Tampere, Finland – June 2010, 9th

Recommender Systems for Interactive TV

Tutorial

Roberto Turrin, Paolo Cremonesi

roberto.turrin@polimi.it
paolo.cremonesi@poliimi.it

Agenda

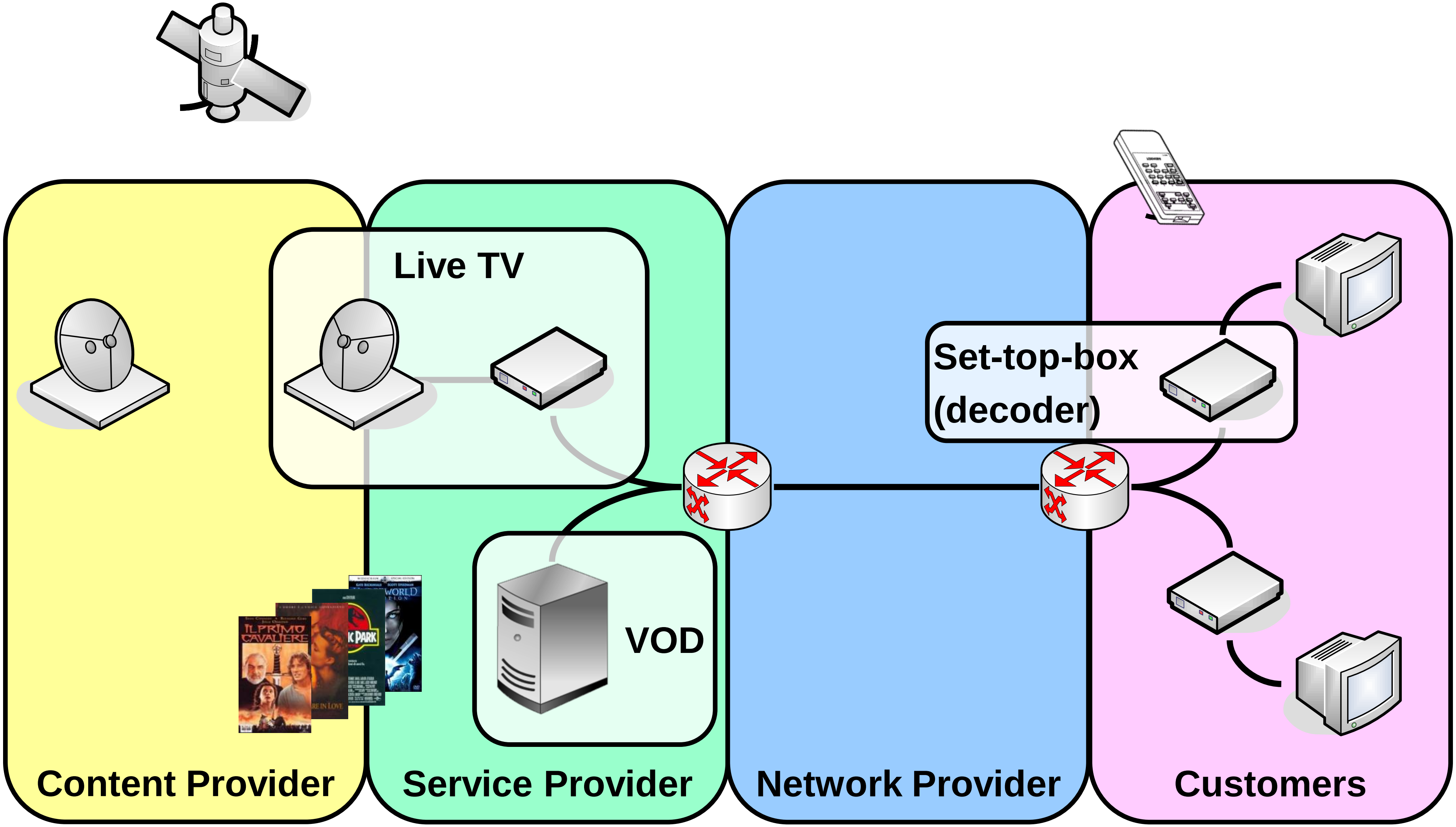
- Introduction
- Recommender systems
  - Content-based and collaborative recommendations
  - Advanced algorithms
- Evaluation of recommender systems

- Case study: a recommender system for IPTV/VOD provider
- Recommender system demo

Agenda

- Introduction
- Recommender systems
  - Content-based and collaborative recommendations
  - Advanced algorithms
- Evaluation of recommender systems
- Case study: a recommender system for IPTV/VOD provider
- Recommender system demo

A scenario: IPTV



IPTV Platform: Now

HUNDREDS LIVE CHANNELS

THOUSANDS VOD ITEMS

CUSTOMERS FACE DIFFICULTIES FINDING THE “<u>RIGHT</u>” CONTENT

CUSTOMER PURCHASES

CUSTOMER FRUSTRATION

IPTV Platform: with a recommender system

From this….

To this.

Today recommendations,
based on your personal taste, are:

Peculiarities in the settings of VoD/IPTV

- Large number of items (movies, TV series, music, etc)
- Slow channel switching
- The available resources increase continuously so each item gets less and less visibility
- The amount of resources interesting for a single user is relatively low
- The interaction between user & system is a remote control

Agenda

- Introduction
- Recommender systems
  - Content-based and collaborative recommendations
  - Advanced algorithms
- Evaluation of recommender systems
- Case study: a recommender system for IPTV/VOD provider
- Recommender system demo

Recommender systems

A recommender system is meant to alleviate the problem of searching for personalized resources by proposing to a user a very limited set of items that he is probably interested in.

IPTV Provider Needs

- Improve user satisfaction, surprise users
  - Users pay for a better service
- Selling new content to users (increase sales)
  - VOD (Video On Demand)
  - Pay-per-view channels
- Targeted advertisement

How recommender systems works

Main **inputs:**

- User tastes?
  - User ratings (profile) ← Required
    - *Explicit*
    - *Implicit*
  - content preferences
- Item content information

Main **outputs:**

- Predicted ratings
- Customized list of items ← Main interest

User ratings

Explicit Ratings: each user can actively express his judgement about each item, for example by assigning a numerical value to it.

- ☺ Easy computation of recommendation
- ☹ Users might get annoyed and leave hasty ratings or don’t leave a rating at all
- ☹ Ratings are subjected to personal interpretation

Implicit Ratings: the recommendation system implicitly infers the rating by observing the user behaviour (time spent watching a particular TV show, purchased items history…)

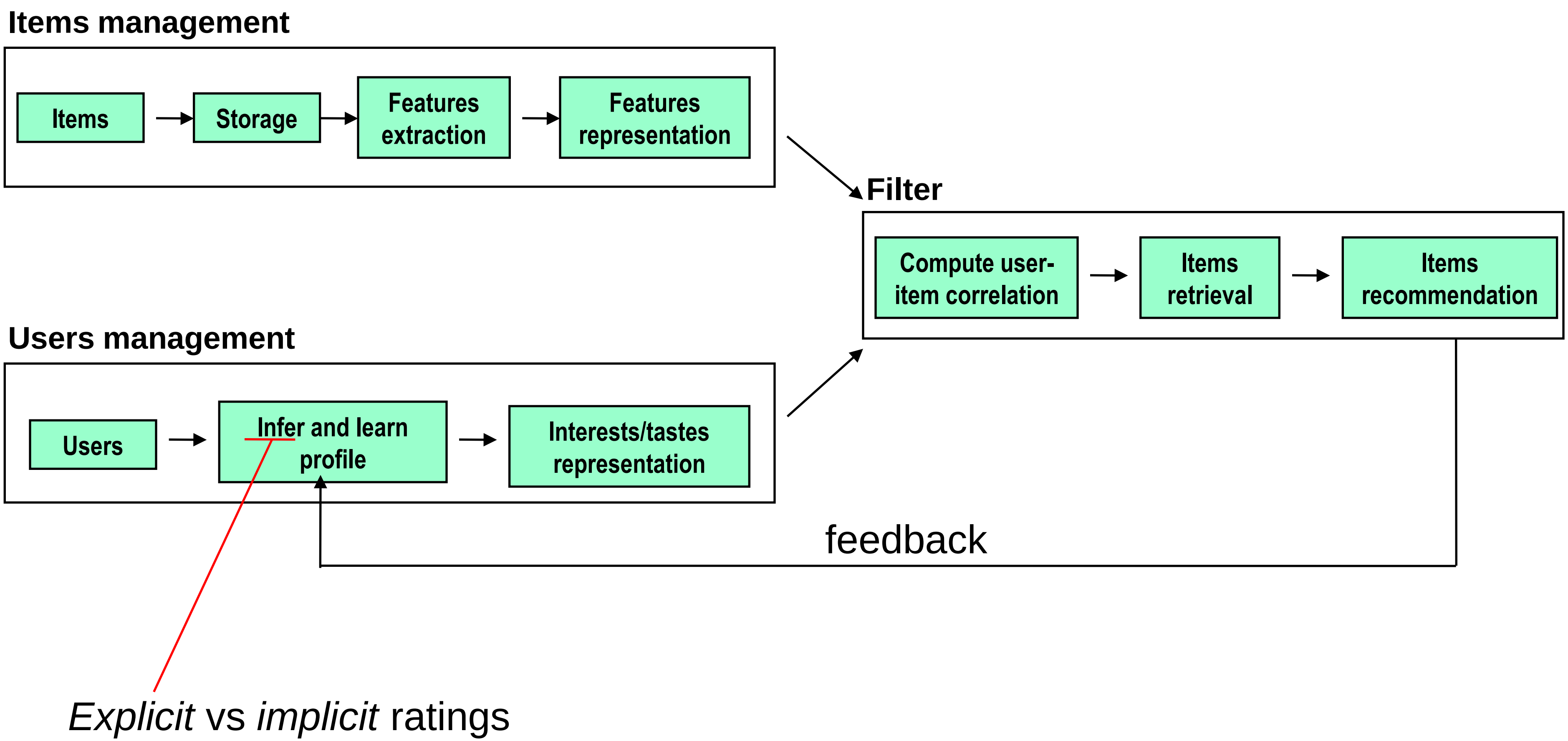
- ☺ Transparent to the users (not annoyed)
- ☺ Often offers effective recommendation...
- ☹ …but not so often perceived as such by the user
- ☹ Requires complex computations/inference

Real domain

In several domains <u>we typically dispose only of an implicit dataset</u> or, sometimes, of very few explicit ratings that coexist in a mainly-implicit dataset, for instance because users get annoyed by leaving too many ratings.

We have a dataset composed by a mixture of explicit and implicit ratings

Recommender architecture



Recommender architecture: requirements

We would like the recommender system to guarantee:

- <u>High quality of recommendations</u>:
  - the capability to suggest personalized items which are potentially interesting to the user and to discard items which are disliked by the user
  - Other aspects of quality..

- <u>Capability to recommend any user at any time</u>

(e.g., even if his profile has just been added)

- <u>Scalability and real-time performance</u>: a recommender system

deals with a large amount of data, so it is crucial to apply algorithms which are capable to satisfy real-time response times

Architecture

The architecture of a recommender system is generally divided into two asynchronous components:

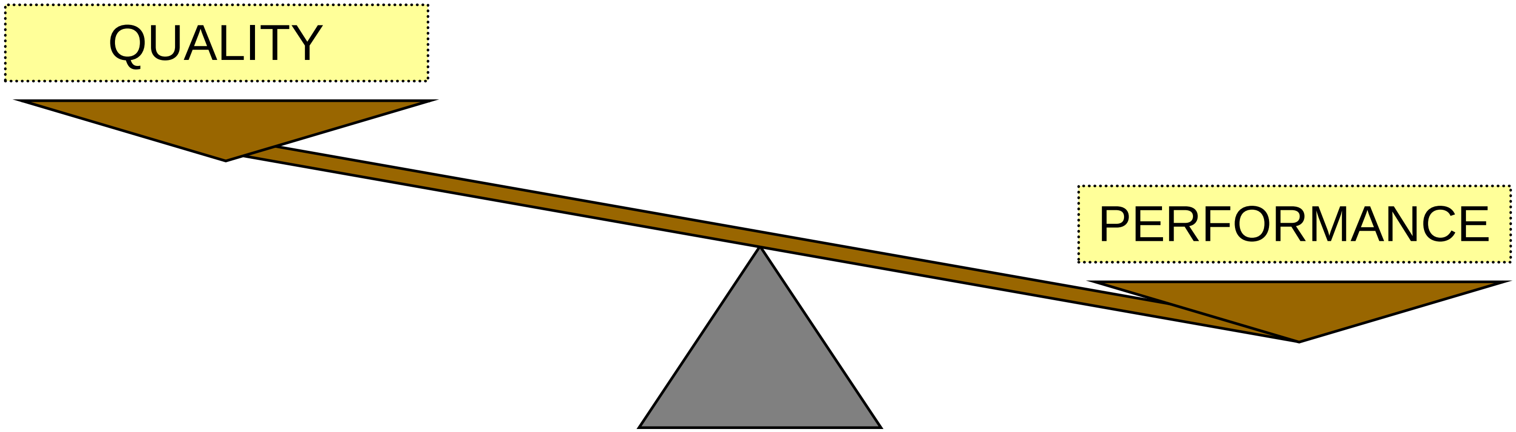
- The ***batch*** or “off line” process
- The ***real-time*** or “on line” process

PURPOSE: move as many operations as possible in the off-line part in order to lighten the on-line part.

Details

- The batch component faces with the most expensive part of the computation with the highest memory and computing requirements

- The real-time component faces the real-time requests for recommendation from the client interface, with rigid time constraints



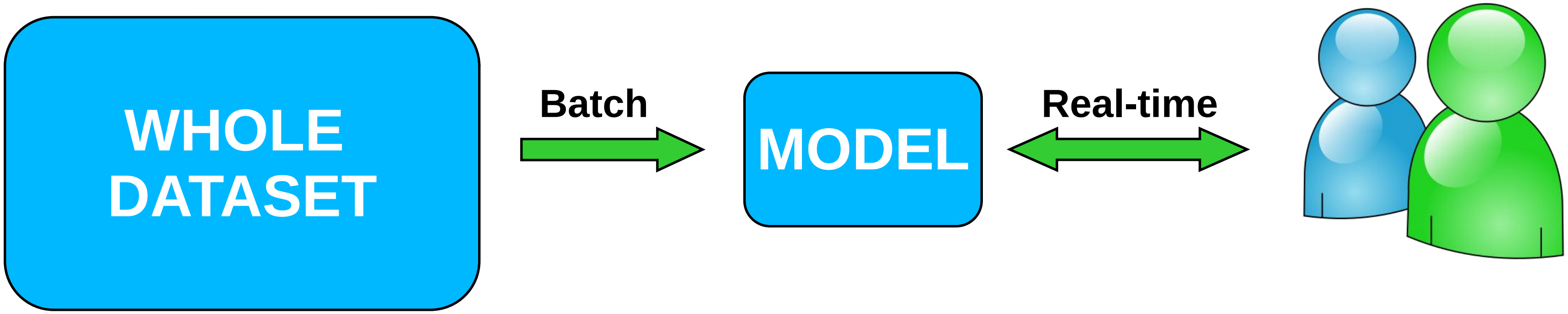
Item-, model-based algorithms

Using **item-, model-based** algorithms!

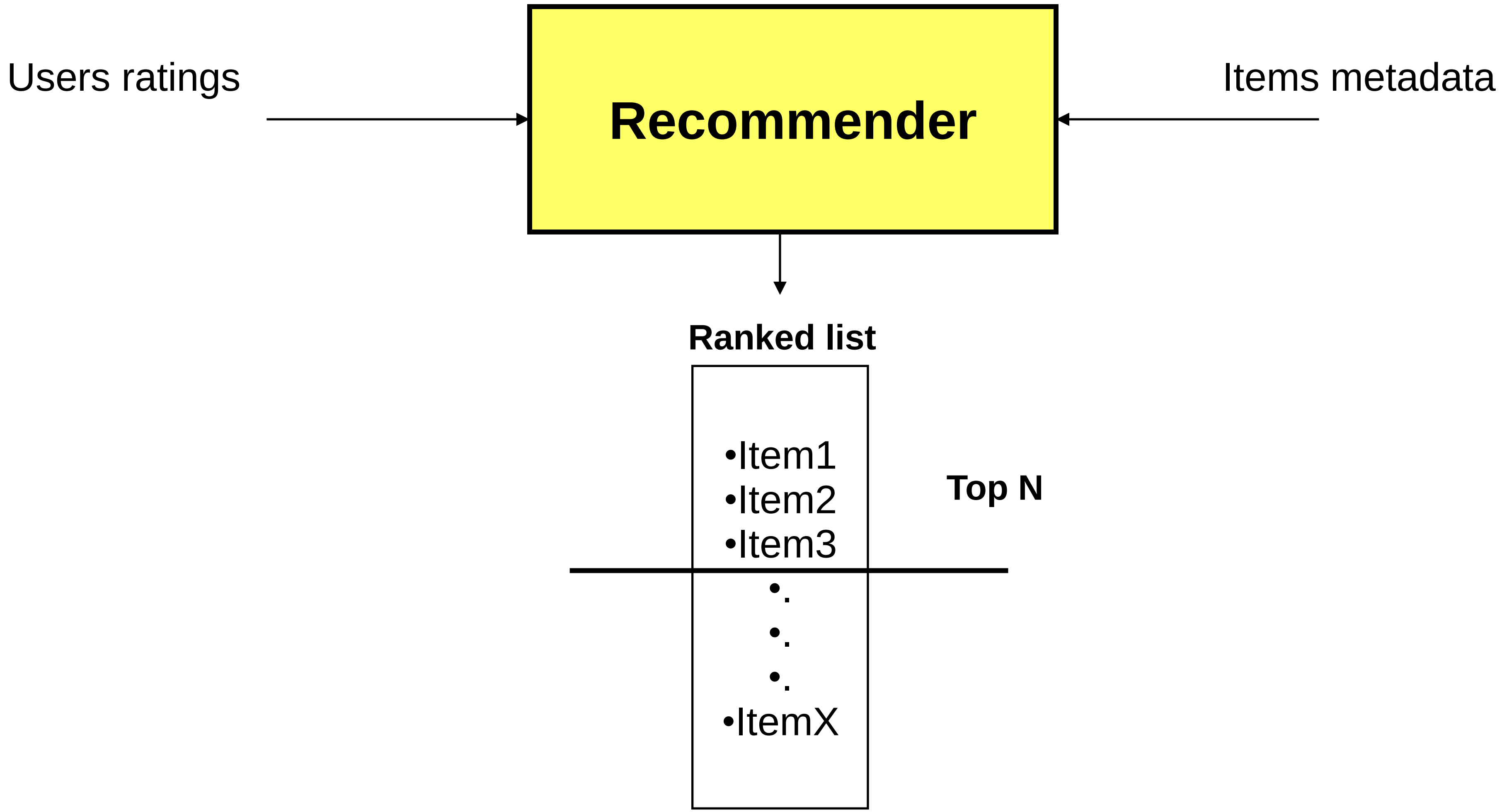
MODEL: a representation of the whole dataset which captures the principal information

item-based: only items are explicitly modeled, user model is implicitly derived

- The real-time component can work on the model instead of the whole dataset
- Generating the model is VERY expensive in terms of calculations.



Problem formulation



Road to personalization

- One of the main goals of recommender systems is providing **personalized suggestions** to customers
  - Higher personalization means better technique

- We can then classify today's solutions in ascending order of personalization effectiveness as follows:
  - ***Non-personalized***: a system which only gives generic suggestions, identical for every viewer
  - ***Roughly personalized***: personalization is trivial (e.g., based on preferred category)
  - ***Personalized***: advanced techniques are used in order to get a strong personalization, this can be obtained via *content-based* or *collaborative* filtering

Non- and roughly personalized systems

- Examples of ***non-personalized*** systems are recommendations based on advertising, popularity (e.g., top sellers) or highest average rating

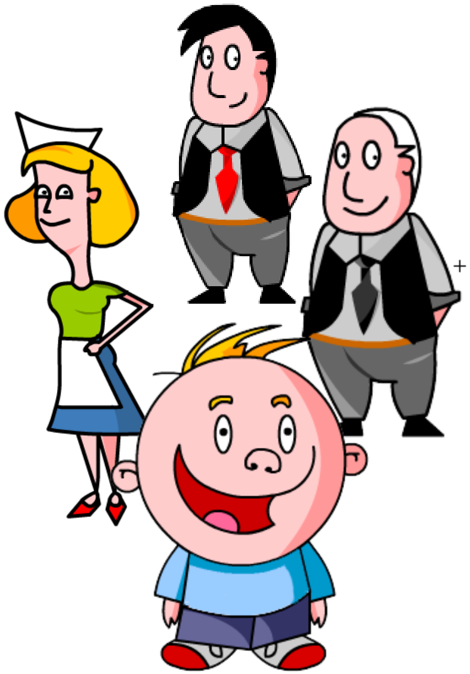
☺ Very easy to implement!
☹ Suggestions are not accurate and thus may be irrelevant

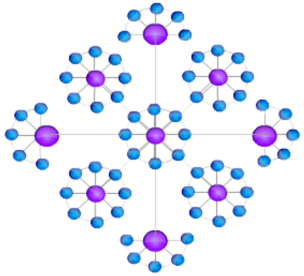
- The main example of ***rough personalization*** is that based on the so-called *demographic profiles*: (e.g., age, gender, location, ...)

☺ The suggestions are not-so-generic
☺ No need to provide info about preferences to get suggestions
☹ High personalization impossible, due to exiguity data about profiles

Non- and roughly personalized systems: baselines

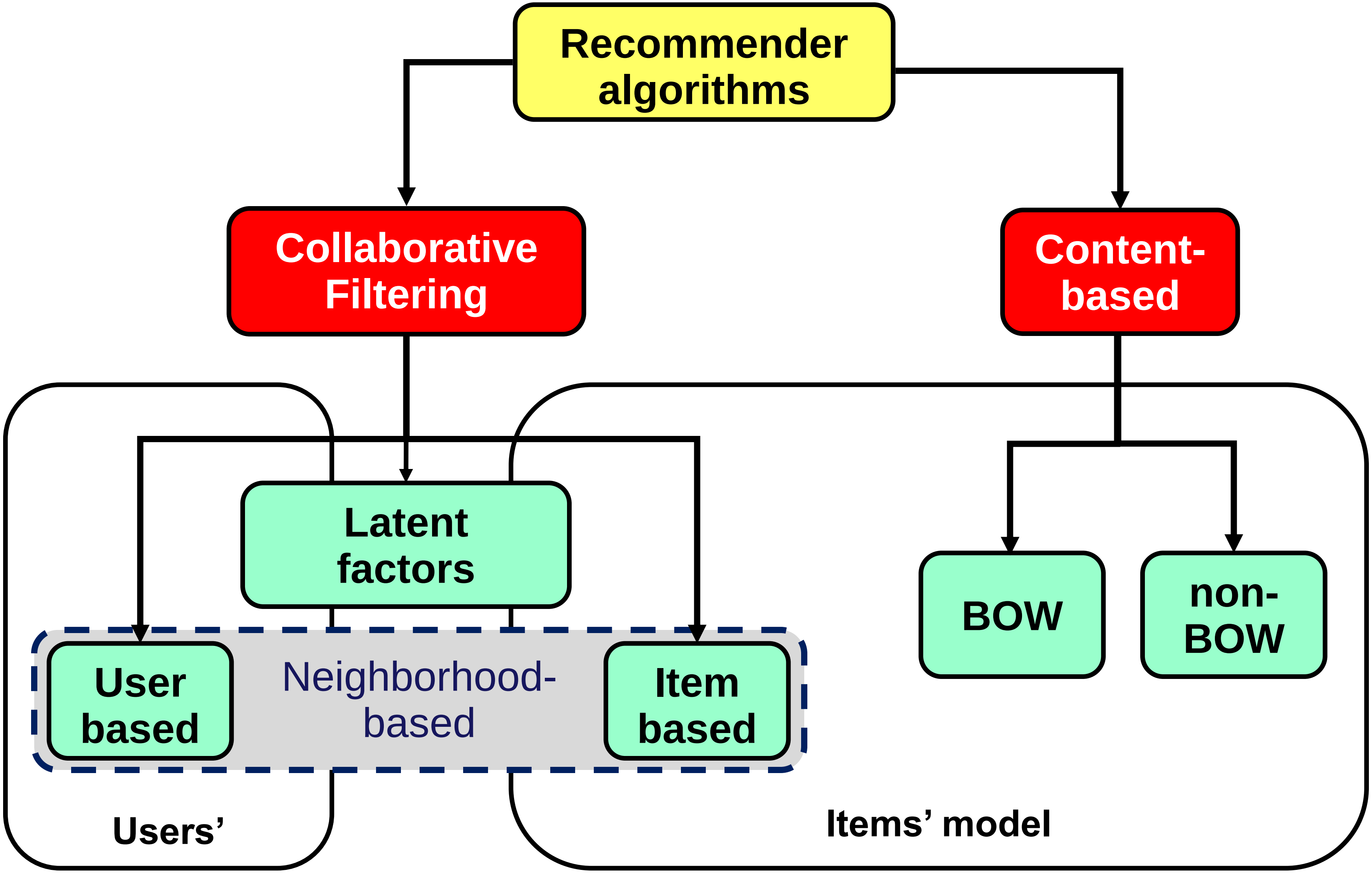
Ratings are affected by subjectivity





$$\tilde{r}_{ui} = r_{ui} - (b_u + b_i + \mu)$$

Personalized systems



- Introduction
- Recommender systems
  - Content-based and collaborative recommendations
  - Advanced algorithms
- Evaluation of recommender systems
- Case study: a recommender system for IPTV/VOD provider
- Recommender system demo

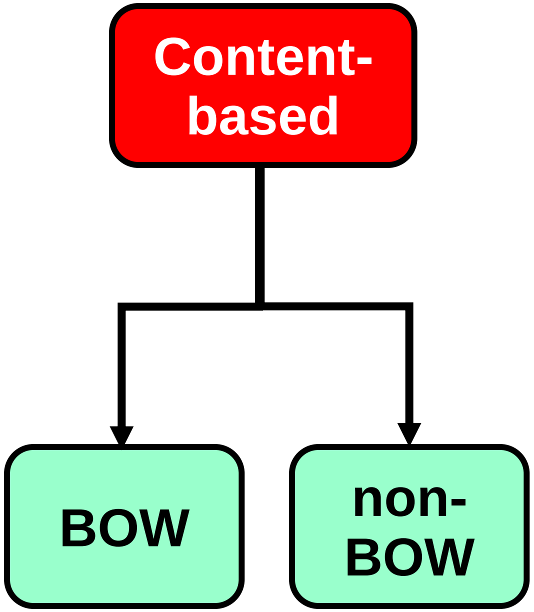
Content-based and collaborative systems

- **Content-based**
  - based on the analysis of the **content** of the items
  - Items are modelled by sets of features representing their content, and the assumption is that those features can somehow capture the meaning and the relevance of the item

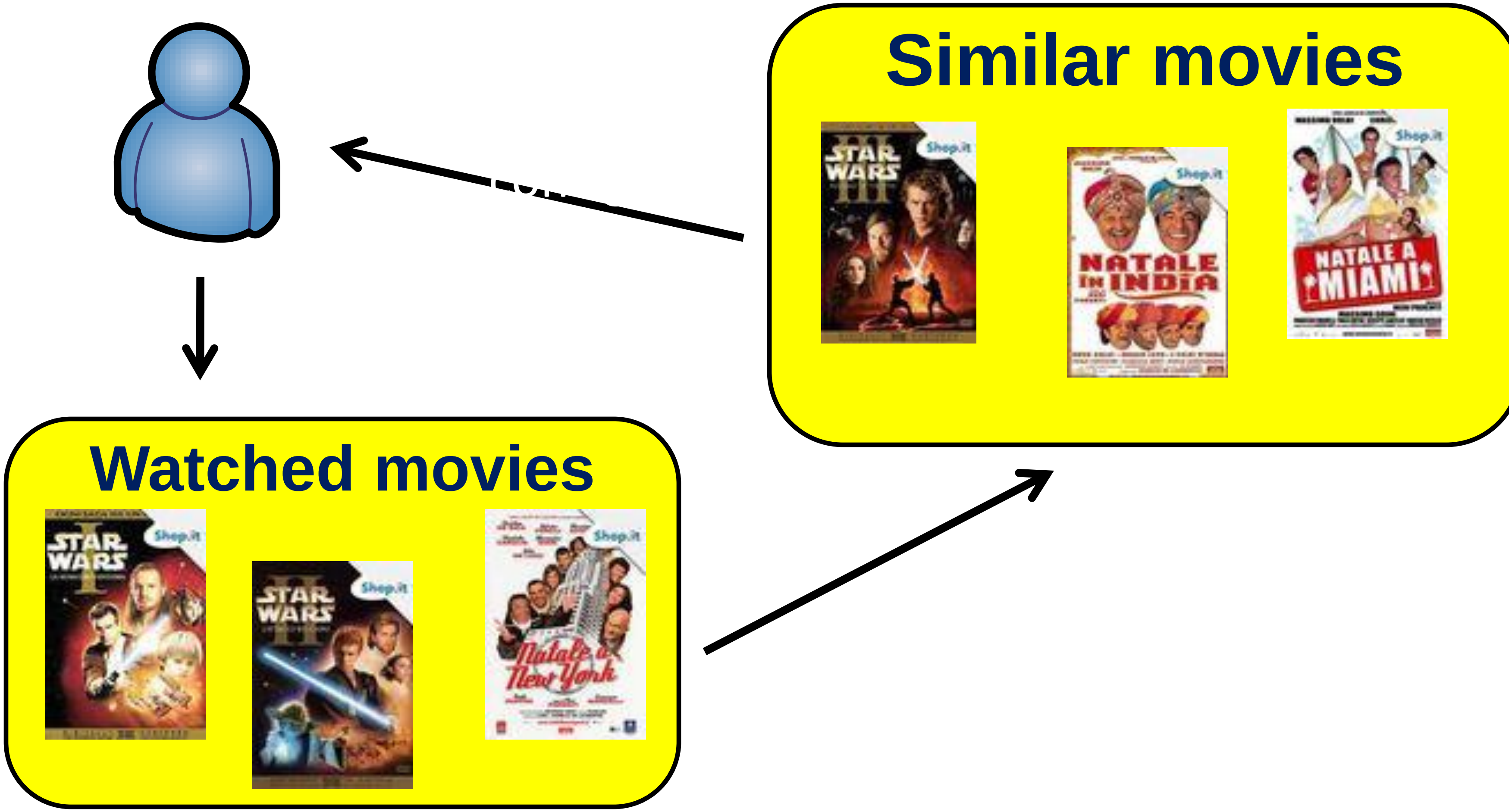
- **Collaborative**
  - based on the preferences expressed by **other users**: it is not needed to extract any explicit features from the items, since recommendations are based only on the community behaviour

Content-based systems

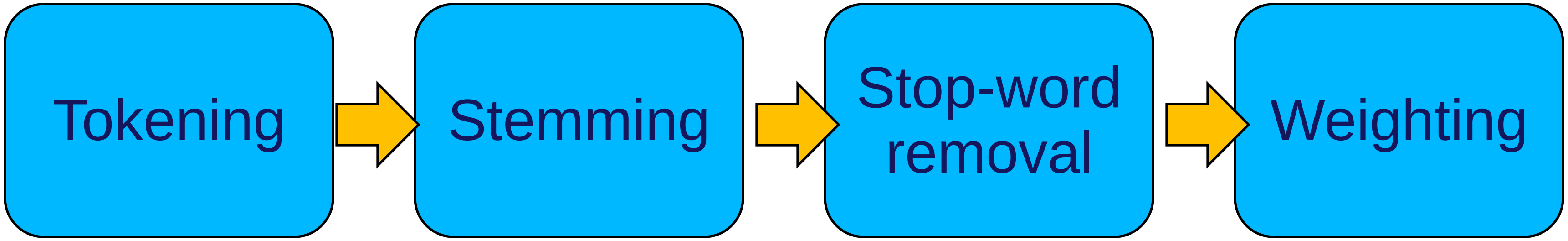
- **Content-based**
  - based on the analysis of the **content** of the items
  - Items are modelled by sets of features representing their content, and the assumption is that those features can somehow capture the meaning and the relevance of the item



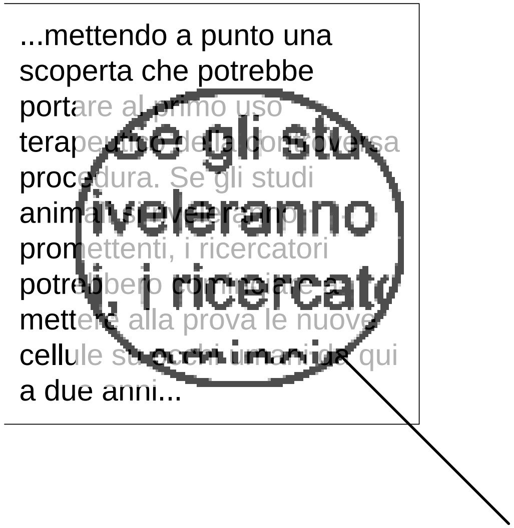
Content-based systems

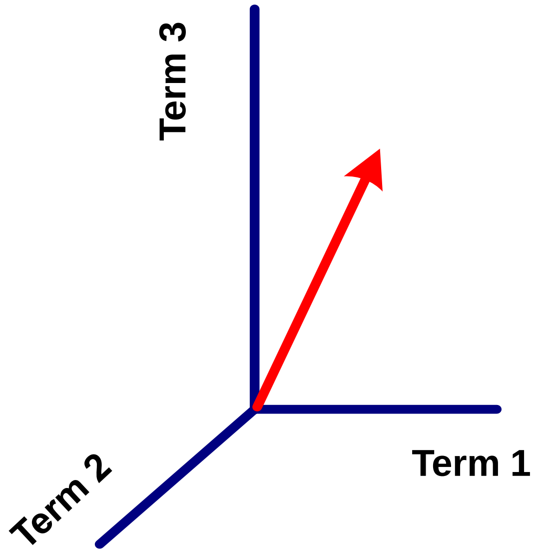


BoW: Bag of Words



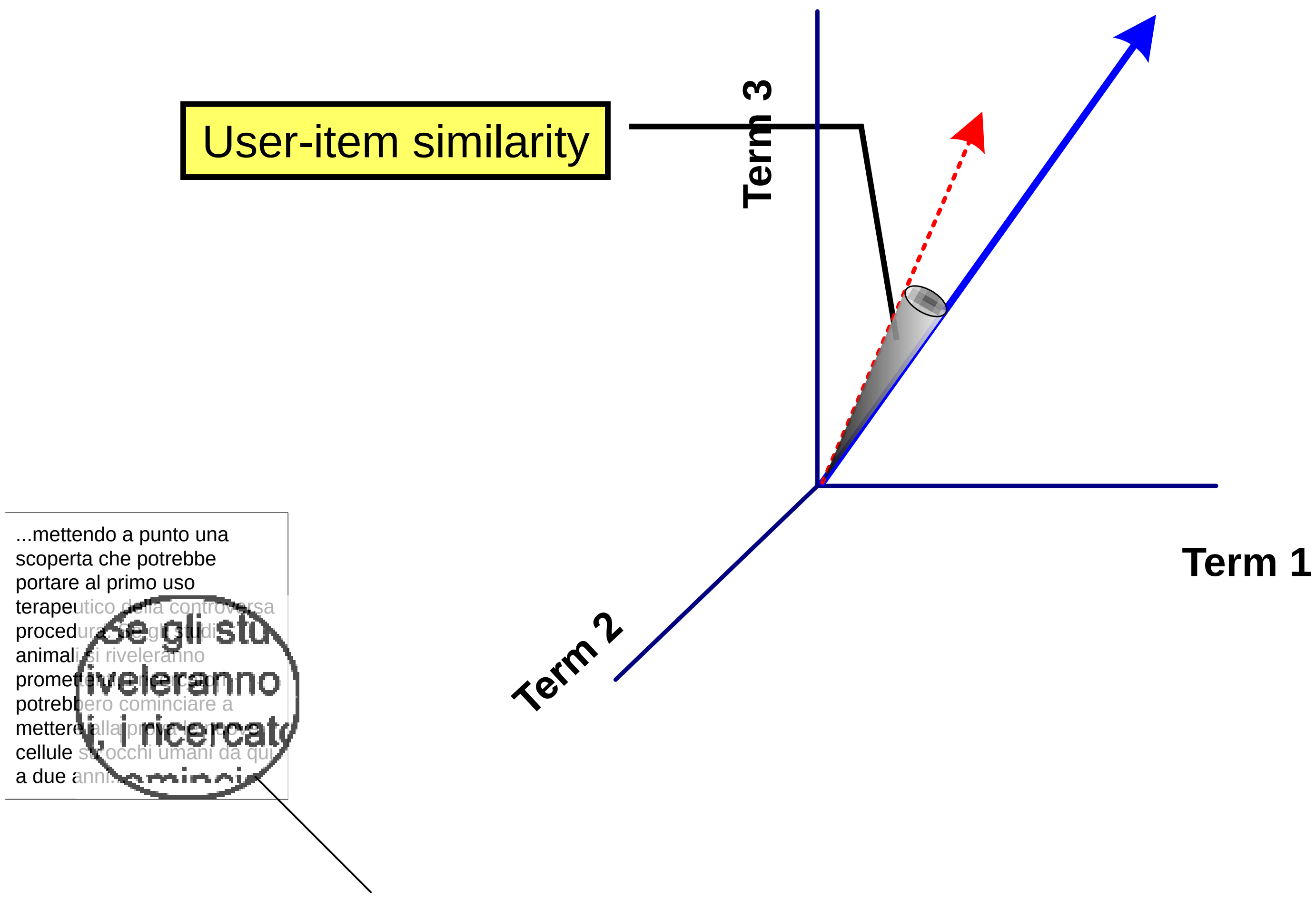
Content-based filtering



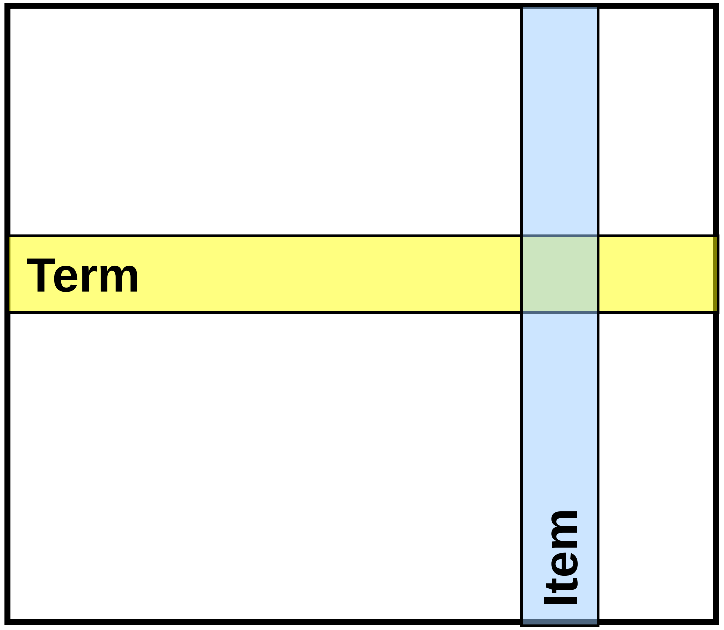


- Similar items contain the same terms (features)
- The more a term occurs in an item, the more representative it is
- The more a term occurs in the collection, the less representative it is (i.e. it is less important in order to distinguish a specific item)

Content-based filtering: prediction



Content-based filtering: Item-Content Matrix



- Each column represents an item, each row a term considered a tag or a keyword indexing a feature of the items
- An element r_{ij} at the *i-th* row and *j-th* column is equal to 1 if the item j contains the term i, 0 otherwise.

- The ***Item-Content Matrix*** will then be a **n**x**m** matrix where **n** is the number of terms and **m** the number of items in the dataset
- Big and sparse matrix: → matrix factorization

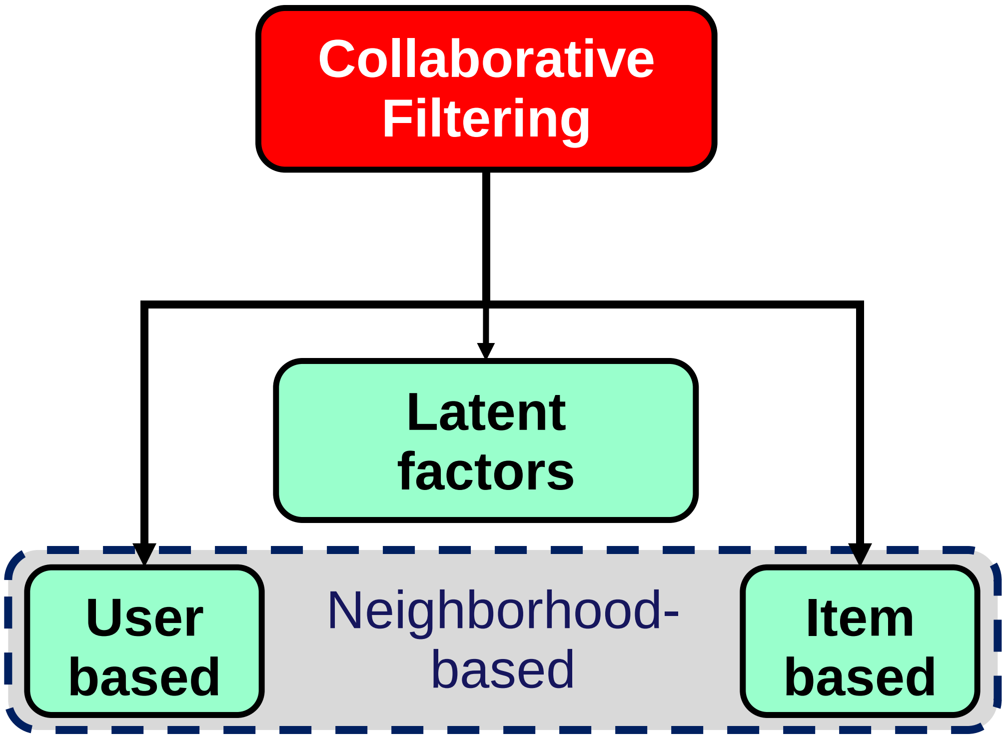
Content-based filtering: pro & cons

- Pro:
  - No need for data about other users' preferences
  - No cold-start or sparsity problems, neither first-rater
    - Able to recommend users with unique tastes
    - Able to recommend new and unpopular items
  - Can provide explanations about recommended items
  - Well-known technology
  - Can integrate explicit (content) preferences

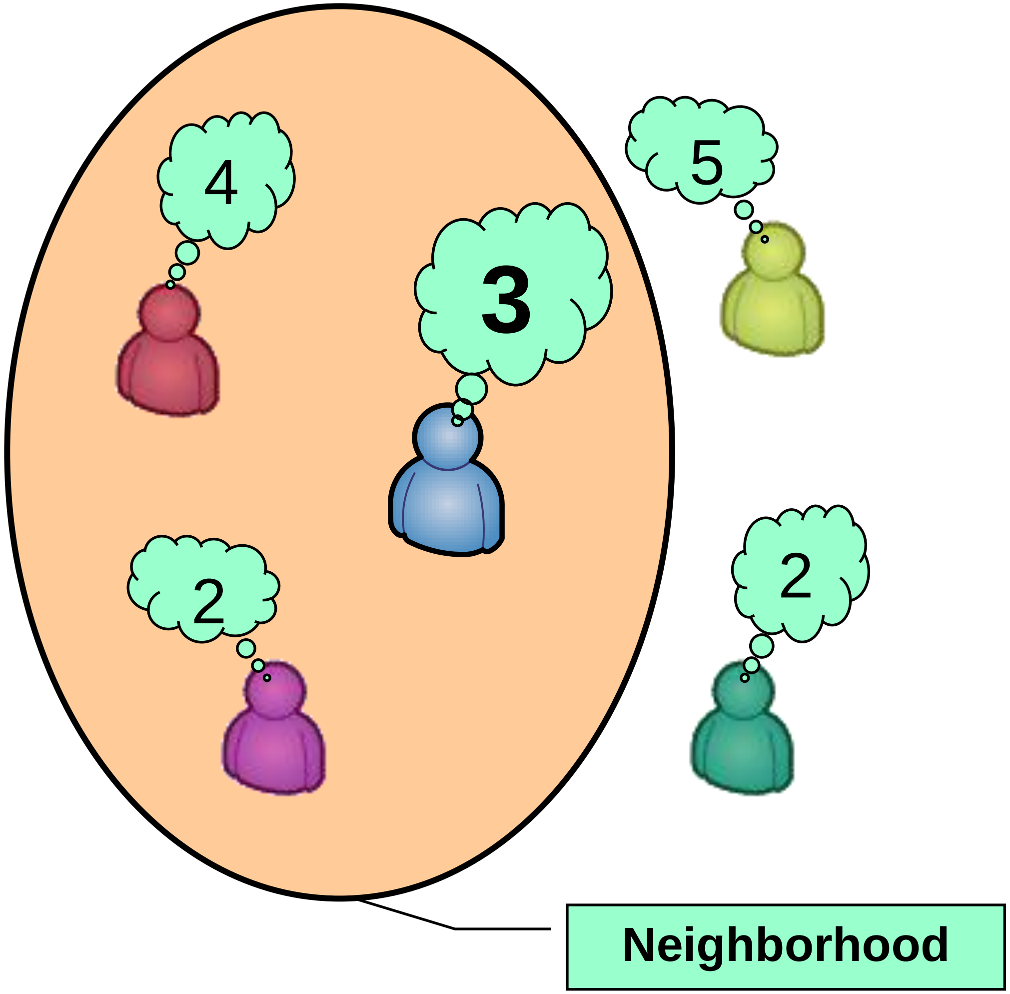
- Cons:
  - Requires a structured content
  - Lower accuracy (than collaborative)
  - Users' tastes must be represented as a function of the content
  - Unable to exploit quality judgments of other users
    - Cannot distinguish between a “valuable” item and a “bad” one
  - Cannot generate new interests in users

Collaborative systems

- **Collaborative**
  - based on the preferences expressed by **other users**: it is not needed to extract any explicit features from the items, since recommendations are based only on the community behaviour



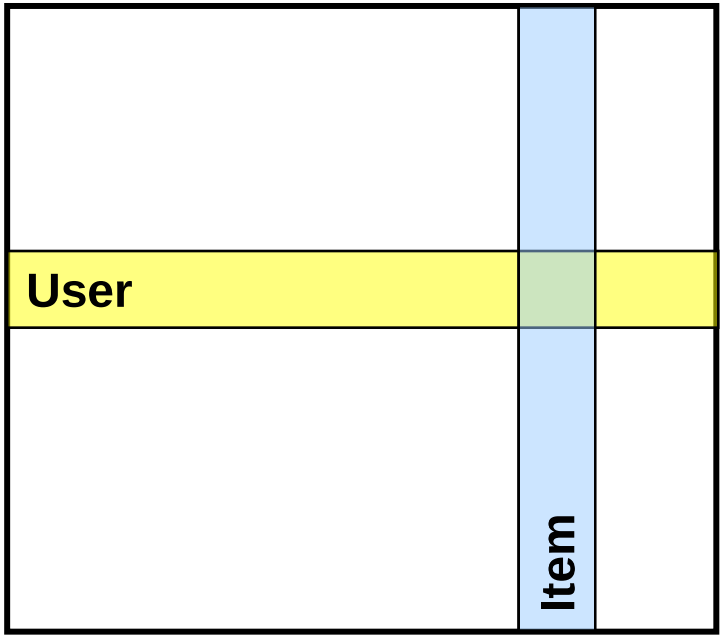
Collaborative filtering: user-based



<u>User-based</u>

similar users rate an item similarly: the *neighborhood* is the set of users who are considered similar to a certain user according to such definition

Collaborative filtering: User-Rating Matrix



- Each column represents an item, each row a user profile
- An element r_{ij} at the *i-th* row and *j-th* column is the rating of user profile i about item j
- An element r_{ij} is null (zero) if the user did not interact with the item

- The ***User-Rating Matrix*** contains info about the user interaction with the system and is the only info required by collaborative algorithms. It is a **n**x**m** matrix where **n** is the number of users and **m** the number of items in the dataset
- Big and sparse matrix: → factorization

Explicit and implicit URM

- The simplest form of implicit rating is a binary classification: r_{ij} is equal to one if the user interacted with an item, zero otherwise

	I_1	I_2	I_3	I_4
U_1	3	4	0	1
U_2	2	2	1	0
U_3	2	0	0	4
U_4	1	5	0	1
U_5	3	0	1	0

Explicit URM

	I_1	I_2	I_3	I_4
U_1	0	1	0	1
U_2	0	0	1	1
U_3	1	1	0	0
U_4	1	1	0	1
U_5	0	1	1	1

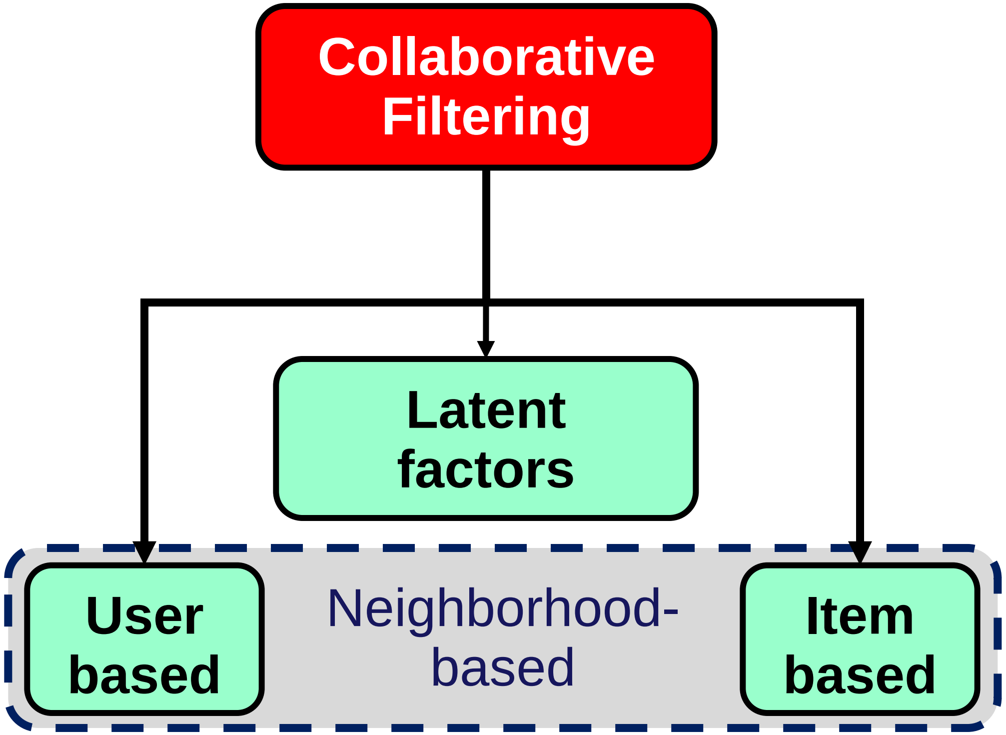
Implicit URM

Collaborative Filtering: pro & cons

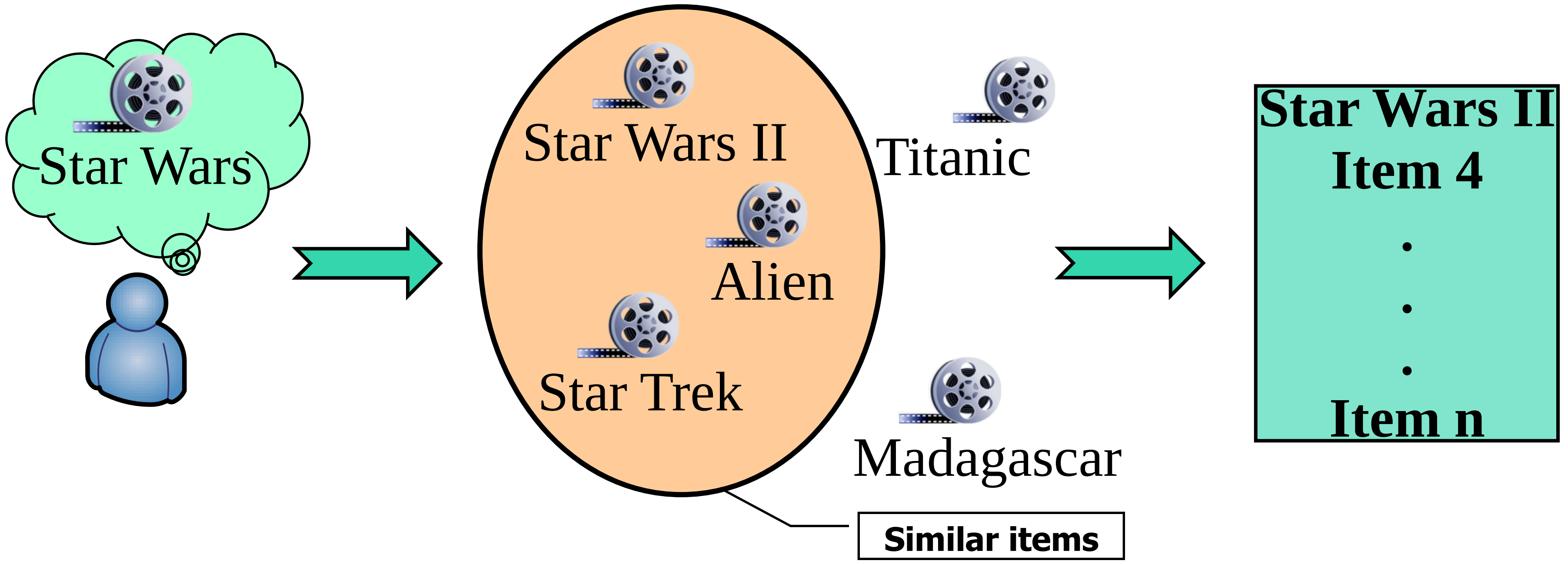
- Pro:
  - There is no need for content

- Cons:
  - ***Cold Start***: we need to have enough users in the system to find a match.
  - ***Sparsity***: when the user/ratings matrix is sparse it is hard to find a neighborhood.
  - ***First Rater***: cannot recommend an item that has not been previously rated by anyone else
  - ***Popularity Bias***: cannot recommend items to someone with unique tastes. Tends to recommend popular items, since they are the most rated (dataset coverage)

Item-based (neighborhood) collaborative filtering



Item-based collaborative filtering



Item-based

similar items are rated by a user similarly: the *neighborhood* is the set of items which are considered similar to a certain item according to the community's ratings

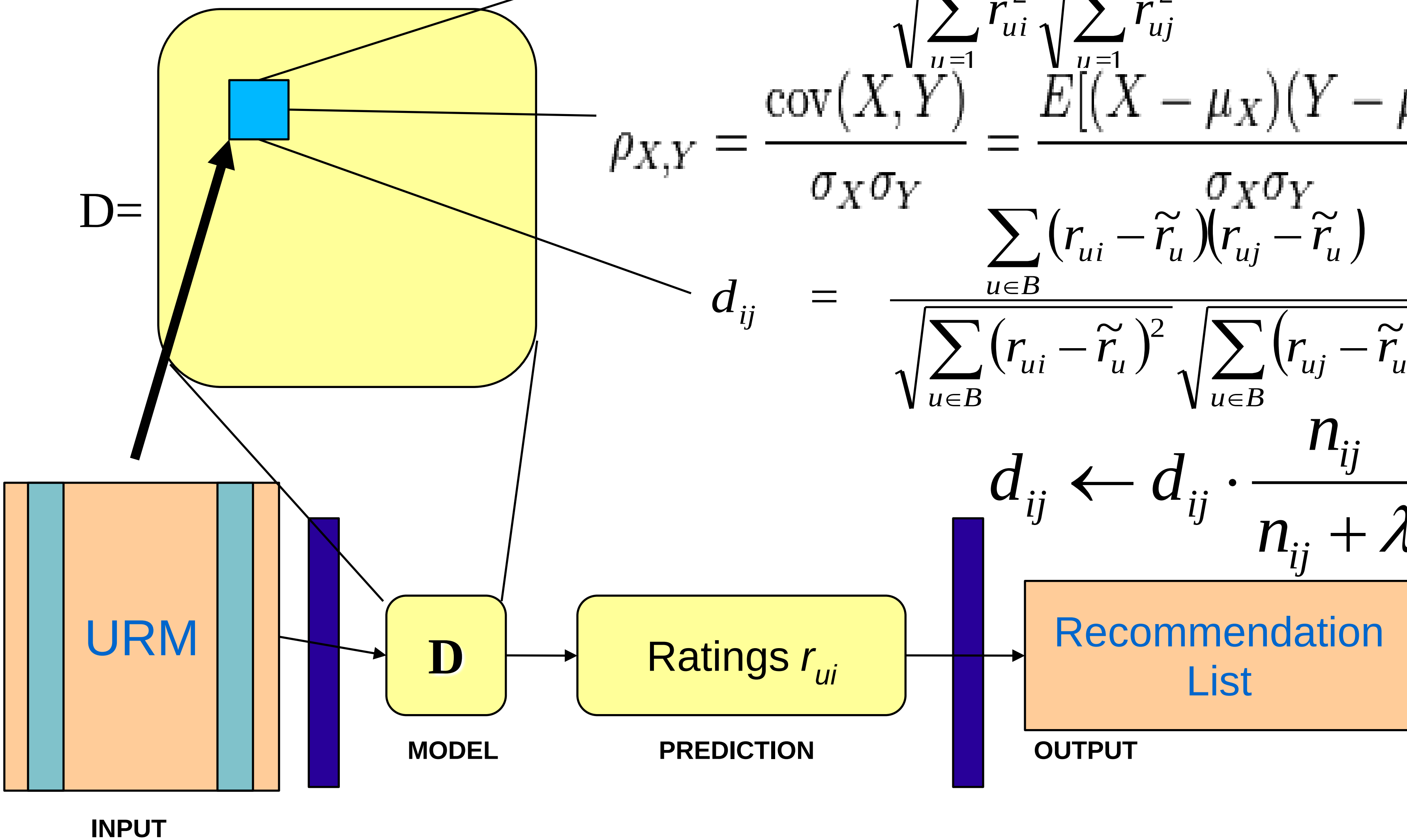
Item-based algorithms

$$d_{ij} = \frac{\sum_{u=1}^{n} r_{ui} r_{uj}}{\sqrt{\sum_{u=1}^{n} r_{ui}^2}\sqrt{\sum_{u=1}^{n} r_{uj}^2}}$$

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y},$$

$$d_{ij} = \frac{\sum_{u \in B} (r_{ui} - \tilde{r}_u)(r_{uj} - \tilde{r}_u)}{\sqrt{\sum_{u \in B} (r_{ui} - \tilde{r}_u)^2}\sqrt{\sum_{u \in B} (r_{uj} - \tilde{r}_u)^2}}$$

$$d_{ij} \leftarrow d_{ij} \cdot \frac{n_{ij}}{n_{ij} + \lambda}$$



Item-based algorithms

- The estimated rating r_{ui} of an item i for a user u is computed as follows:

$$\hat{r}_{ui} = K + \frac{\sum_{j \in S^k(i;u)} d_{ij} r_{uj}}{Q} \qquad Q = \sum_{j \in S^k(i;u)} \left(d_{ij} \right)$$

$$\Downarrow$$

$$\hat{r}_{ui} = b_{ui} + \left| R^k(i;u) \right|^{-\frac{1}{2}} \sum_{j \in R^k(i;u)} w_{ij} (r_{uj} - b_{uj}) + \left| N^k(i;u) \right|^{-\frac{1}{2}} \sum_{j \in N^k(i;u)} c_{uj}$$

Item-based algorithms

- The model – **W**, **C**, **b$_u$**, **b$_i$** – is learnt by **minimizing** the **error (RMSE)** between known ratings and predicted ratings

$$\min \sum \left(r_{ui} - \hat{r}_{ui}\right)^2$$

$$\min \sum \left(r_{ui} - \hat{r}_{ui}\right)^2 + \boxed{\lambda\left(b_u^{\ 2} + b_i^{\ 2} + \sum w_{ij}^{\ 2} + \sum c_{ij}^{\ 2}\right)}$$

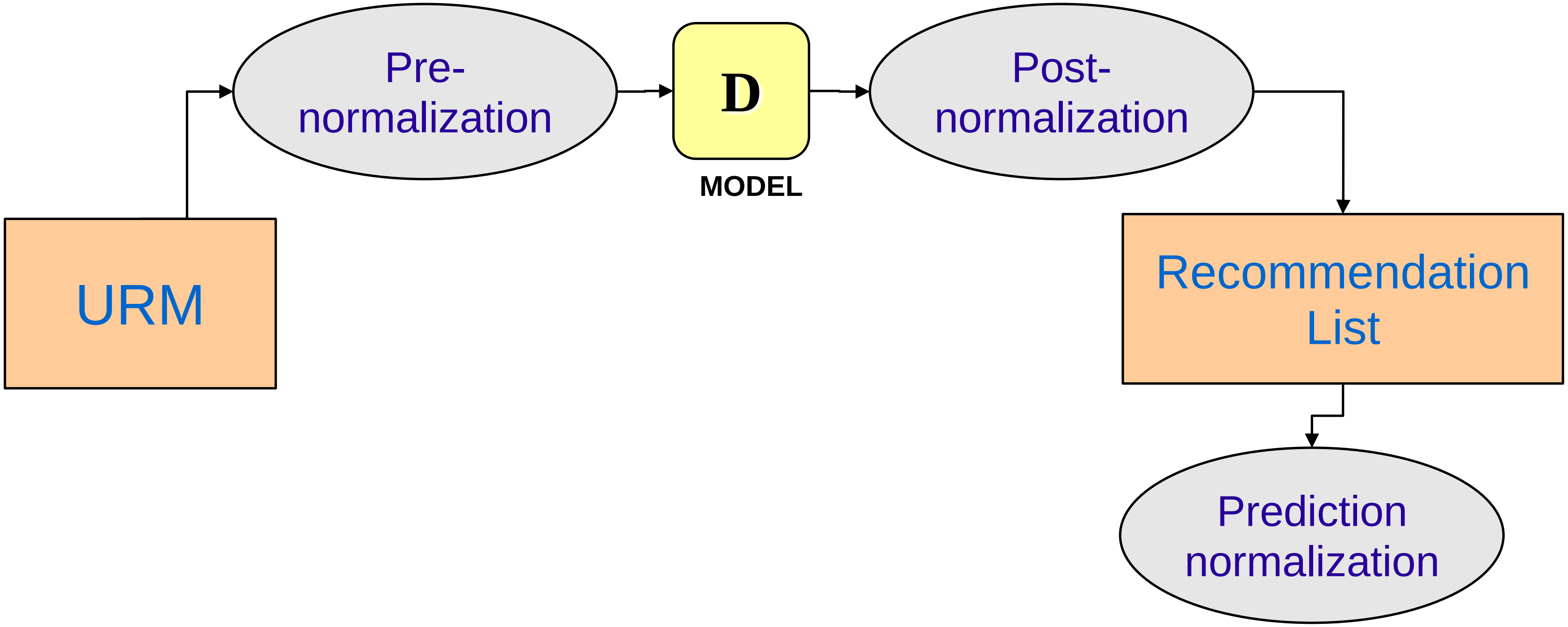
prediction rule

$$\hat{r}_{ui} = b_{ui} + \left|R^k(i;u)\right|^{-\frac{1}{2}} \sum_{j \in R^k(i;u)} w_{ij}\left(r_{uj} - b_{uj}\right) + \left|N^k(i;u)\right|^{-\frac{1}{2}} \sum_{j \in N^k(i;u)} c_{uj}$$

Gradient descent

Observation: Normalization

- The *adjusted cosine* similarity metric shows a first example of **normalization** applied to the dataset



Item-based algorithms: managing implicit URM

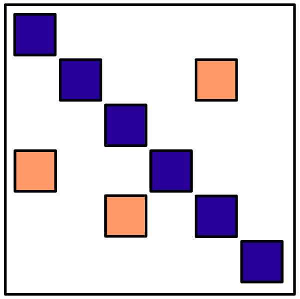
- Note: not all metrics for explicit datasets can be used
  - E.g., *adjusted cosine* does not work: subtracting the average of all user ratings would reset to zero all the user ratings!

- Similarity metric is usually ***frequency-based***
  - For instance, a high similarity between item *i* and item *j* means that if user *u* bought item *i* he will most likely buy also item *j*
- *Cosine similarity* can still be used, but a post-normalization process is desirable.
  - Furthermore, with regard to implicit ratings, cosine is just a special case of a more general approach that we refer to as ***Direct Relations (DR)***

Item-based algorithms: Direct Relations

- Given the (implicit) URM **R**, the item-item matrix **D** used with DR is computed as

$$\mathbf{D} = \mathbf{R}^{\mathbf{T}} \cdot \mathbf{R}$$



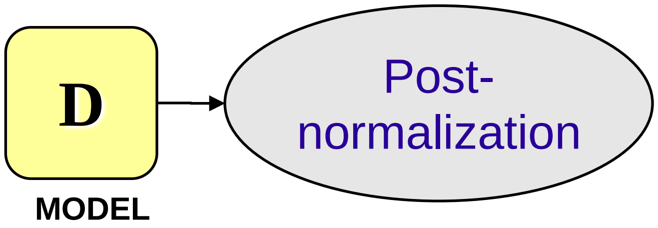
- The element d_{ii} on the principal diagonal represents the total number of ratings for item *i*

- The element d_{ij} represents instead the number of users that have watched both movie i and movie j

Item-based algorithms: post-normalization

- Given the (implicit) URM **R**, the item-item matrix **D** used with DR is computed as

$$\mathbf{D} = \mathbf{R}^T \cdot \mathbf{R}$$



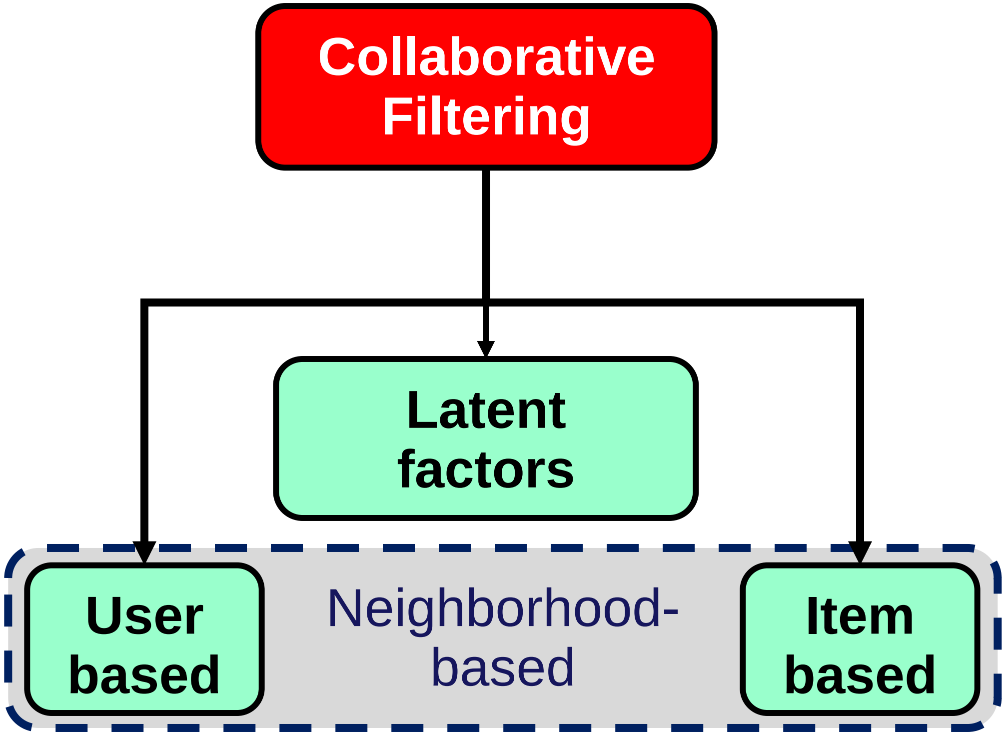
- General definition of post normalization:

$$d_{ij} \leftarrow \frac{d_{ij}}{d_{ij}^{\beta} d_{jj}^{\gamma}}$$

- β and γ are constant parameters whose optimal value depends on the dataset
- both parameters set to 0.5 corresponds to the cosine

- Introduction
- Recommender systems
  - Content-based and collaborative recommendations
  - Advanced algorithms
- Evaluation of recommender systems
- Case study: a recommender system for IPTV/VOD provider
- Recommender system demo

Collaborative filtering: Latent factor models



Latent factor algorithms

- The dataset (users and items) is described by means of a limited set of *f* **hidden features**

Item feature vector

$$\hat{r}_{ui} = p_u^T q_i$$

User feature vector

- We don't know the features!
  - Error minimization (Gradient descent)
  - Matrix factorization

Latent factor algorithms: gradient descent

$$\hat{r}_{ui} = b_{ui} + p_u{}^T q_i \qquad \min \sum (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(b_u{}^2 + b_i{}^2 + \sum |p_u|^2 + \sum |q_i|^2 \right)$$

$$e_{ui} = \hat{r}_{ui} - r_{ui} \Rightarrow \begin{aligned} b_u &= b_u + \gamma(e_{ui} - \lambda b_u) \\ b_i &= b_i + \gamma(e_{ui} - \lambda b_i) \\ &\vdots \\ p_u &= p_u + \gamma(e_{ui} q_i - \lambda p_u) \end{aligned}$$

- We don't know the features! ⇒
  - **Error minimization (gradient descent)**
  - Matrix factorization

Latent factor algorithms: best NETFLIX algos

$$\hat{r}_{ui} = b_{ui} + p_u^{T} q_i$$

$$\hat{r}_{ui} = b_{ui} + q_i^{T}\left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right) + |R^k(i;u)|^{-\frac{1}{2}} \sum_{j \in R^k(i;u)} w_{ij}(r_{uj} - b_{uj}) + |N^k(i;u)|^{-\frac{1}{2}} \sum_{j \in N^k(i;u)} c_{uj}$$

Roughly person.

Item-based neigh

SVD++

Integrated Model

- We don’t know the features!

- **Error minimization (gradient descent)**
- Matrix factorization

Latent factor algorithms: ..a step back to model-based

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right)$$

SVD++

$$p_u = |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} (r_{uj} - b_{uj}) x_j$$

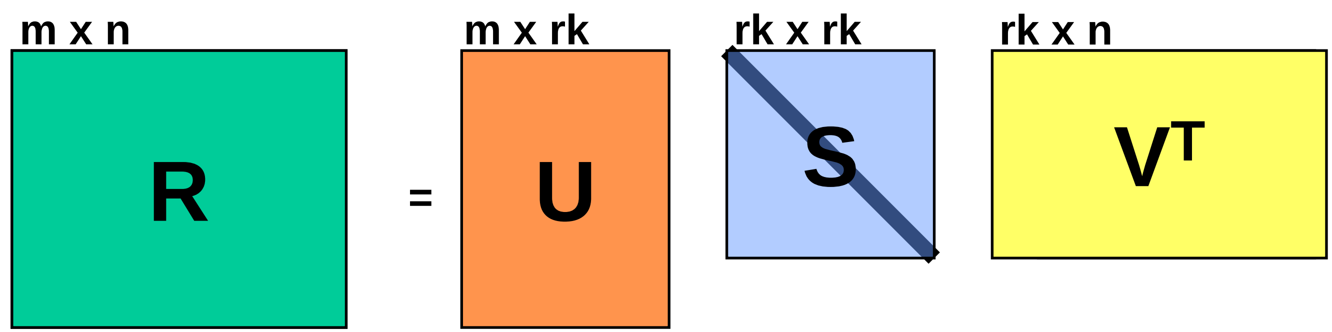
AsySVD

- We don't know the features!

- **Error minimization (gradient descent)**
- Matrix factorization

Latent factor algorithms: ..SVD

$$\hat{r}_{ui} = p_u^T q_i$$



SVD

$$M = U \cdot S \cdot V^T$$

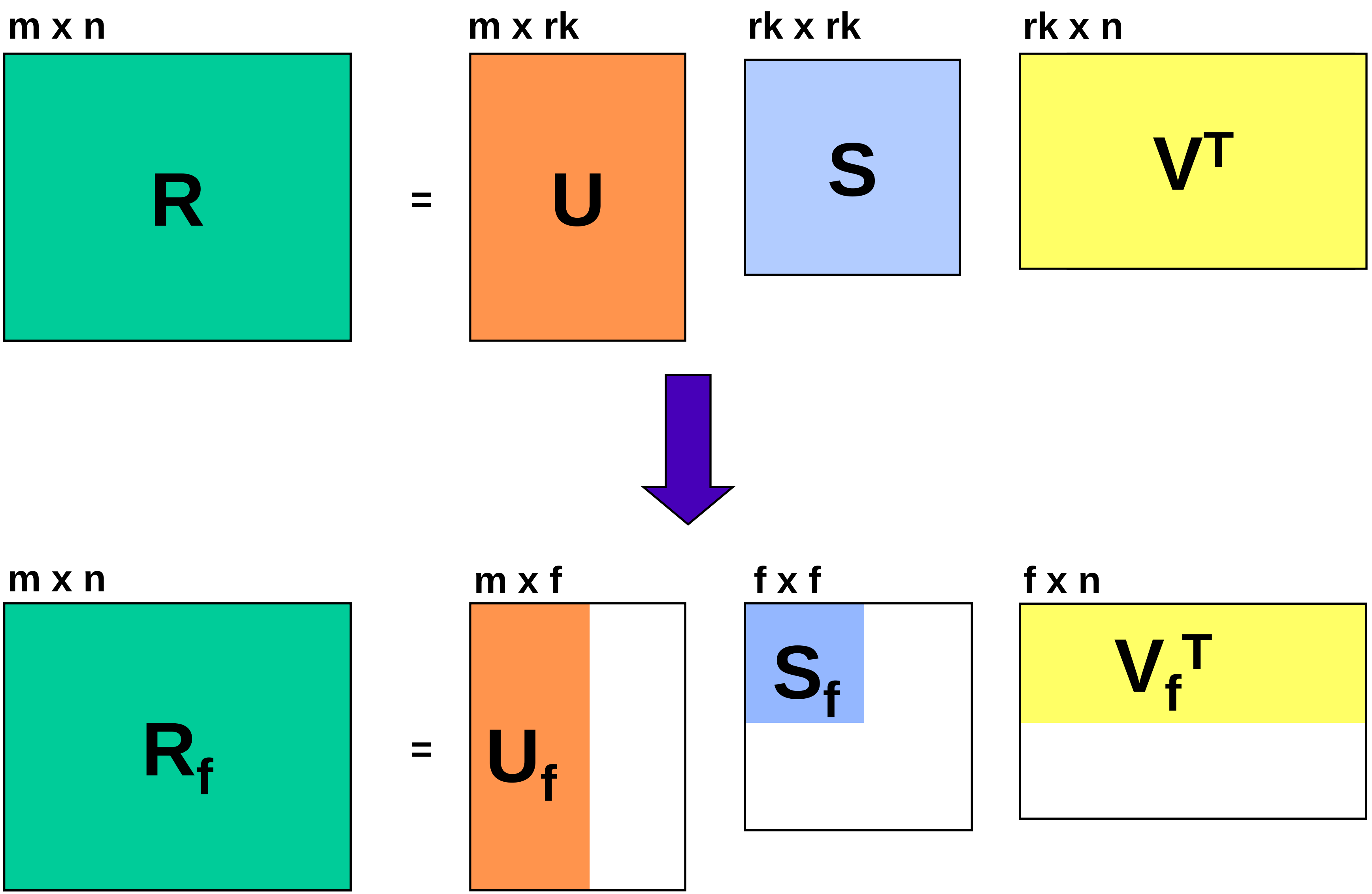
$U^T \cdot U = I$

$V^T \cdot V = I$

- We don't know the features!

- Error minimization (Gradient descent)
- **Matrix factorization**

Singular Value Decomposition



PureSVD

$$\hat{r}_{ui} = p_u^T q_i$$

$$R_{\cancel{f}} = U_{\cancel{f}} \cdot S_{\cancel{f}} \cdot V_{\cancel{f}}^T$$

User feature vector

Item feature vector

$$p_u = U \cdot \sqrt{S}$$

$$q_i = \sqrt{S} \cdot V^T$$

PureSVD : some steps over..

$$R = U \cdot S \cdot V^T$$

$U^T \cdot U = I$
$V^T \cdot V = I$

$$R \cdot (V) = U \cdot S \cdot V^T \cdot (V)$$

$$R \cdot V = U \cdot S$$

$$\boxed{u_u \cdot S = r_u \cdot V}$$

PureSVD: some nice properties

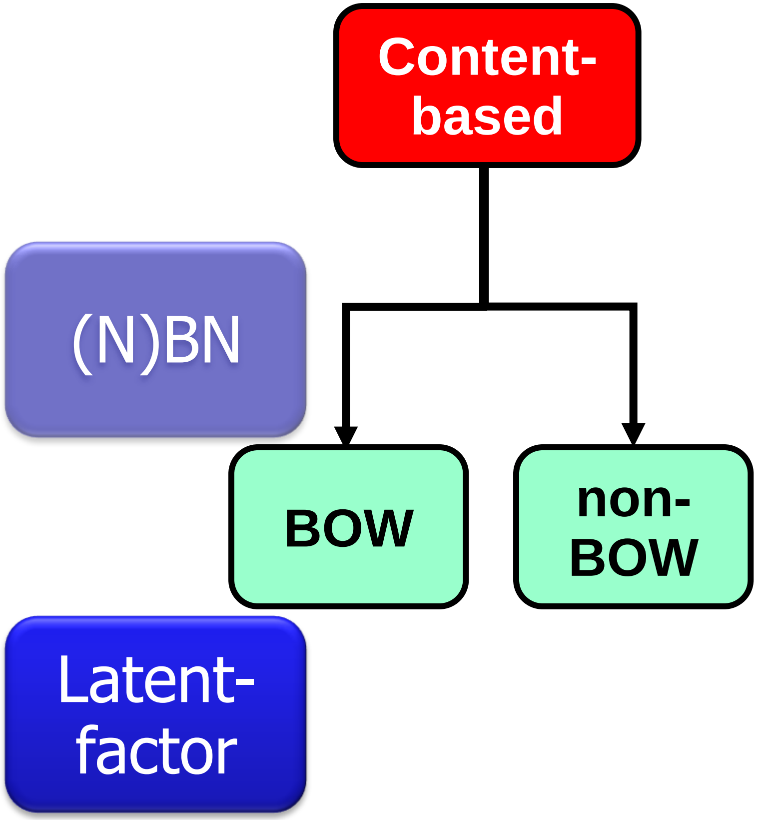
$$\hat{r}_{ui} = r_u \cdot V \cdot v_i^T$$

$$\hat{r}_u = r_u \cdot V \cdot V^T$$

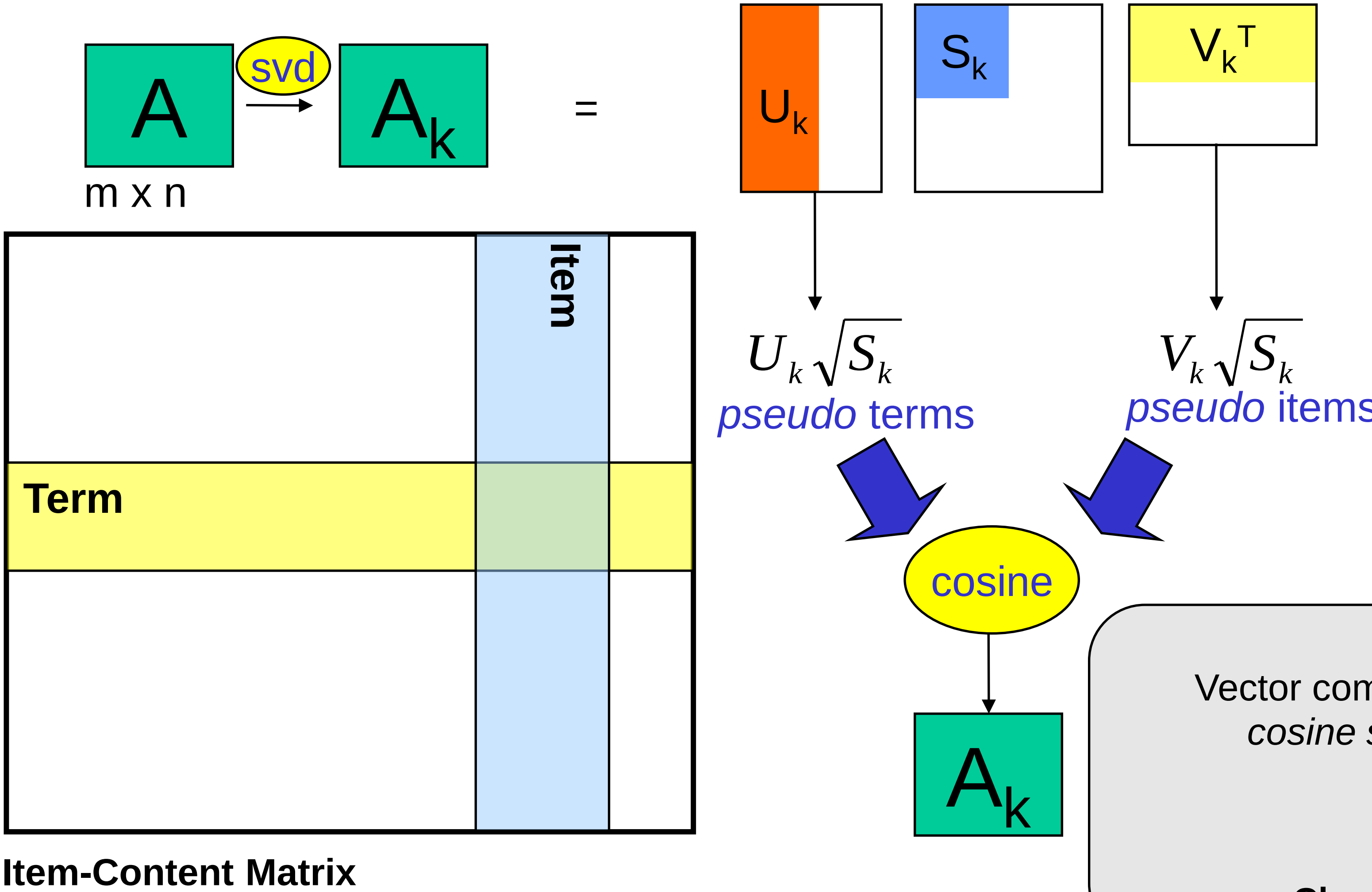
$$u_u \cdot S = r_u \cdot V$$

- **V·V$^{\mathbf{T}}$** is a **m**x**m** item-item matrix!
- PureSVD is a **model-based** algorithm!

Content-based: LSA



Content-based Filtering: LSA rank-reduced SVD



Vector comparison via *cosine similarity*

↓

Clustering

Further observations about IPTV recommenders

- Multi-language content
  - (e.g., Switzerland)
- New user problem
  - (user-based algorithms)
- New item problem
  - (all collaborative algorithms)
- Semantic problem
  - (e.g., *house* and *home*)

Agenda

- Introduction
- Recommender systems
  - Content-based and collaborative recommendations
  - Advanced algorithms
- Evaluation of recommender systems
- Case study: a recommender system for IPTV/VOD provider
- Recommender system demo

The multiple facets of recommender system evaluation

- Relevance
  - Error
  - Classification/accuracy
- Confidence
- “Explainability”
- Serendipity
- Diversity
- Novelty
- Coverage
- Training rate

Problem

- Relevance
  - Error
  - Classification/accuracy

Many works do not describe clearly the methods used for performance evaluation and model comparison

Different **dataset partition** methodologies and evaluation **metrics** lead to divergent results

The Netflix prize has **improperly** focused the research attention on
Hold-out
RMSE

Dataset partitioning

In order to evaluate the capability of a recommender system in suggesting items to users the URM (User Rating Matrix) must be partitioned into two different sets:

- **Training set**: used to generate the model of the system (it can be further divided into a model and a validation set for tuning some parameters of the algorithm)
- ***Test set***: used to test the model generated into the training step

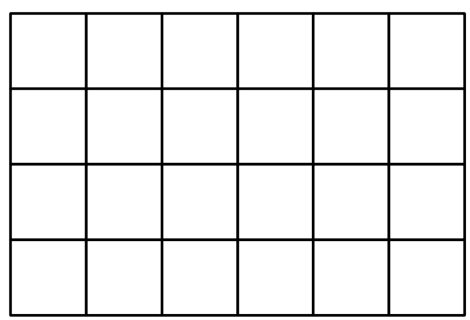
IMPORTANT!

Test set must be different and independent from training set

Partitioning techniques

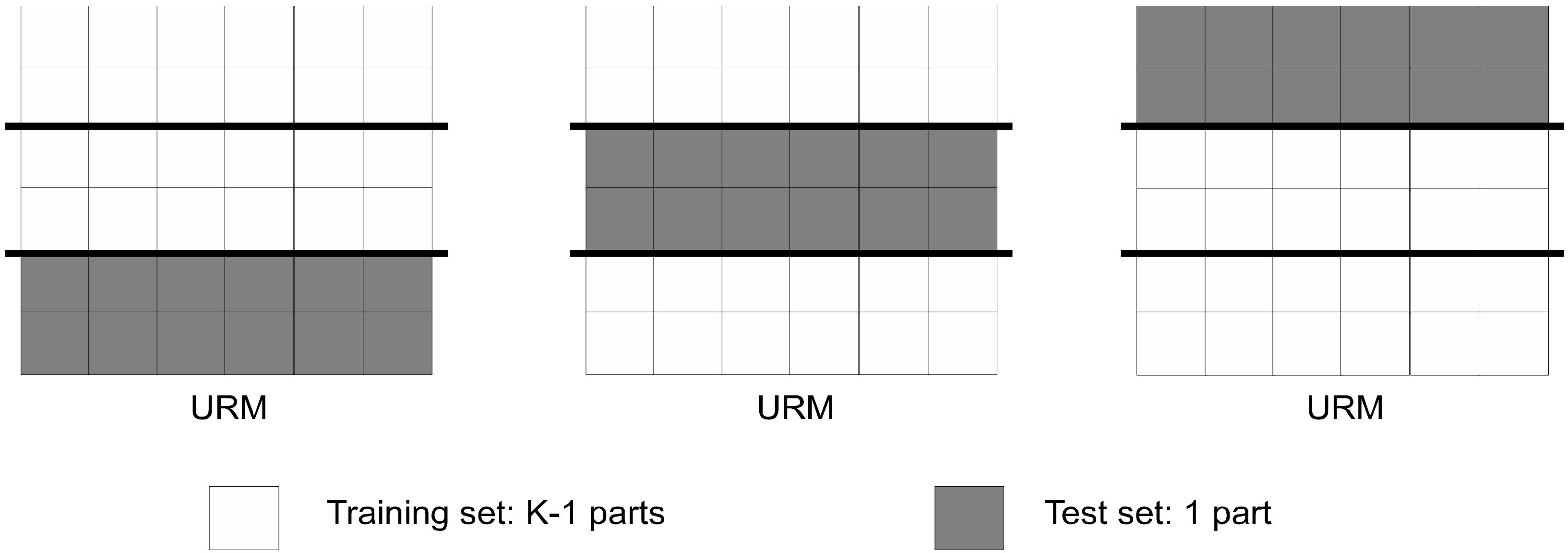
Different techniques can be used to partition the URM:

- **K-fold cross validation**
- **Leave-one-out**
- **Hold-out**



K-fold cross validation

The K-fold cross validation divides the users of the URM into K distinct partitions and at each step K-1 partitions are used to generate the model and the remaining partition si used for the tests.

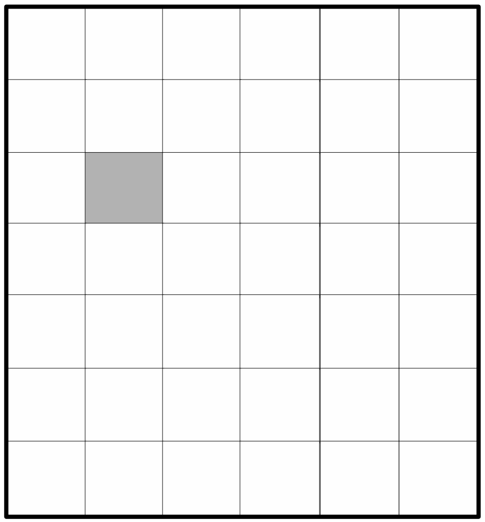


- The number of partitions suggested to have a robust test is 10.

- The tested users are unknown to the system because they are not used to build the model.

Leave-one-out

During the construction of the model we withhold one non null user rating of the URM (es: set it to 0), while the remaining ratings are used to generate the model.



URM

Training set: (m x n - 1) ratings

Test set: 1 rating

- The withheld rating represents the test set
- The procedure is repeated for all (or a sample of) the ratings

☺ Leave-one-out is attractive because almost all the ratings are available to build the model

☹ However it suffers from **overfitting** problems and it has a high computational complexity

Hold-out

A random set of ratings is withheld from the URM and it is used as test set. The remaining part of the ratings are used as training set.

- ☹ Can suffer the same overfitting problems as leave-one-out, because the tested users are not totally unknown to the model (especially if the test set is too small).
- ☹ Moreover this technique modifies the user profiles and this can lead to erroneous results (particularly perceptible in case of datasets with short user profiles.

RELEVANCE: Evaluation metrics

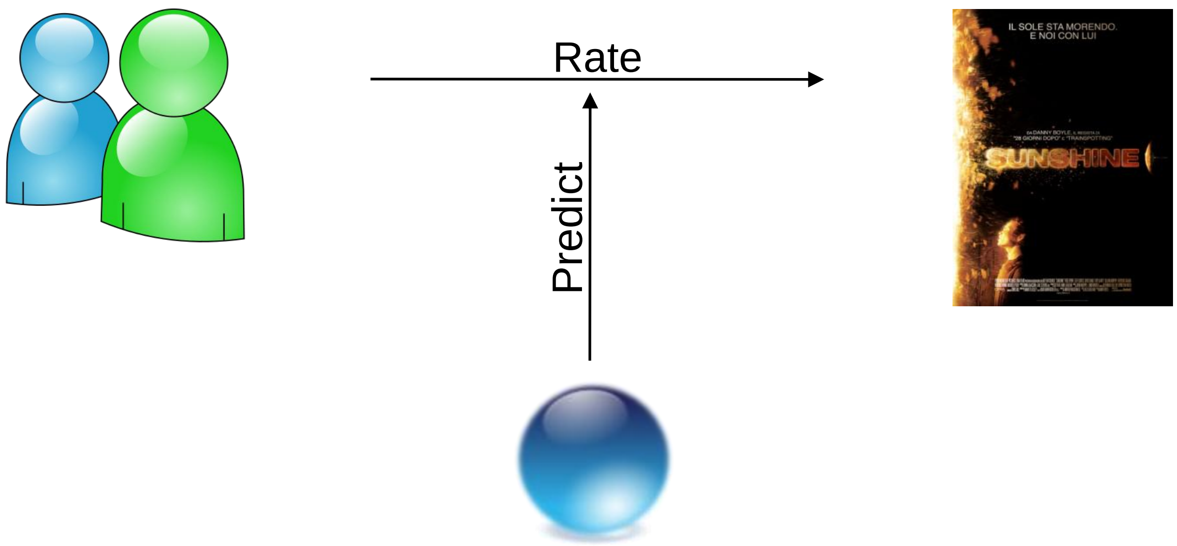
The **evaluation metrics** are techniques used to evaluate a recommender system and they are so important when we decide which recommender system is better to adopt for a specific application domain.

- Quality estimation

- Error metrics
  - RMSE
- Classification metrics
  - Recall, precision, f-measure

Error metrics

Error metrics evaluate the error made in the prediction of the rating given by user ***u*** for item ***i***.



IMPORTANT!

Error metrics can be applied **only** to explicit datasets since in the implicit URM we have no information about the level of preference of users!

Error metrics details

r_{ui} = true rating given by the user *p* for item i

$\hat{r}_{ui}$ = value predicted by the recommender algorithm

ERROR METRIC

$$f\left(r_{ui} - \hat{r}_{ui}\right)$$

An error metric estimates **the difference** between r_{ui} and $\hat{r}_{ui}$

Error metrics details

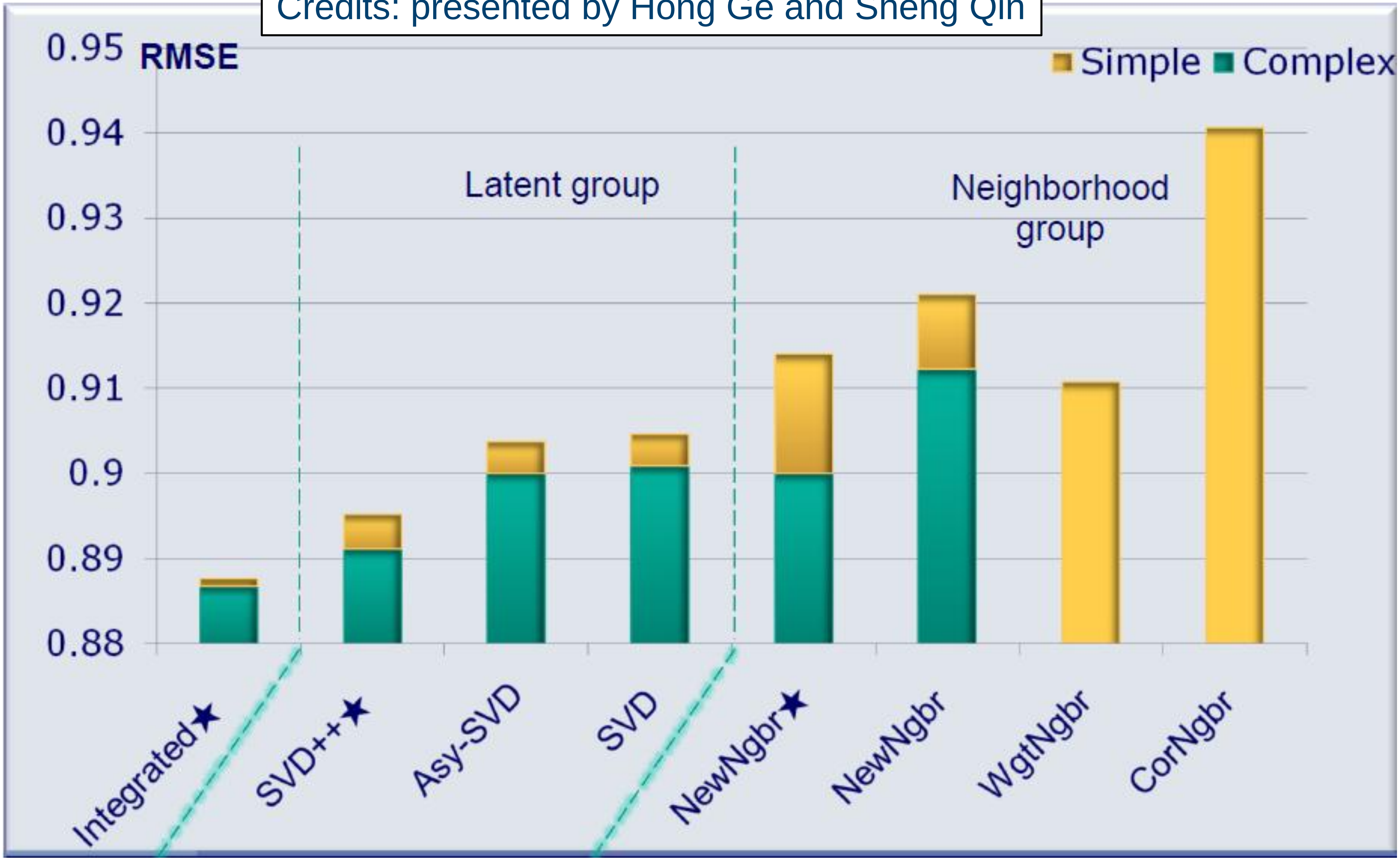
Among error metrics, the most used in the evaluation of reccommender systems are ***root mean squared error*** (RMSE), ***mean squared error*** (MSE) and ***mean absolute error*** (MAE).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{r}_{ui} - r_{ui}\right)^2 \qquad MAE = \frac{1}{n}\sum_{i=1}^{n}\left|\hat{r}_{ui} - r_{ui}\right|$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{u,i}\left(\hat{r}_{ui} - r_{ui}\right)^2}$$

From "**Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model**"- *Y. **Koren***, 2008 –

Credits: presented by Hong Ge and Sheng Qin



Error metrics PRO/CONS

Pro:

☺ RMSE, MSE and MAE are error metrics easy to compute but...

Cons:

☹ ...they are suitable **only** for explicit datasets and...

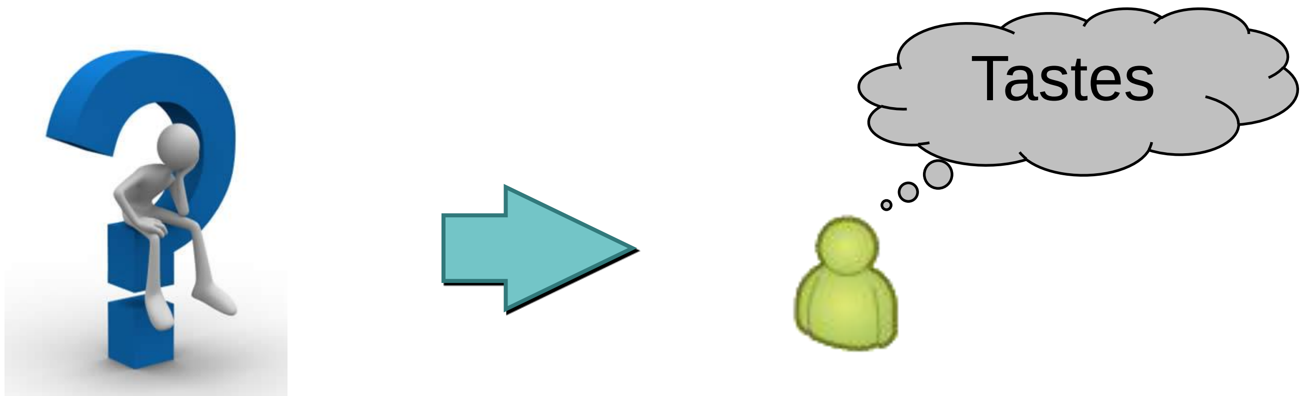
☹ ...these metrics can suffer the ***outlier problem*** so they are sensible to ratings with large prediction errors.

Note: with respect to MSE and RMSE, MAE has a more intuitive interpretation since it represents the average distance between true rating and predicted one.

Accuracy Metrics

Accuracy metrics (also called classification metrics) allow to measure the capacity of recommender systems in suggesting interesting or uninteresting items to the user.

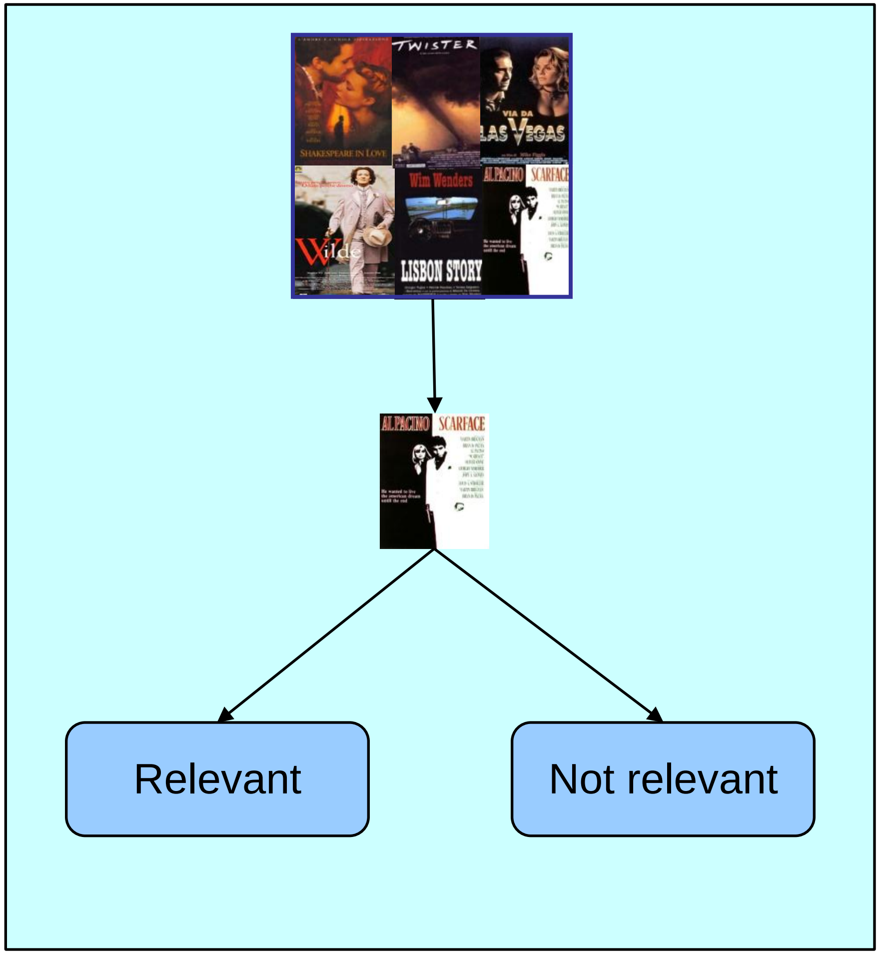
So the purpose of these metrics is not to predict the exact value of the ratings but they allow to understand whether a suggested item will be interesting or not to the user.

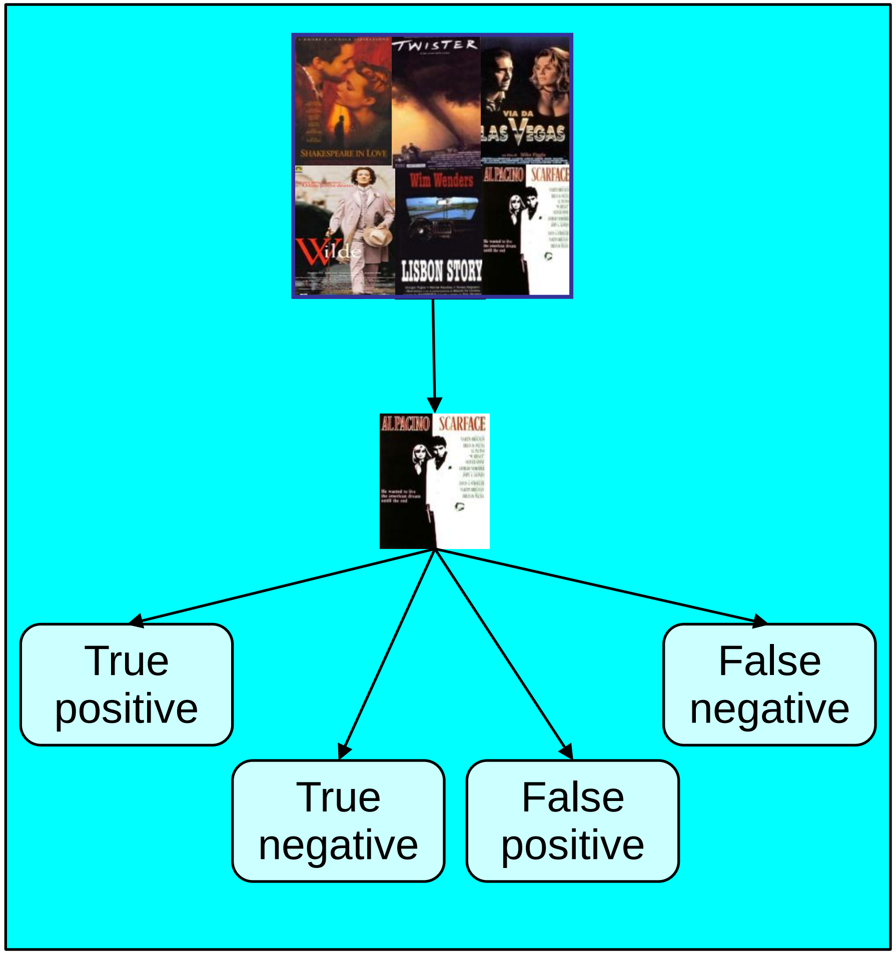


Classification of items into the URM

Classification of items for <u>information retrieval</u>

Classification of items according to <u>accuracy metrics</u>

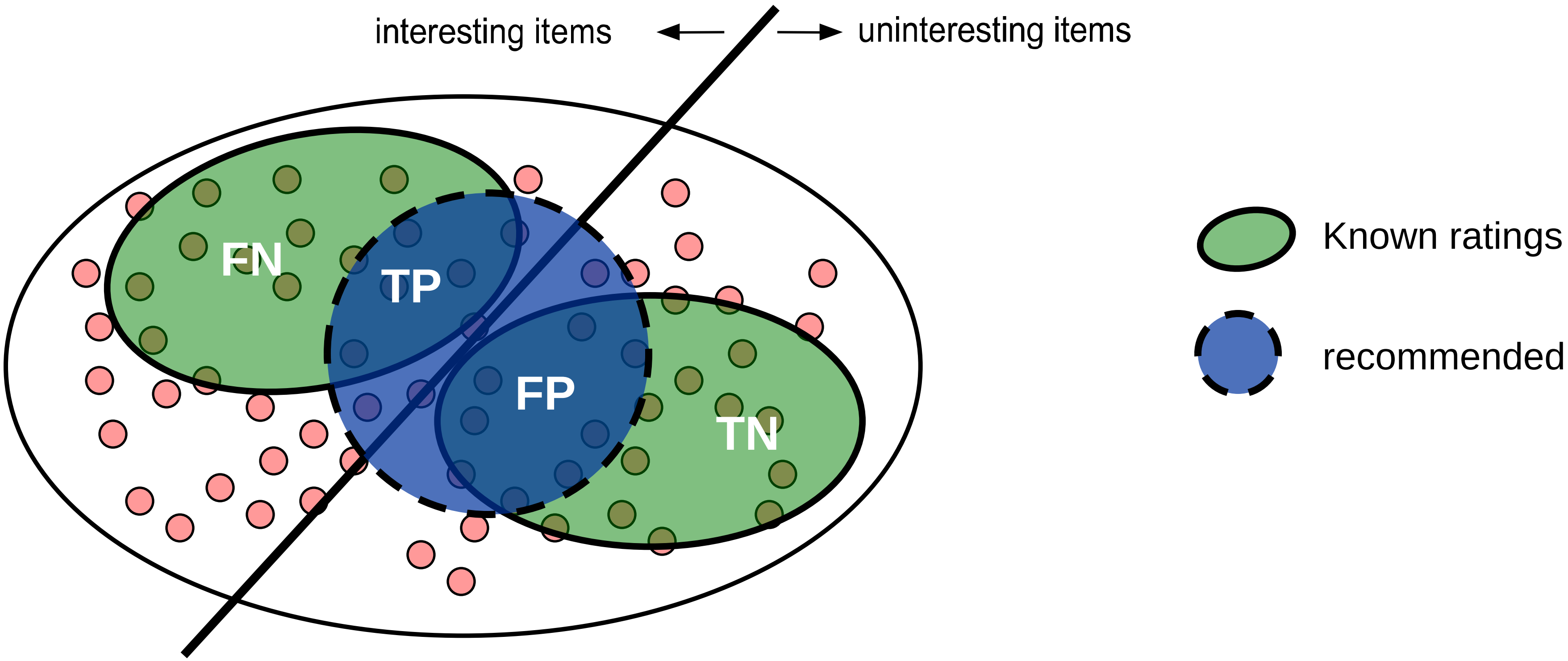




Classification details

- For a given user, items can be classified either in:
  - ***relevant***
  - **Non-*relevant***
- For a given user and a given **recommendation rule**, items can be classified either in:
  - ***True positive (TP)***: the system suggests a relevant item to the user
  - ***True negative (TN)***: the system doesn't suggest that the user dislikes
  - ***False positive (FP)***: the system suggests an item that the user dislikes
  - ***False negative (FN)***: the system doesn't suggest an item interesting for the user

Classification metrics



Recall and Fallout

The ***recall*** metric evaluates the capability of the system in recommending interesting items to the user (this metric is also referred to as true positive rate.

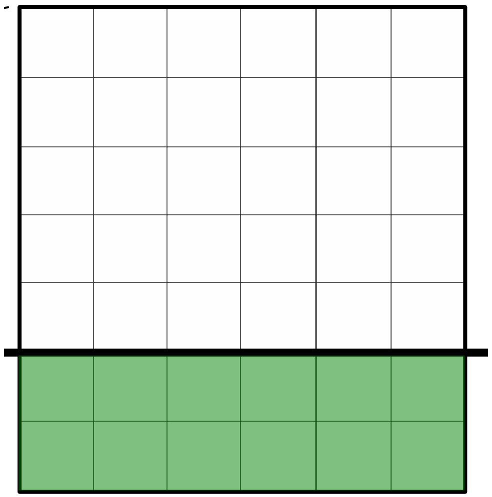
$$T = \frac{TP}{TP + FN}$$

On the other side, ***fallout***, also referred to as <u>false positive rate</u>, is a metric complementary to recall and measures how frequently the recommender system suggests non-interesting items to the user.

$$F = \frac{FP}{TN + FP}$$

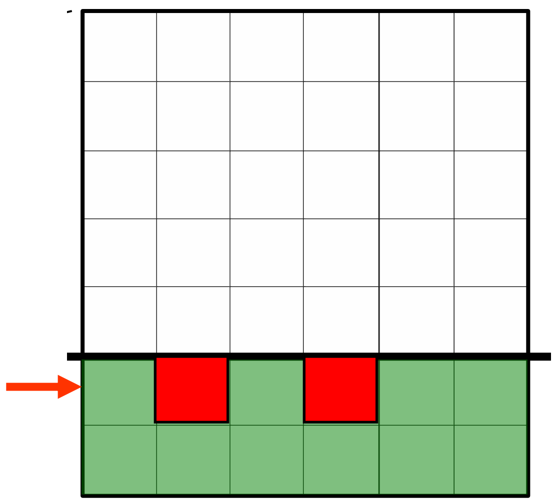
A possible methodology

- 10-fold cross validation
  - 9 folds: training set
  - 1 fold: test set
    - Leave-one-out



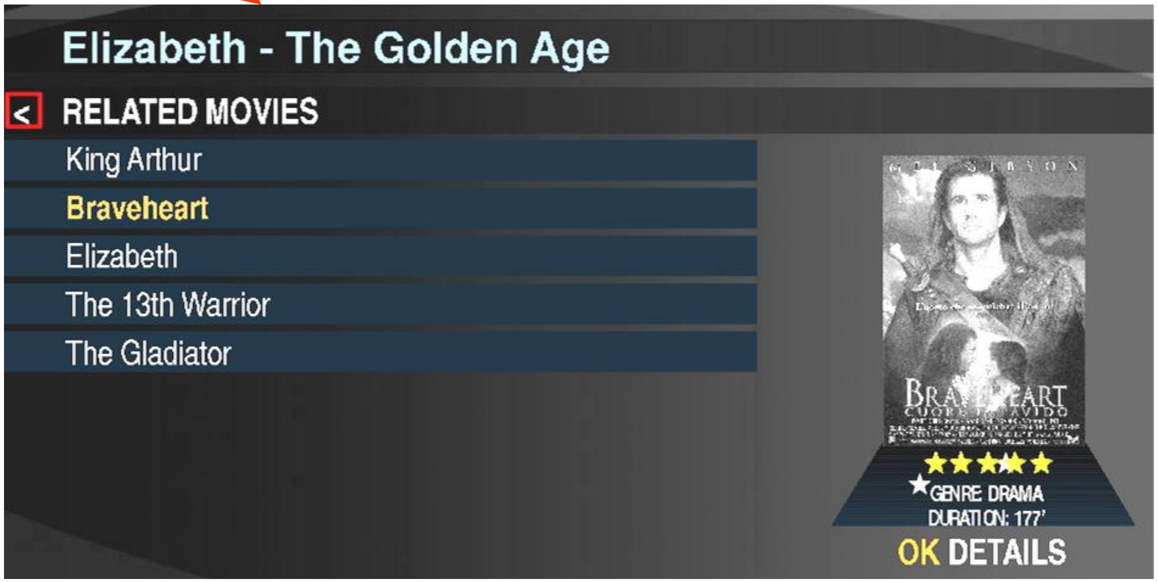
A possible methodology

- 10-fold cross validation
  - 9 folds: training set
  - 1 fold: test set
    - Leave-one-out → test item



Test item recommended in top-5?

- Positive rating → TP
- Negative rating → FP

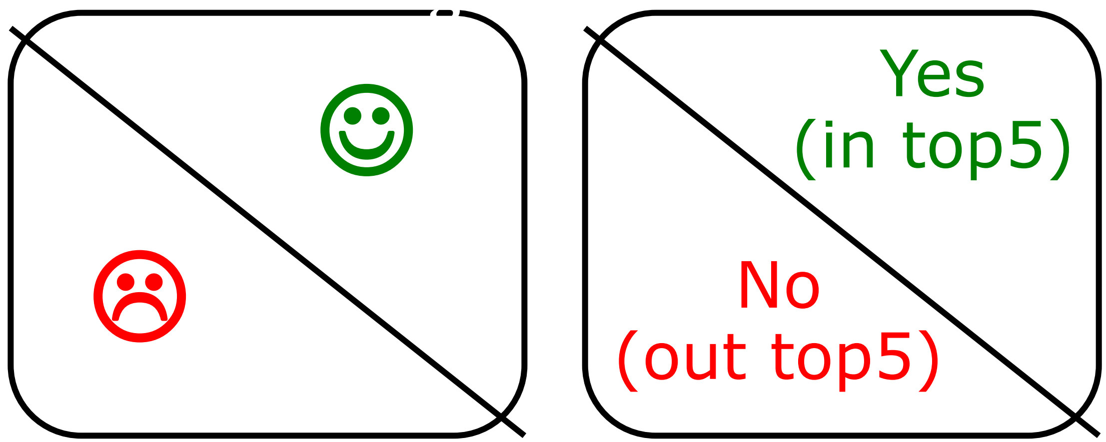


A possible methodology

- 10-fold cross validation
  - 9 folds: training set
  - 1 fold: test set
    - Leave-one-out → test item

Test item recommended in top-5?

- Positive rating → TP
- Negative rating → FP



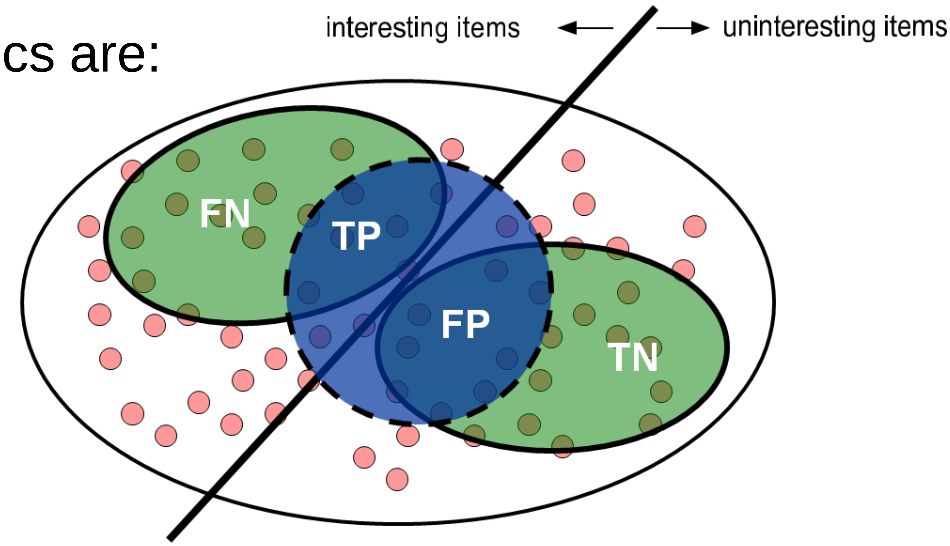
- Metrics
  - Recall
  - Precision
  - F-measure

Accuracy metrics: reformulation

The most popular accuracy metrics are:

- ***Recall (T)***
- ***Fallout (F)***
- ***Precision (P)***

And can be redefined as:



$$T = \frac{TP}{TP+FN} = \frac{\#hits^{+}}{|Test|}$$

$$F = \frac{FP}{TN+FP} = \frac{\#hits^{-}}{|tests|}$$

$$P = \frac{TP}{TP+FP} = \frac{\#hits^{+}}{N \cdot |Test|} = \frac{T}{N}$$

Where N is the number of recommended items.

Precision

The definition of precision is the following:

$$P = \frac{TP}{TP + FP} = \frac{T}{N}$$

...and when applied in the settings of recommender algorithms considers unrated items as non-interesting items.

So we assume that all recommended items for which we have no rating information are to be considered as ***non-relevant*** and ***they count negatively in computing the precision***.

Reducing error in precision metric

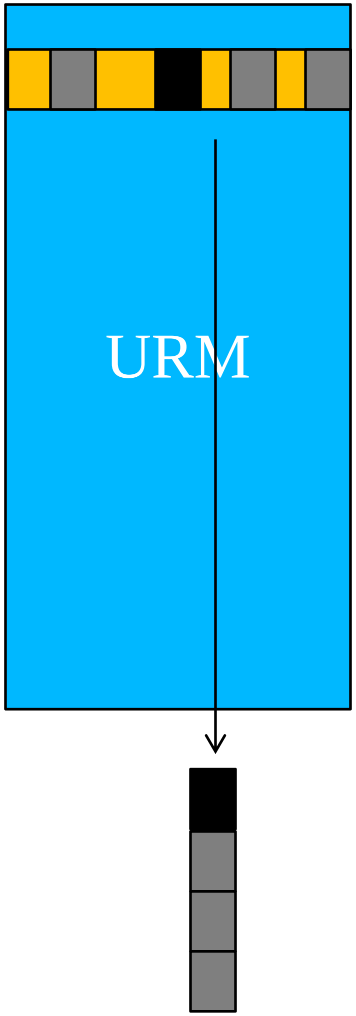
- One way to reduce the error in the estimate of the precision is to increase the number of ratings withheld from the user profile.
→ Moving from a Leave-one-out to and hold-out…

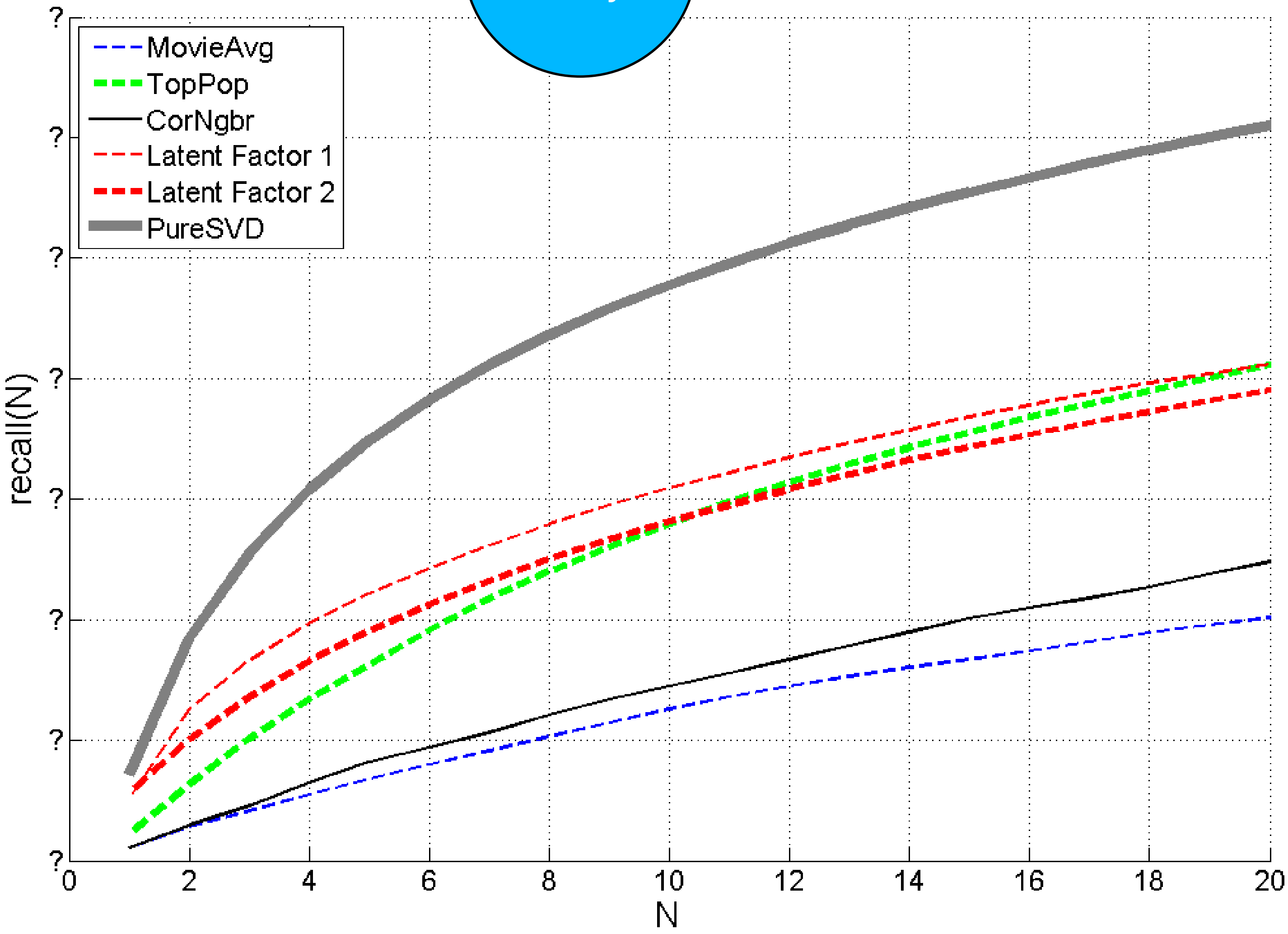
but

This implies reducing the number of ratings used to describe a user which negatively affects the quality of the recommender algorithm.

Another methodology

Ongoing study





Recall/fallout and precision: some considerations

- ***Recall,…*** are suitable in the evaluation of the global behavior of a recommender algorithm in top-N tasks!

- Both ***precision*** and ***recall*** heavily depend on the number of rated items per user and, thus, their values should not be interpreted as absolute measures but only to compare different algorithms on the same dataset.

- Measuring ***precision*** and recall is difficult because it is often unknown, for a certain user, how many relevant items there exist in the catalogue.

- Any catalogue probably contains a large number of items that meet a specific user taste. If that user has rated only a small percentage of such items a recommender algorithm might appear to have a low ***recall***, since the system may recommend unrated, relevant items (which are considered as irrelevant).

Netflix dataset: test user profile

Title	Rating
King Kong	5
The Village	5
Pocahontas	5
Men in Black II	5
Little Women	5
The Lord of the Rings: the Return of the king	1

Netflix dataset: BASELINE

Title	Rating
King Kong	5
The Village	5
Pocahontas	5
Men in Black II	5
Little Women	5
The Lord of the Rings: the Return of the king	1

Title
The Lord of the Rings: the Fellowship ... Ring
The Lord of the Rings: the Two Towers
Lost: Season 1
The Shawshank Redemption
Arrested Development

RMSE: 0.95
Recall: 1%
F-measure: 0.01

Netflix dataset: Item-based neighborhood

Title	Rating
King Kong	5
The Village	5
Pocahontas	5
Men in Black II	5
Little Women	5
The Lord of the Rings: the Return of the king	1

Title
Dinosaur Planet
Isle of Man TT 2004 Review
Character
Paula Abdul's Get Up & Dance
The Rise and Fall of ECW

RMSE: 1.6
Recall: 8%
F-measure: 0.16

Netflix dataset: PureSVD

Title	Rating
King Kong	5
The Village	5
Pocahontas	5
Men in Black II	5
Little Women	5
The Lord of the Rings: the Return of the king	1

Title
I Robot
Independence Day
Men in Black
Harry Potter and the Prisoner of Azkaban
The Day After Tomorrow

RMSE: 2.7
Recall: 17%
F-measure: 0.28

Metrics comparison summary

- **Error metrics**
  - Mean Square Error (MSE)
  - Root Mean Square Error (RMSE)
  - Mean Absolute Error (MAE)

☹ Only for explicit datasets

☹ Top-N recommender systems

- **Accuracy metrics**
  - Recall
  - Precision
  - Fallout

☺ Both implicit and explicit datasets

There is NOT a "universal" metric perfect for all application domains but we must choose the most suitable one for each situation.

User-perceived vs algorithmic quality

JUST PRELIMINARY RESULTS! (with few subjects)

88 screened subjects

USER-PERCEIVED QUALITY VS. ALGORITHMIC METRICS

Algorithm	Profile length	Statistical metrics		User-centered metrics		
				Mean feedback		Serendipity
		RMSE	*Recall*	*All movies*	*Novel movies*	
Collaborative	*Short*	1.76	5.71	3.63	2.55	14%
	Long	1.73	5.13	3.77	2.77	10%
Content-based	*Short*	2.47	3.77	3.17	2.75	46%
	Long	2.42	2.33	3.29	2.65	36%

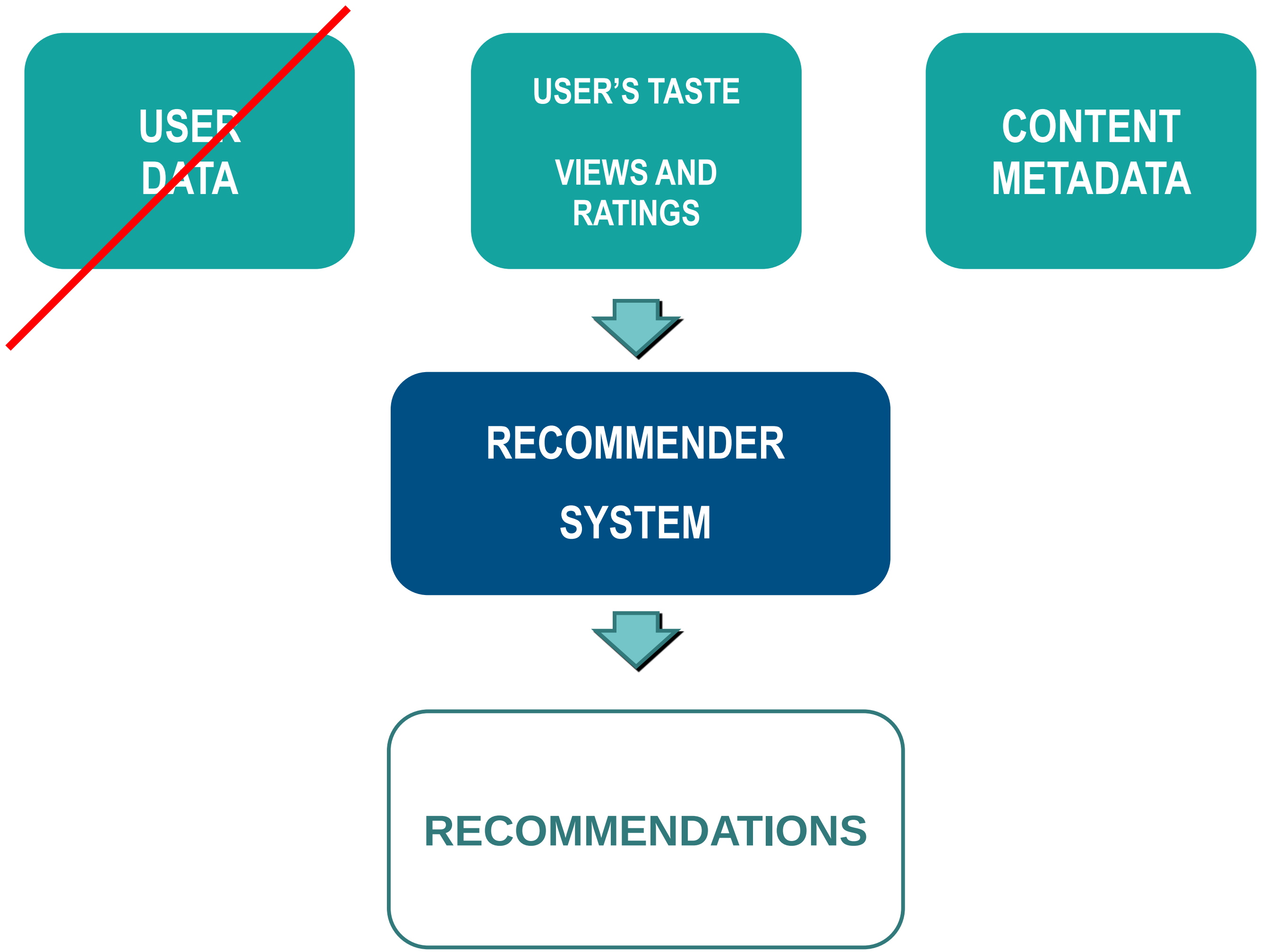
Agenda

- Introduction
- Recommender systems
  - Content-based and collaborative recommendations
  - Advanced algorithms
- Evaluation of recommender systems
- Case study: a recommender system for IPTV/VOD provider
- Recommender system demo

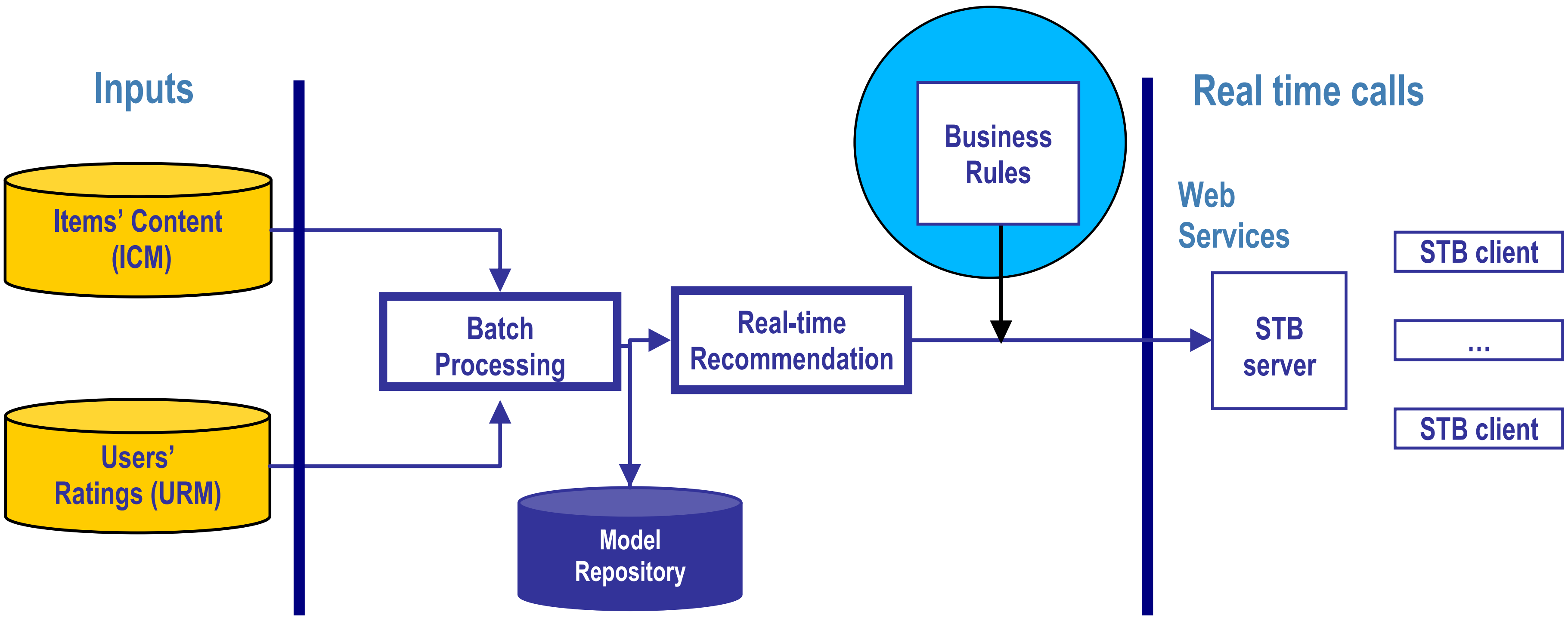
Large-scale European IPTV/VOD provider

- IP-based broadband TV (2001)
  - Hundreds of thousands of customers
  - Catalog of thousands of multimedia contents
- Recommender system (Content Wise) integrated
  - Small list of recommended items
    - Screen definition
    - Reduced navigation capabilities
  - Real-time constraints
    - Scalability (30000 recommendations per day)
  - Live boradcast channels
  - User identification
  - Quality of content information

Recommender System



Model-based architecture

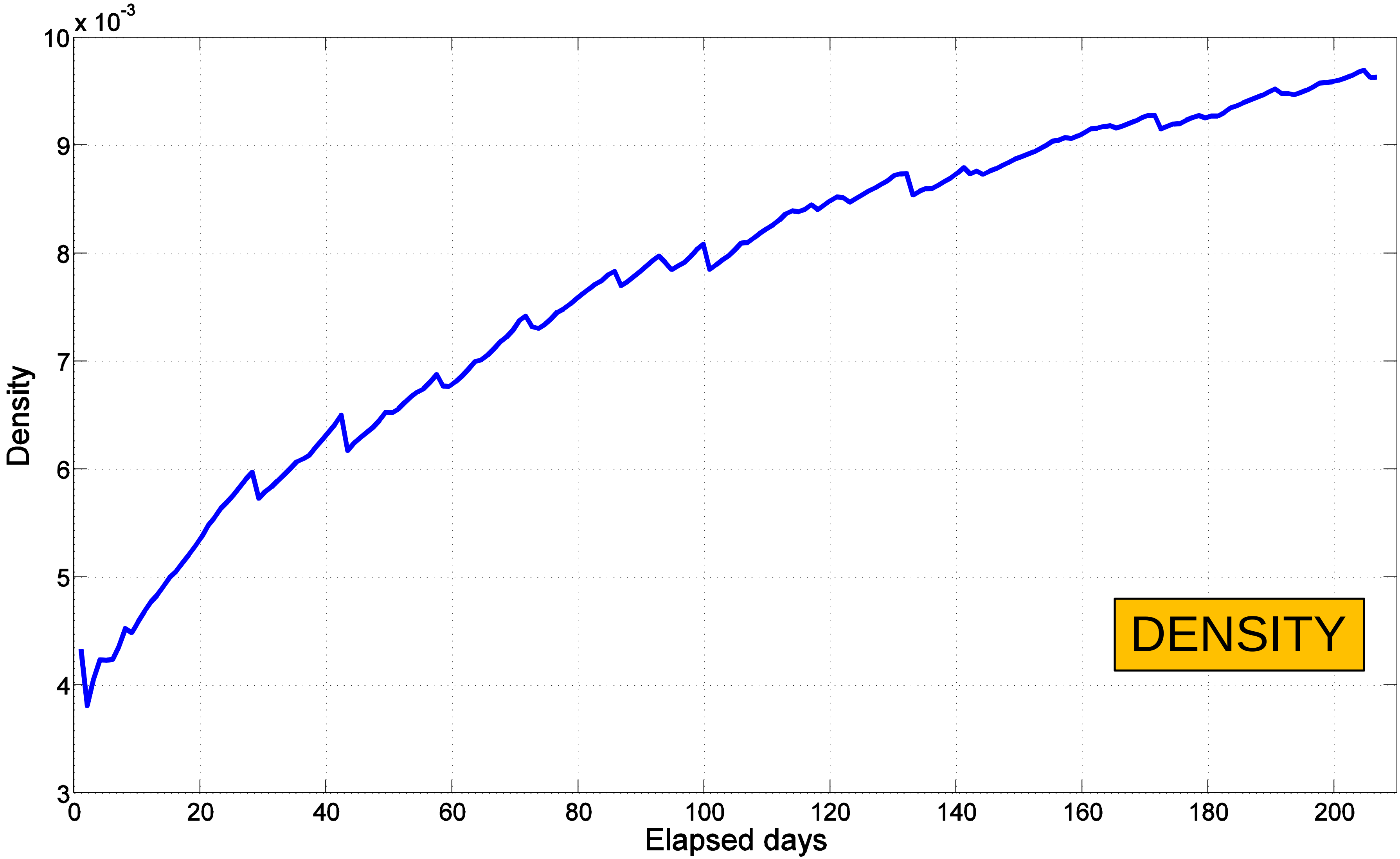


Proposed approach

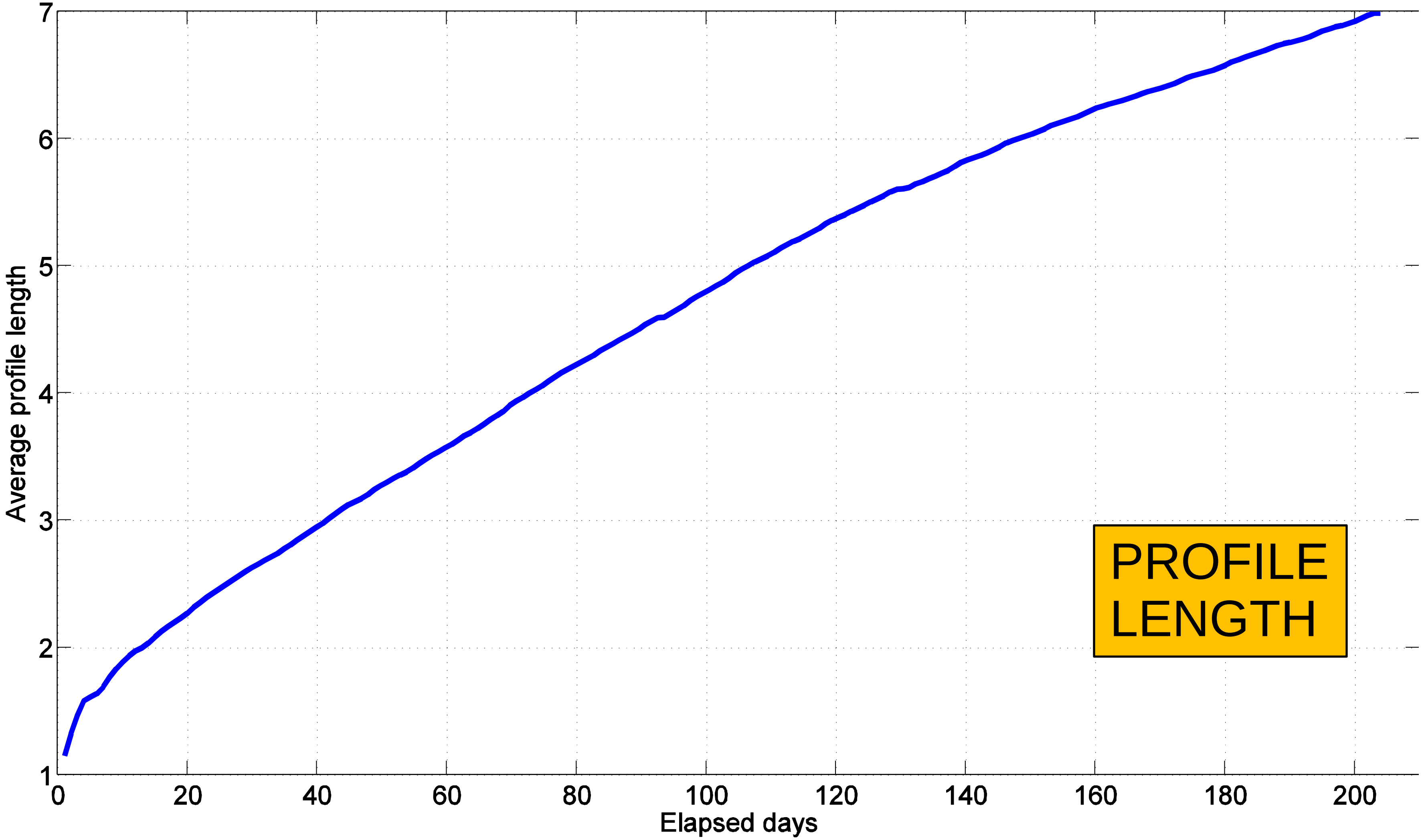
- ***Batch system***
  - Statistical analysis of the dataset
  - Definition of a number of models
  - Accuracy evaluation for different user profiles

- ***Run-time system***
  - User profile analysis
  - Selection of best candidate model
  - Recommendation

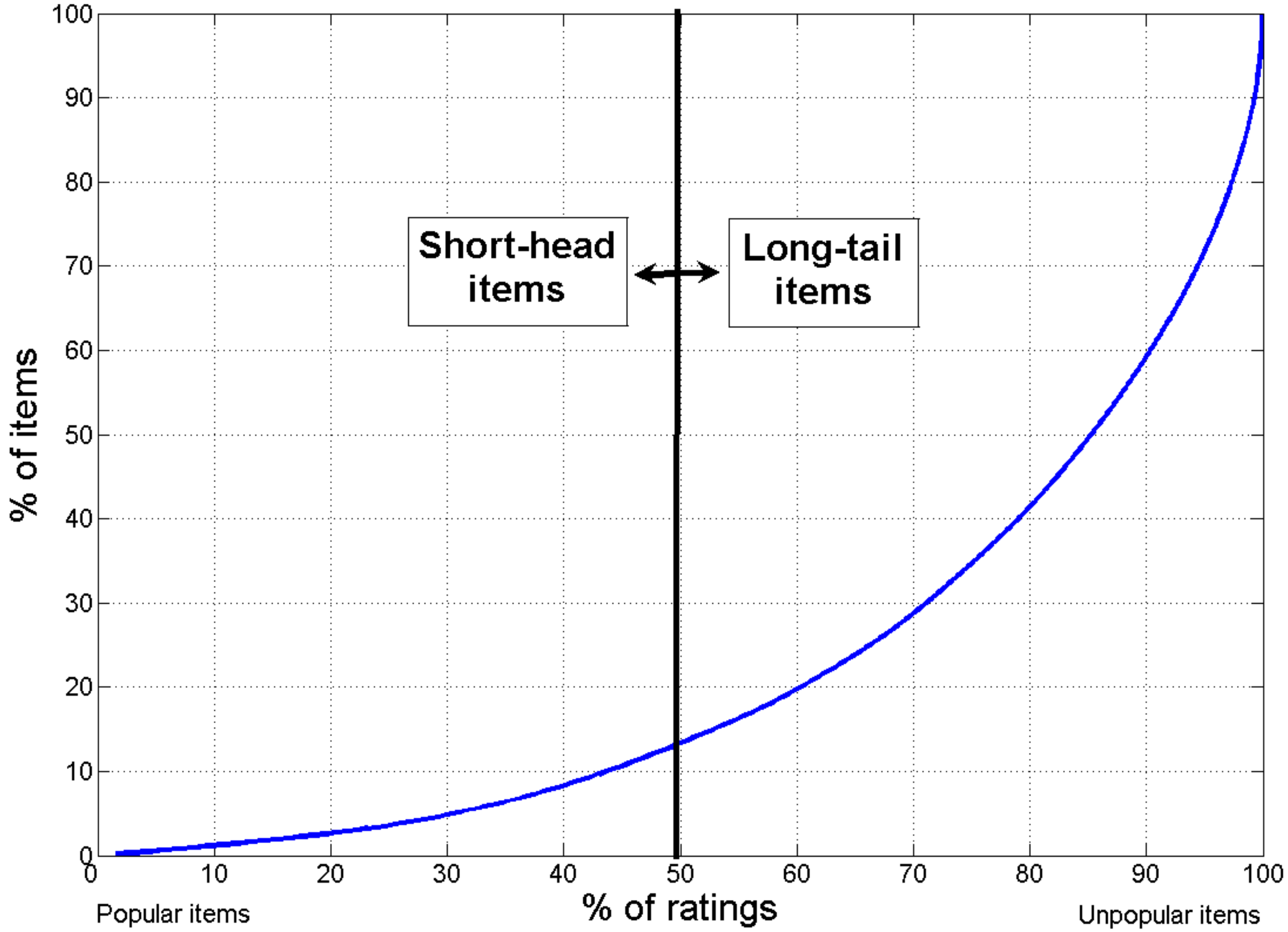
Some data..


x 10^-3
10
9
8
7
6
5
4
3
Density
0
20
40
60
80
100
120
140
160
180
200
Elapsed days
DENSITY

Some data..


7
6
5
4
3
2
1
0
20
40
60
80
100
120
140
160
180
200
Average profile length
Elapsed days
PROFILE LENGTH

Some data..



100
90
80
70
60
50
40
30
20
10
0
% of items
Short-head items
Long-tail items
0
10
20
30
40
50
60
70
80
90
100
Popular items
% of ratings
Unpopular items

Recommendation: relevance on VoD

Algorithm	Parameter	Recall 3 months	Recall 6 months
Item-based-CF	$k = 10$	16.8%	14.9%
	$k = 50$	18.7%	16.4%
	$k = 100$	**19.0%**	**16.6%**
	$k = 150$	18.8%	16.5%
SVD-CF	$l = 5$	15.1%	12.7%
	$l = 15$	12.6%	13.3%
	$l = 25$	10.9%	11.5%
	$l = 50$	9.3%	9.9%
	$l = 100$	6.3%	8.0%
LSA-CB	$l = 50$	1.9%	1.7%
	$l = 100$	2.3%	2.3%
	$l = 150$	2.4%	2.4%
	$l = 200$	2.5%	2.5%
Top-rated		12.2%	7.7%

Recommendation: relevance on VoD

Algorithm	Parameter	Recall non-top-10 3 months	Recall non-top-10 6 months	Recall non-top-50% 3 months	Recall non-top-50% 6 months
Item-based-CF	$k = 10$	14.0%	13.2%	7.7%	9.6%
	$k = 50$	**14.0**%	**13.8**%	6.8%	9.0%
	$k = 100$	13.8%	13.5%	6.2%	8.3%
	$k = 150$	13.5%	13.2%	6.1%	7.9%
SVD-CF	$l = 5$	6.6%	6.8%	0.7%	1.4%
	$l = 15$	11.5%	10.2%	1.2%	3.5%
	$l = 25$	12.6%	12.0%	2.2%	4.9%
	$l = 50$	11.4%	11.2%	4.8%	7.8%
	$l = 100$	7.6%	9.3%	**9.8**%	**11.8**%
LSA-CB	$l = 50$	2.1%	1.8%	1.8%	1.7%
	$l = 100$	2.3%	2.3%	2.0%	2.5%
	$l = 150$	2.5%	2.5%	2.1%	2.5%
	$l = 200$	2.6%	2.6%	2.2%	2.6%
Top-rated		0.4%	1.0%	0%	0%

Recommendation: relevance on VoD

	2 hours	24 hours	7 days
All	17.0%	19.8%	24.7%
Top 10	5.1%	7.0%	10.6%
Non-top 10	24.2%	27.6%	32.1%
Top 50%	9.4%	11.5%	16.2%
Non-top 50%	28.4%	32.2%	36.1%

Agenda

- Introduction
- Recommender systems
  - Content-based and collaborative recommendations
  - Advanced algorithms
- Evaluation of recommender systems
- Case study: a recommender system for IPTV/VOD provider
- Recommender system demo

References

- P.Cremonesi, Y. Koren, and R.Turrin, "Performance of Recommender Algorithms on Top-N Recommendation Tasks".
  To appear on Proc. of the 4th International Conference on Recommender Systems (RecSys), ACM, 2010, Barcelona, Spain.

- P.Cremonesi, and R.Turrin, "Time-evolution of IPTV Recommender Systems".
  Proc. of the 8th European Conference on Interactive TV and Video, ACM, 2010, Tampere, Finland.

- R. Bambini, P.Cremonesi, and R.Turrin, "A Recommender System for an IPTV Service Provider: a Large-Scale Production Environment".
  Book Chapter on Recommender Systems Handbook (P.B.Kantor, F.Ricci, L.Rokach, and B. Shapira), Springer, 2010.

- Herlocker, J.; Konstan, J.; Terveen, L. & Riedl, J. “Evaluating collaborative filtering recommender systems” .
  ACM Transactions on Information Systems (TOIS), ACM Press New York, NY, USA, 2004, 22, 5-53

- Adomavicius, G. & Tuzhilin, A. “Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions”.
  Knowledge and Data Engineering, IEEE Transactions on, 2005, 17, 734-749

References

- Adomavicius, G. & Tuzhilin, A. “Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions”.
  Knowledge and Data Engineering, IEEE Transactions on, 2005, 17, 734-749

- P.Cremonesi, and R.Turrin, "Analysis of cold-start recommendations in IPTV systems".
  Proc. of the 3rd International Conference on Recommender Systems (RecSys), ACM, 2009, New York, USA

- P.Cremonesi, E. Lentini, M. Matteucci, and R.Turrin, "An evaluation methodology for recommender systems".
  Proc. of the 4th International Conference on Automated Solutions for Cross Media Content and Multi-channel Distribution (AXMEDIS), IEEE, 2008, Firenze, Italy.

- Bell, R. M. & Koren, Y.”Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights”.
  7th IEEE Int. Conf. on Data Mining, IEEE International Conference on Data Mining (ICDM'07), 2007, 43-52

- Koren, Y. “Factorization meets the neighborhood: a multifaceted collaborative filtering model”.
  KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2008, 426-434

- E. Campochiaro, R. Casatta, P.Cremonesi, and R.Turrin, "Do metrics make recommender algorithms?".
  Proc. of the International Symposium on Mining and Web (MAW), IEEE, 2009, Bradford, UK.