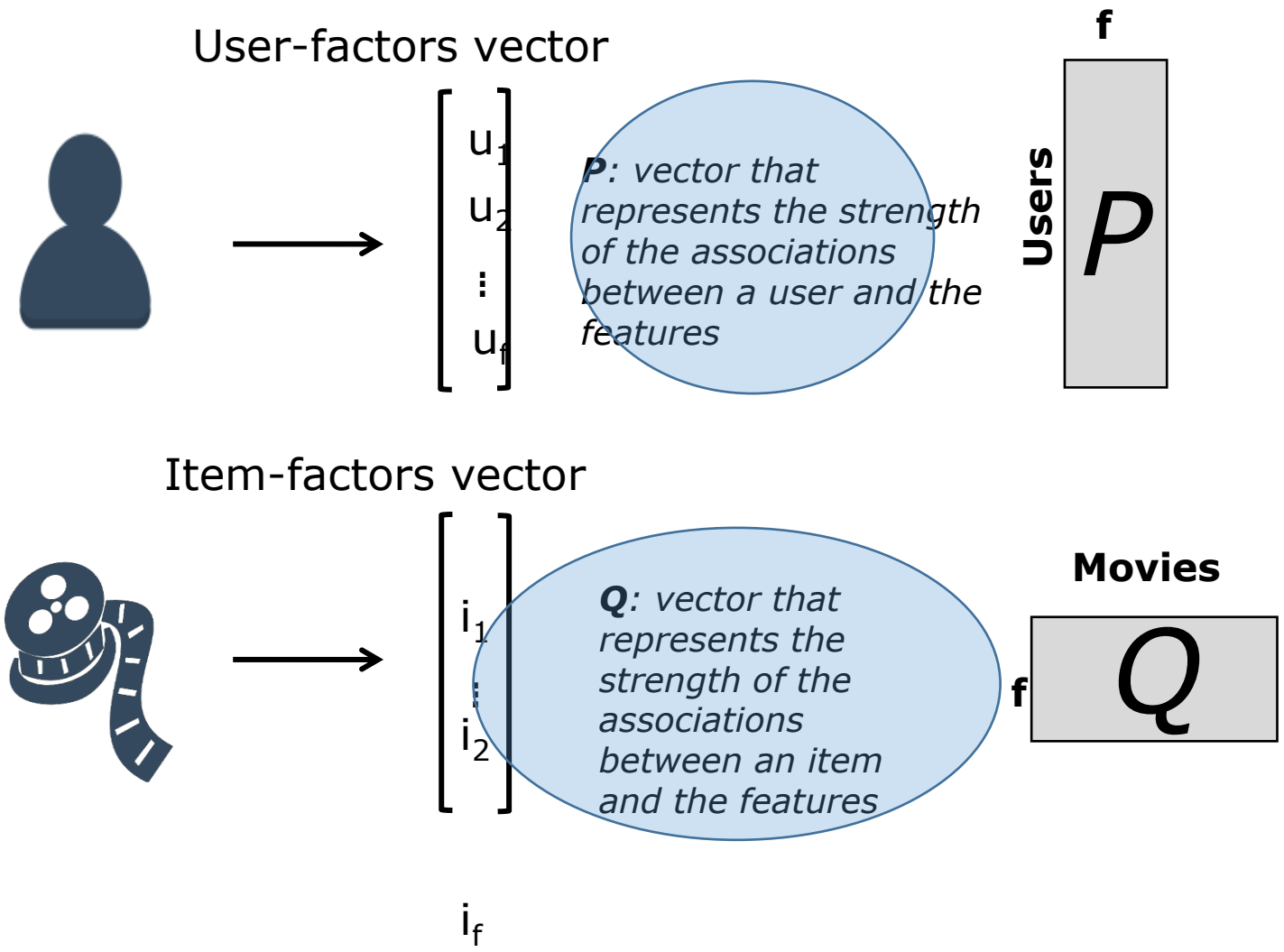
Matrix – Factorization: Ratings matrix is approximated



Find two matrices P and Q such that product approximates R

Matrix - Factorization:

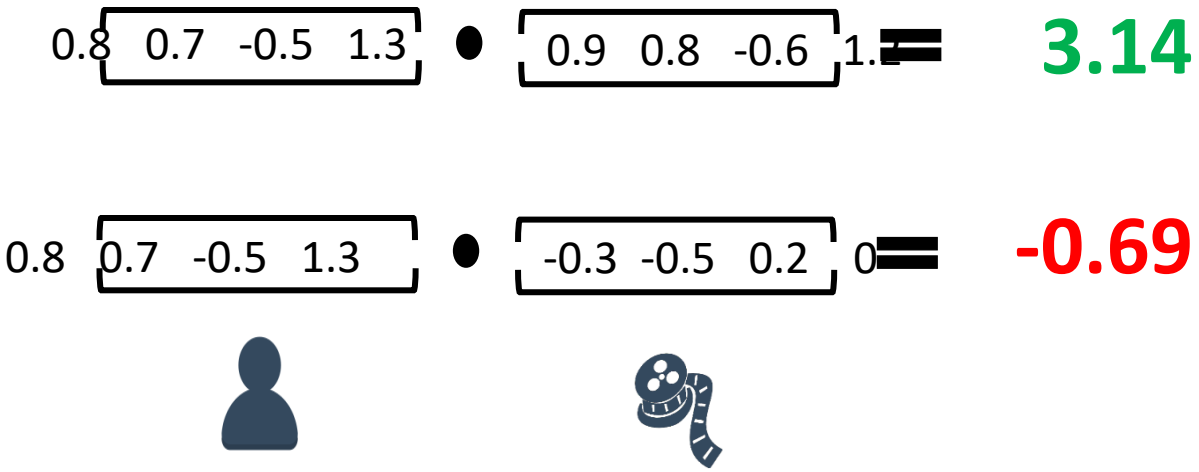
Each user & item is characterized with a vector of factors



Factors - *f*: mathematic artifact for computing relationship

Matrix - Factorization:

Preference - prediction is the dot product of user & item factors



Item factors: The extent to which an item has some characteristics

User factors: Level of preference for the corresponding characteristics

Switch to notebook

Matrix - Factorization:

Preference - prediction is the dot product of user & item factors

To get the prediction of a rating of an **item d_j** by a **user u_i** calculate the dot product of the two vectors corresponding to u_i and d_j

Vector corresponding to d_j

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$$

Vector corresponding to u_i

How do we obtain the P and Q matrices?

Matrix - Factorization:

How we obtain the P and Q matrices

1. First we initialize the matrices with some values
2. We calculate how ***'different'*** their product is to M
3. We try to minimize this difference iteratively.

So it is really a minimization problem!

Matrix - Factorization:

Solution to our minimization problem

Gradient Descent: Aims at finding a local minimum of the difference

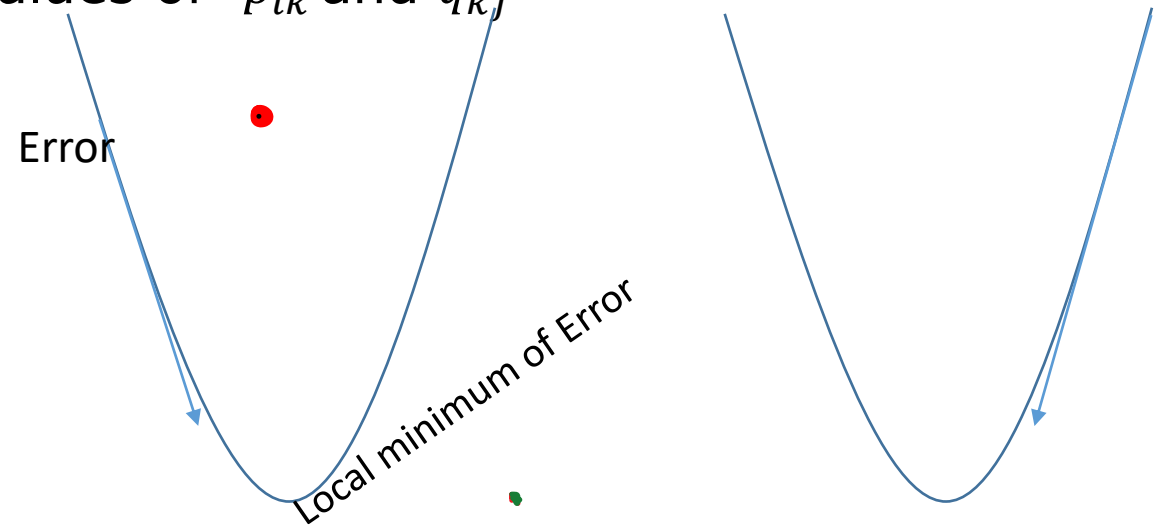
Difference: The error between the estimated rating and the real rating. It can be calculated for each user-item pair:

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

We consider the squared error because the estimated rating can be either higher or lower than the real one.

Gradient descent is a first-order iterative optimization algorithm for finding the minimum o a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient (or approximate gradient) of the function at the current point. If instead one takes steps proportional to the *positive* of the gradient, one approaches a local maximum of that function; the procedure is then known as **gradient ascent**. **When the function is convex, all local minima are also global minima so the gradient discent can converge to the global solution.**

To minimize the error, need to know in which direction we have to modify the values of p_{ik} and q_{kj}



When the function is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

We need to know the gradient at the current values.
Therefore we differentiate the error equation with respect to p_{ik} and q_{kj}

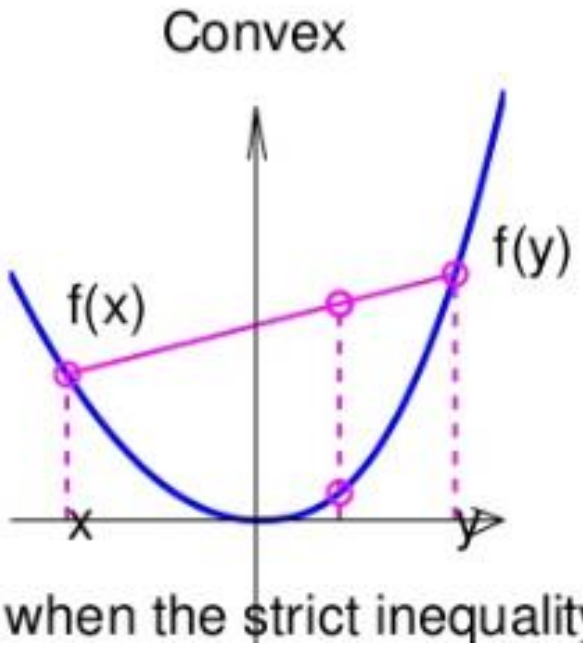
Convex function

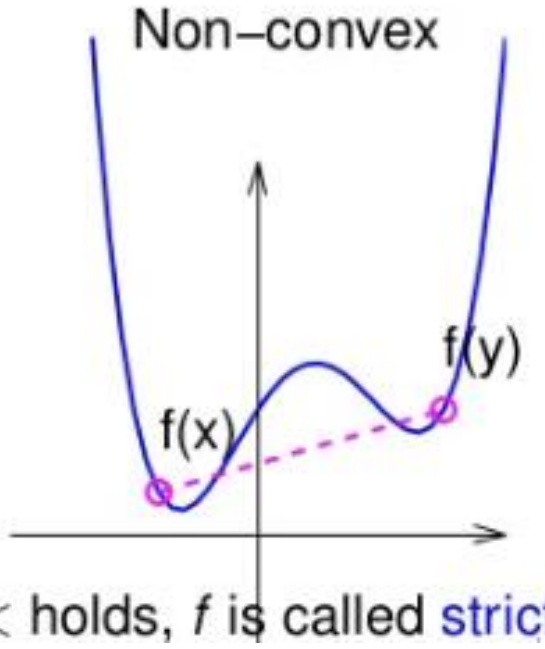
A function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is a convex function

$\Leftrightarrow$ the function f is below any line segment between two points on f; that is,

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n, \forall \lambda \in [0,1], \quad f((1-\lambda)\boldsymbol{x} + \lambda \boldsymbol{y}) \leq (1-\lambda) f(\boldsymbol{x}) + \lambda f(\boldsymbol{y})$$

(Jensen's inequality)





Johan Jensen

1859 – 1925

NB: when the strict inequality $<$ holds, f is called strictly convex.

Matrix - Factorization:

Differentiation of Error Equation

$$\frac{\partial e_{ij}^2}{\partial p_{ik}} = \frac{\partial \left(r_{ij} - \widetilde{r_{ij}}\right)^2}{\partial p_{ik}} = 2*(r_{ij} - \widetilde{r_{ij}})'$$

$$=2*(r_{ij} - \widetilde{r_{ij}}) * (r_{ij} - \widetilde{r_{ij}})' = 2* (r_{ij} - \widetilde{r_{ij}}) *(r_{ij} - \sum_{k=1}^{K} p_{ik} * q_{kj})'$$

$$= 2* (r_{ij} - \widetilde{r_{ij}}) *(r'_{ij} - (\sum_{k=1}^{K} p_{ik} * q_{kj})')$$

$$= 2* (r_{ij} - \widetilde{r_{ij}}) * (-(p_{ik} * q_{kj})') = -2* (r_{ij} - \widetilde{r_{ij}}) * (p_{ik}' * q_{kj})$$

$$= -2* (r_{ij} - \widetilde{r_{ij}}) * q_{kj} = -2* e_{ij} * q_{kj}$$

Similarly when we differentiate with respect to q_{kj}

Matrix - Factorization: Update Rules – Cost Function

Having obtained the gradient, we can now formulate the update rules for both p_{ik} and q_{kj}:

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$

$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij} p_{ik}$$

Cost function

$$\min_{q^*, p^*} \sum_{(u,i) \in K} (r_{ui} - q_i^T p_u)^2$$

α: constant that determines the rate of approaching the minimum

Matrix - Factorization: Overfitting

A model **learns** the **detail and noise** in the **training data** to the extent that it **negatively impacts the performance** of the model on **new data**

This means that the **noise** or **random fluctuations** in the training data is picked up and **learned by the model**

Problem: these concepts do not apply to new data & negatively impact the models ability to generalize!

Matrix - Factorization: Overfitting - Solution

We introduce regularization parameters to our cost function:

Cost function

set of known ratings

prediction error

Regularization parameter

$$\min_{q^*,p^*} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|^2 + \| p_u \|^2)$$

predicted rating

Update rules
(Stochastic Gradient Descend)

Learning rate

previous values

prediction error

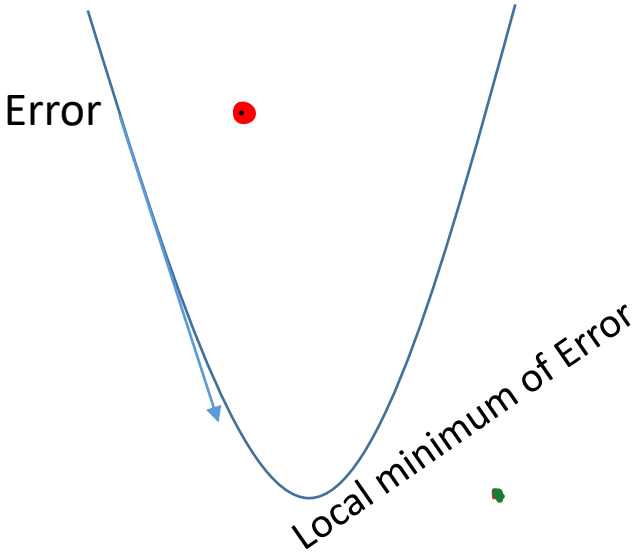
$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$$

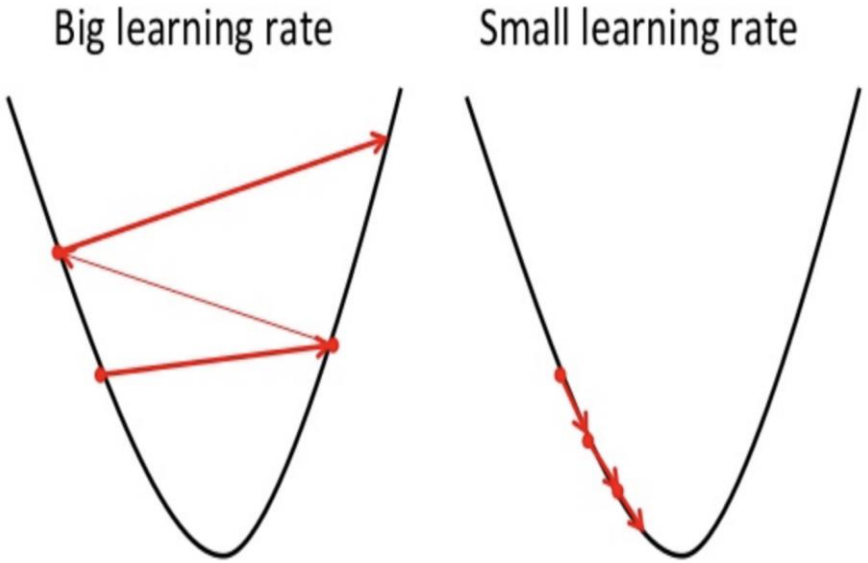
$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$$

Learning rate (γ) and regularization parameter (λ) are hyper-parameters

Matrix - Factorization: Learning rate selection

We choose a small learning rate to avoid making too large a step towards the minimum as we may run into the risk of missing the minimum and end up oscillating around it





Notes

The convexity guarantees that the local minimum is also a global one.

The learning rate will determine how fast it converge and the accuracy of the convergence.

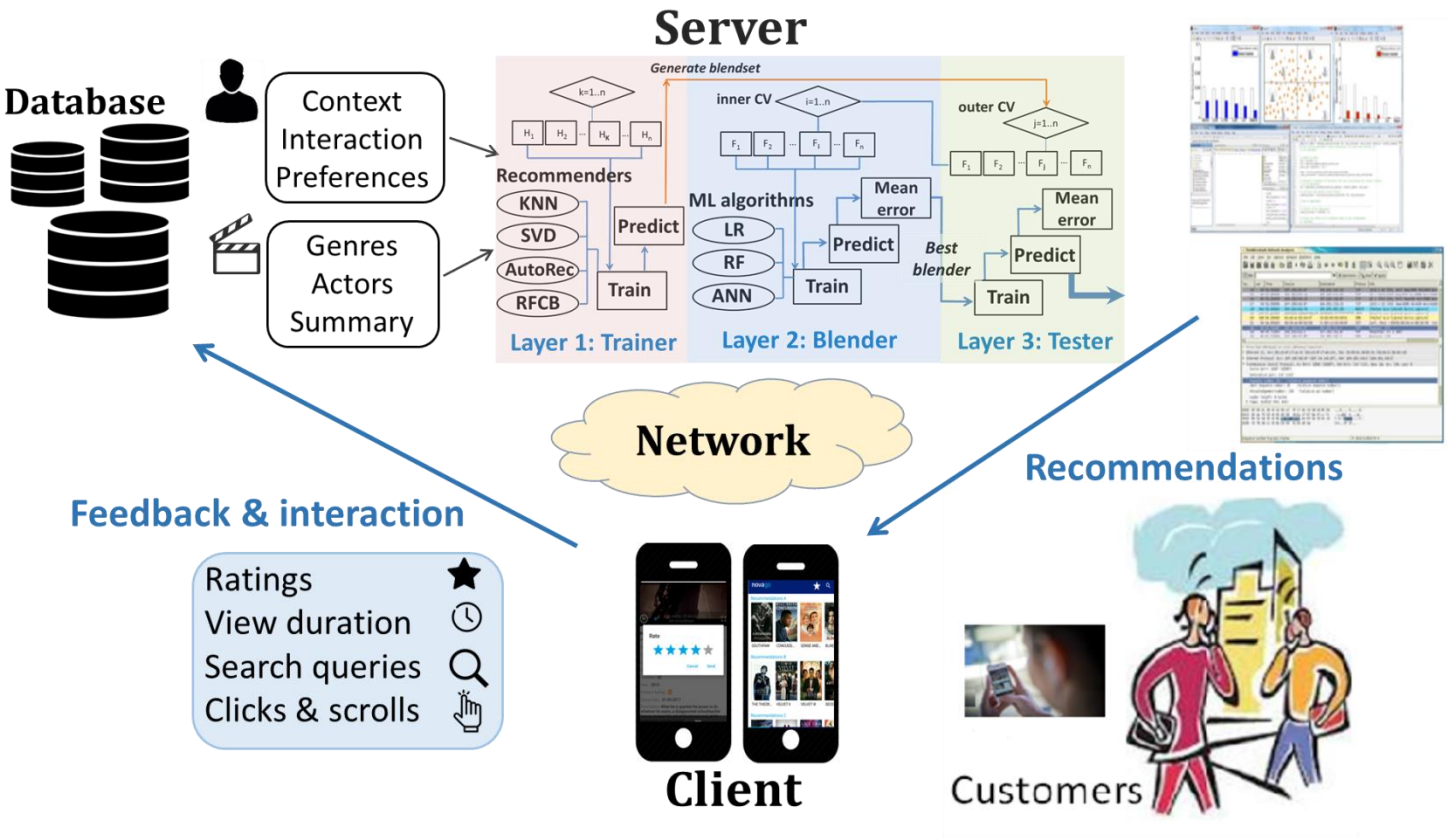
A very small step will guarantee that we will move along the convex function.

Matrix Factorization Techniques

These are collaborating filtering approaches that include methods such as

- Stochastic Gradient Decent (SGD),
- Singular Value Decomposition (SVD),
- Probabilistic Matrix Factorization (PMF), and
- Non-negative Matrix Factorization (NMF).

Recommendation System



Client collects user preferences & watching behavior

Server stores user information & trains periodically recommenders