INFO-H-515:
BIG DATA ANALYTICS

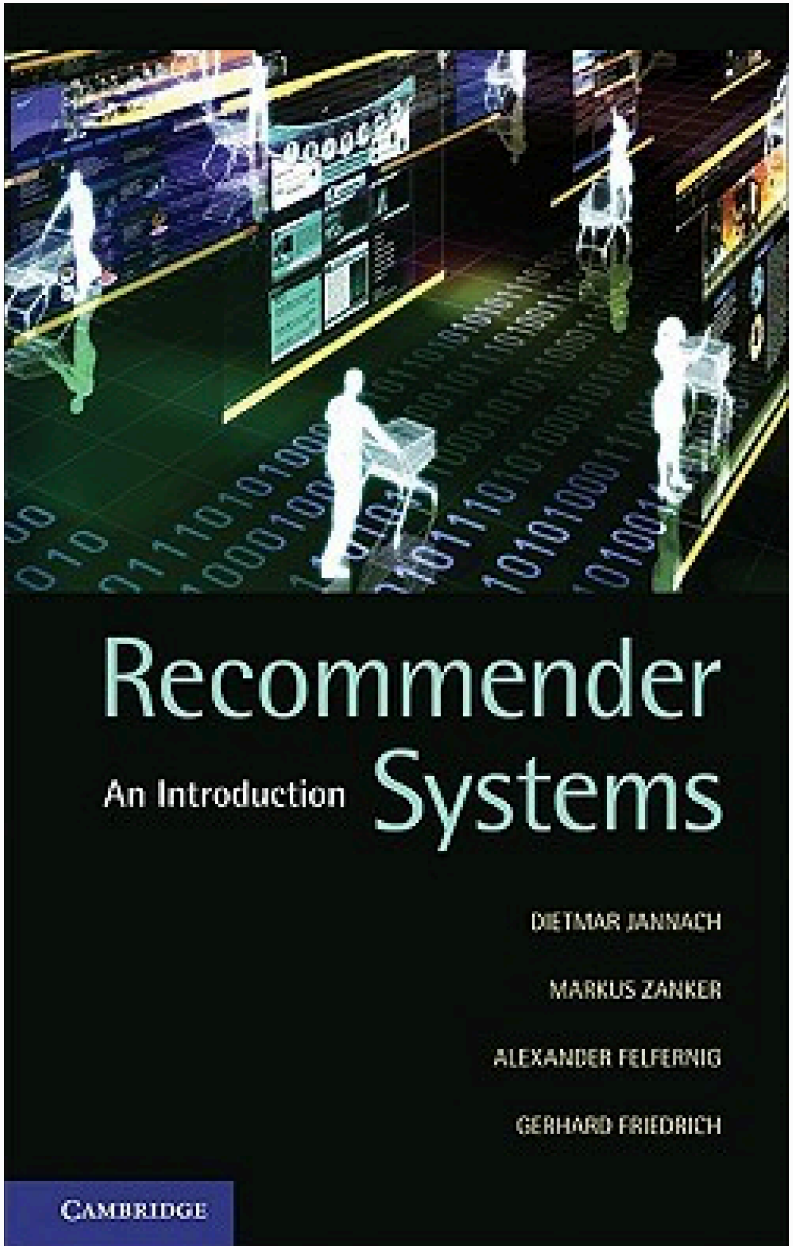
Recommendation engines

Gianluca Bontempi
Machine Learning Group
Boulevard de Triomphe - CP 212
http://mlg.ulb.ac.be

RECOMMENDER SYSTEMS

- Goal: support users in (online) decision making (products, movies, news). Provide easily accessible, high quality and personalized recommendation to a large user community
- Difference between physical and online retailers: physical retailers can have only a subset of products but show all of them. Online retailers may keep all the products but show only a part of them (need of recommendation systems).
- Issues related to volume and velocity of data
- Need of user model or user profile (implicit vs explicit approach)
- Community related information
- Link with information retrieval and filtering
- Issues of explanation and evaluation

DISCLAIMER



Much of the following material is taken from the book [1] .

MAIN APPROACHES

3 main approaches (and their hybrid combination)

1. Collaborative
2. Content-based
3. Knowledge-based

Two main entities: users and products (or items) for which users have preferences

COLLABORATIVE FILTERING

- Idea: if users shared the same interests in the past, they will do it also in the future
- It relies only on past user behavior (e.g. previous transactions or product ratings) and uses relationships between users and interdependencies among products
- Largely used and successful approach (online retail sales)
- Issues:
  - How to find users with similar tastes and what does it mean similar?
  - What about users for which no buying history is available?
  - What about recent items (not bought yet)?
- (+) content agnostic
- (+) no need of item description
- (+) large number of benchmark datasets
- (-) cold start issue

COLLABORATIVE FILTERING

- Training set: a matrix of user/product ratings
- Query: a user and a not-yet rated product
- Outputs: numerical prediction of the user rating of an product or list of most relevant products for a user.
- 3 types of CF
  1. User-based nearest neighbor recommendation: given a rating database and the ID of the current user, identify other users with similar preferences in the past. Use neighbors ratings for predicting the rating of the target product
  2. Item-based nearest neighbor recommendation: identify similar products and use the ratings of the same user for similar products
  3. Latent factor models based on factorization: it aims to explain the ratings by characterizing both items and users on a certain number k of factors inferred from the ratings patterns. Factors may be interpretable (comedy vs action, for kids vs adults) or not.

RATINGS (OR UTILITY) MATRIX

Given a set of n_u users and a n_p products, we need a matrix (not necessarily full) R of size $n_u \times n_p$ of ratings

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

Given the rating history of Alice and 4 Users, we wish to predict her rating for Item5.
We can use information about users to recommend products and about products to recommend users.

CHALLENGES OF NEAREST-NEIGHBOR APPROACHES

- Volume of data (millions of users and items)
- Similarity metric: Jaccard distance (ignore the ratings value), Pearson correlation, Spearman rank, cosine similarity, Euclidean distances?
- How to consider missing values in the distance? should rating be normalized (e.g. subtracting the average rate of each user)?
- Weighting of items: variance (e.g. give more importance to more controversial ones), significance (e.g. take into account the number of user ratings)
- Lack of symmetry between items and users: items typically belong to a specific cluster (e.g. a genre) while users may have several clusters of interest (e.g. hard rock and classical music).
- Number of neighbors
- Preprocessing: precomputing of neighborhood matrices
- Missing data (is a missing sample informative?), sparse rating matrix (indirect associations, default voting)

CHALLENGES OF NEAREST-NEIGHBOR APPROACHES

- Clustering of users? of items?
- Implicit (e.g. browsing/search/purchase/clicking history) vs explicit ratings (e.g. Netflix)
- Granularity of ratings scale
- Cold-start problem
- Nonstationarity
- Memory-based vs model-based

REMARKS

- Ideally, given the rating matrix the goal would be to predict all the blanks
- In practice, it is not really necessary to predict every blank entry
- It is only necessary to discover some entries that are likely to be high
- Sometimes, it is enough to find a large subset of blank entries with the highest ratings

MATRIX FACTORIZATION

- Successful approach in the 1M $ Netflix competition
- Rating matrix as the product of two long and thin matrices.
- They derive a set of latent (hidden) factors from the rating matrix and characterize both users and items with such factors
- Note that those factors are not necessarily interpretable
- According to the SVD theorem the $[n_u, n_p]$ rating matrix can be decomposed as follows

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

  where $U[n_u, k]$ and $V[n_p, k]$ are the left and the right singular vectors and the diagonal of $\Sigma[k, k]$ are called singular values.
- By retaining only the most important singular values (e.g. $k = 2$) we can associate to each user and each item a small vector of values
- Each query point has to be projected in the compressed space before being used for prediction
- Good compromise between scalability and accuracy

EXAMPLE FROM BOOK

	Matrix	Alien	Star Wars	Casablanca	Titanic
Joe	1	1	1	0	0
Jim	3	3	3	0	0
John	4	4	4	0	0
Jack	5	5	5	0	0
Jill	0	0	0	4	4
Jenny	0	0	0	5	5
Jane	0	0	0	2	2

Rank 2 matrix with $n_u = 7$ and $n_p = 5$. Two main concepts in the rating scheme: science-fiction and romance movies [2].

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 2 & 2 \end{bmatrix}}_{M} = \underbrace{\begin{bmatrix} .14 & 0 \\ .42 & 0 \\ .56 & 0 \\ .70 & 0 \\ 0 & .60 \\ 0 & .75 \\ 0 & .30 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} .58 & .58 & .58 & 0 & 0 \\ 0 & 0 & 0 & .71 & .71 \end{bmatrix}}_{V^{\mathrm{T}}}$$

Matrix U connects people to concepts (boys like science-fiction) and matrix V connects items to concepts (the first three movies belong to the concept one). Matrix Σ gives the strength of the concept (related to the number of samples)

SVD

$$\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n_p} \\ r_{21} & r_{22} & \cdots & r_{2n_p} \\ \vdots & \vdots & \ddots & \\ r_{n_u1} & r_{n_u2} & & r_{n_un_p} \end{bmatrix} = \begin{bmatrix} u_{11} & \cdots & u_{1k} \\ u_{21} & \cdots & r_{2k} \\ \vdots & \ddots & \\ u_{n_u1} & & u_{n_uk} \end{bmatrix} \begin{bmatrix} s_{11} & \cdots & 0 \\ 0 & \cdots & 0 \\ \vdots & \ddots & \\ 0 & & s_{kk} \end{bmatrix} \begin{bmatrix} v_{11} & \cdots & v_{1n_p} \\ v_{21} & \cdots & v_{2n_p} \\ \vdots & \ddots & \\ v_{k1} & & v_{kn_p} \end{bmatrix}$$

- Matrix U encodes users while matrix V encodes items
- Each user and each item is represented in a k dimensional space.
- Suppose we have a new user (represented by a rating vector x of size $[1, n_p]$) and we want to situate it in the latent compressed space. We can do it by the transformation

$$y = xV\Sigma^{-1}$$

which returns a vector y of size $[1, k]$

CHALLENGES OF MATRIX FACTORIZATION

- assessment of quality of decomposition (e.g. which cost function to measure the distance between R and $U\Sigma V$?)
- computational complexity of SVD decomposition is $O(n_p^2 n_u + n_u^3)$ is not manageable in very large dimensional spaces
- iterative minimization process
- sparseness: conventional SVD is undefined when knowledge about the matrix is incomplete
- cost of imputation, risk of adverse impact of imprecise imputation,
- overfitting
- choice of the number k of singular values,

LATENT FACTORS MODEL

- Both users and items are mapped to a joint latent factor space of dimensionality k such that user-item interactions are modeled as inner-product

$$R = UP^T$$

- Each item j is associated to the vector $P_j \in \mathbb{R}^k$ and each user i to the vector $U_i \in \mathbb{R}^k$ such that

$$r_{ui} \approx U_i P_j^T$$

.

- Let U be the $[n_u, k]$ user matrix and P the $[n_p, k]$ item factor matrix. The optimal value of these matrices minimizes the quantity

$$\sum_{i,j} \left((r_{ij} - U_i P_j^T)^2 + \lambda(\|U_i\|^2 + \|P_j\|^2) \right) = \sum_{i,j} \left((e_{ij})^2 + \lambda(\|U_i\|^2 + \|P_j\|^2) \right)$$

where the sum is limited to observed r_{ij} terms and the λ parameter acts as a regularizing term (i.e. to avoid overfitting).

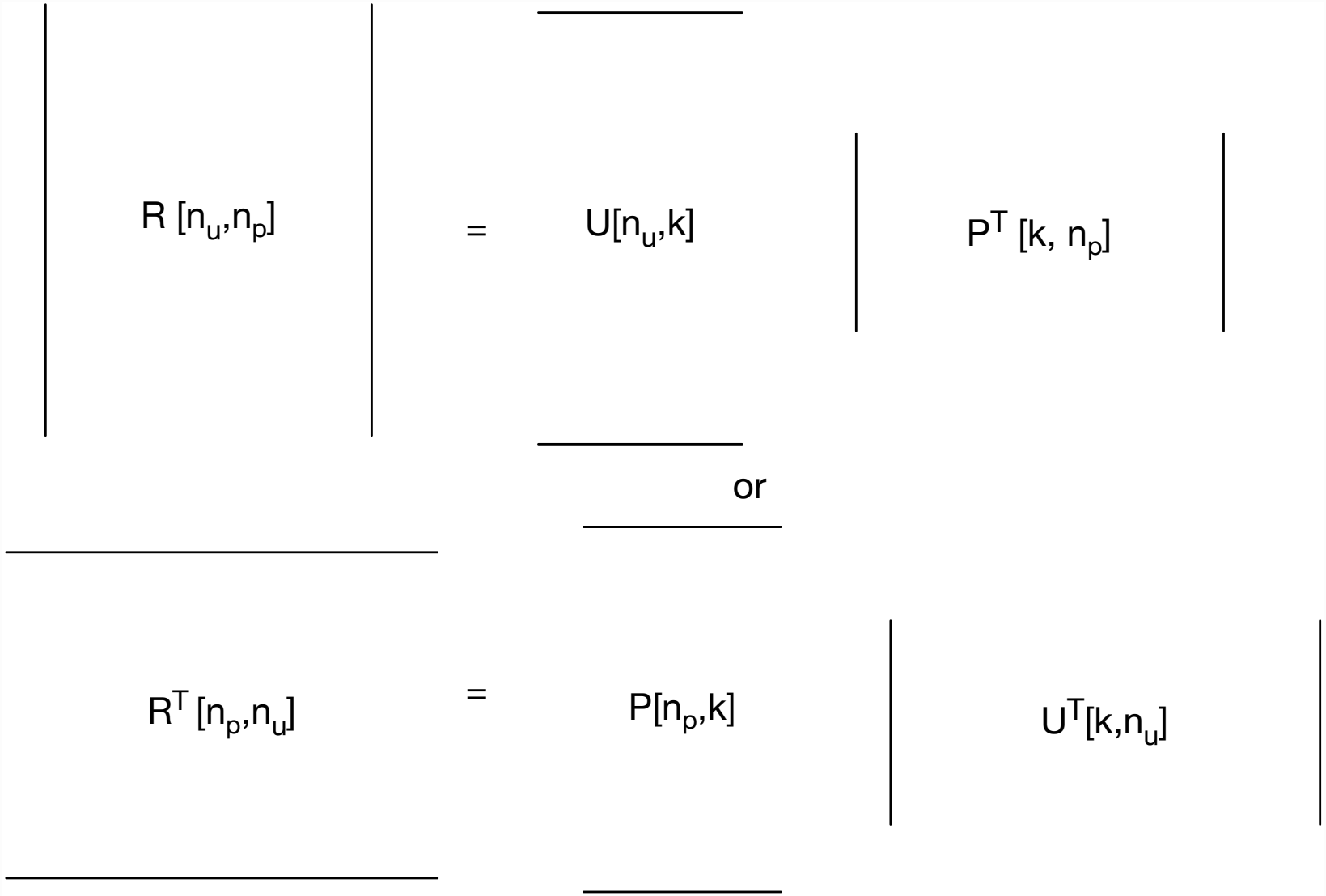
LATENT FACTORS MODEL FITTING

Note that the objective cost is non convex (because of the $U_i P_j^T$) term and then NP-hard to minimize.
Two solutions of the minimization problem

1. Alternating least squares: it works by iteratively solving a series of least-squares regression problems. In each iteration, one of the user or the item-factor matrices is treated as fixed, while the other one is updated using the fixed factor (input) and the rating data (output). This process continues until it has converged. ALS is powerful, achieves good performance, can be massively parallelized
2. Stochastic gradient descent.

Recommandations:

- perform a preprocessing normalization step (e.g. first remove average user rating and then average item rating).
- run several iterations with different initial conditions and average the results.


R [n_u,n_p] = U[n_u,k] P^T [k, n_p]
or
R^T [n_p,n_u] = P[n_p,k] U^T[k,n_u]

MULTI-OUTPUT LEAST SQUARES

- Let us consider a multi-input multi-output (or multi-target) regression problem where N is the number of observations, n is the number of inputs and m is the number of outputs.
- In matrix form

$$Y = XB + E$$

  where Y is the $[N, m]$ output matrix, X is the $[N, n]$ input matrix, E is the errors matrix $[N, m]$ and B is the parameter matrix of size $[n, m]$.
- If we make the hypothesis of uncorrelated errors, the least-squares estimation has the same form as the single output case

$$\hat{B} = (X^T X)^{-1} X^T Y$$

ALS (MATRIX FORM)

Initialize $U_{(0)}$ and $P_{(0)}$
Repeat until convergence

1. Use of the relationship $R^T = P_{(t)}U_{(t)}^T$. We set $Y = R^T$ ($[n_p, n_u]$), $X = P_{(t)}$ ($[n_p, k]$) and $B = U_{(t)}^T$ ($[k, n_u]$).

$$U_{(t+1)}^T = \hat{B} = (X^TX)^{-1}X^TY = (P_{(t)}^TP_{(t)})^{-1}P_{(t)}^TR^T$$

   is the solution of the multi-output least-squares problem where n_u is the number of outputs, n_p is the number of samples and k is the number of parameters.

2. Use of the relationship $R = UP^T$. We set $Y = R$ ($[n_u, n_p]$), $X = U_{(t)}$ ($[n_u, k]$) and $B = P_{(t+1)}^T$ ($[k, n_p]$).

$$P_{(t+1)}^T = \hat{B} = (X^TX)^{-1}X^TY = (U_{(t)}^TU_{(t)}^T)^{-1}U_{(t)}^TR$$

   is the solution of the multi-output least-squares problem where n_p is the number of outputs, n_u is the number of samples and k is the number of parameters.

ALS (PSEUDO-CODE)

Initialize $P_{(0)}$ and $Q_{(0)}$;
repeat
  for $i = 1, \ldots, n_u$ **do**
    $U_i^T = (\sum_{\{j=1,..,n_p\}} P_j^T P_j)^{-1} \sum_{\{j=1,..,n_p, w_{i,j}=1\}} r_{ij} P_j^T$
  end
  for $j = 1, \ldots, n_p$ **do**
    $P_j^T = (\sum_{\{i=1,..,n_u\}} U_i^T U_i)^{-1} \sum_{\{i=1,..,n_u, w_{i,j}=1\}} r_{ij} U_i^T$
  end
until convergence;

- r_{ij} is the rating of item j made by user i
- U_i ($[1, k]$) is the ith row ($i = 1, \ldots, n_u$) of $U_{(t)}$
- P_j ($[1, k]$) is the jth row ($j = 1, \ldots, n_p$) of $P_{(t)}$
- $w_{i,j} = 1$ means that the ith user has rated the jth product .

COMPUTATIONAL COST

- The matrix to be inverted has size $[k, k]$ and the computational cost of the inversion is $O(k^3)$.
- Update each U_i costs $O(n_i k + k^3)$ where n_i is the number of items rated by user i.
- Update each P_j costs $O(n_j k + k^3)$ where n_j is the number of users rating item j.
- Predict all user-item ratings costs $O(n_u n_p k)$

ALS DISTRIBUTED COMPUTATION

There are three methods to distribute the ALS computation

- Join: this is inspired to the method already introduced to distribute in a MapReduce fashion the least-squares computation in case of low number of inputs. This is the case here since $k << n_p, k << n_u$. Note also that the multi-output least squares can be easily parallelized because of the independence of each estimation.
- Broadcast: it makes the assumption that U and P are small enough to be stored locally on each machine
- Block: it is a block version of the broadcast method where only a part of the U and P matrices are sent to each worker. These portions are determined by precomputing the subset of users addressed by each worker and the related items.

ALS DISTRIBUTED COMPUTATION BY JOIN

Let us consider the update

$$U_i^T = \Big(\sum_{\{j=1,..,n_p\}} P_j^T P_j \Big)^{-1} \sum_{\{j=1,..,n_p, w_{i,j}=1\}} r_{ij} P_j^T$$

1. Store R as RDD of triplets (i, j, r_{ij}). This is convenient given the sparse nature of the rating
2. Store U and P as RDDs
3. Join R and P by using the key j (items)
4. Map the joint RDD to compute $P_j^T P_j$ and return $(i, P_j^T P_j)$ $i = 1, \ldots, n_u$
5. Map the joint RDD to compute $r_{ij} P_j^T$ and return $(i, r_{ij} P_j^T)$ $i = 1, \ldots, n_u$
6. Reduce by user key to compute $\sum_j P_j^T P_j$
7. Reduce by user key to compute $\sum_j r_{ij} P_j^T$
8. Invert $\sum_j P_j^T P_j$ and update U_i

ALS DISTRIBUTED COMPUTATION BY BROADCAST

1. Partition R (rowwise) such that the same user is on the same machine. Let us call it R_1
2. Partition R^T such that the same item is on the same machine. Let us call it R_2
3. Broadcast U and P.
4. Given the local availability of P on each worker, map R_1 such to compute for each user the update

$$U_i^T = (\sum_j P_j^T P_j)^{-1} \sum_j r_{ij} P_j^T, \qquad i = 1, \ldots, n_u$$

5. Given the local availability of U on each worker, map R_2 such to compute for each product the update

$$P_j^T = (\sum_i U_i^T U_i)^{-1} \sum_i r_{ij} U_i^T, \qquad j = 1, \ldots, n_p$$

STOCHASTIC GRADIENT DESCENT

- By computing the gradient of the cost function we obtain the gradient-based iterative formulas

$$\begin{cases} U_i^{(t+1)} & = U_i^{(t)} - \gamma(e_{ij} P_j^{(t)} + \lambda U_i^{(t)}) \\ P_j^{(t+1)} & = P_j^{(t)} - \gamma(e_{ij} U_i^{(t)} + \lambda P_j^{(t)}) \end{cases}$$

  where $e_{ij}^{(t)} = r_{ij} - U_i^{(t)} (P_j^{(t)})^T$
- If we assume that the factor matrices U and P fit in memory, we can implement it in a streaming fashion where ratings data r_{ij} arrive sequentially
- Effective in nonstationary settings (product perception and popularity may change as time passes by).

LEARNED LESSONS FORM THE NETFLIX COMPETITION

- In 2006 Netflix offered a prize of 1,000,000 $ to the first person or team to beat their own recommendation algorithm, called CineMatch, by 10%.
- After over three years of work, the prize was awarded in September, 2009.
- Training dataset was a rating (1-5 stars) matrix of half a million users on 17,000 movies. This data was selected from a larger dataset, and proposed algorithms were tested on their ability to predict the ratings in a secret remainder of the larger dataset.
- The UV-decomposition algorithm was found by three students and gave a 7% improvement over CineMatch, when coupled with normalization and a few other tricks.
- The winning entry was actually a combination of several different algorithms that had been developed independently.
- Genre and IMDB information appeared to be useless
- Time of rating turned out to be useful: blockbusters are rated faster than others

CONTENT BASED FILTERING

- It relies on the availability of (manual or automatically extracted) description of items (e.g. topic, genre, actor, directors) and of a user profile (e.g. sex, age) that gives importance to such items
- Issues:
  - How acquire and improve user profiles?
  - How to determine the match between item and user profiles?
  - How to automatically extract the item description?
- (+): it does not require large user groups
- (+): items can be recommended independently
- (-): hard to describe qualitative features
- (-): need of manual (and time consuming) annotation

EXAMPLE OF ITEM PROFILE

If the item is a movie the profile could be composed of

- Set of actors
- Director
- Year
- Genre (e.g. from the IMDB database)

EXTRACTING CONTENT FROM UNSTRUCTURED DATA

- Documents: it is possible to extract features related to word statistics (e.g. TF.IDF scores)
- Images: availability of user tags

NEAREST NEIGHBOUR APPROACH

Suppose we want to rate a new item

- Like in CF also in content-based approach we may adopt a nearest neighbor approach
- In CF the neighborhood is determined by the rating history of many users. Two items (e.g. movies) are similar if many users rated them in similar manner
- Here the neighborhood depends on a number of descriptive features (e.g. same topic, author)
- Other general purpose classification systems may be used as well, e.g. create a classification model for each user (or class of users) returning the degree of interest for a given item profile.
- Use of LSH functions for huge datasets to define buckets of similar items/users

KNOWLEDGE BASED RECOMMENDATION

- Content based and collaborative filtering make the assumption that we buy the product often. Now, this is not always the case (house, car,...). in this case the number of ratings is too small.
- It addresses one-time buyers (common in consumer electronics)
- No purchase history
- Preferences and tastes (e.g. car, computer) evolve with time.
- Interaction with user: which are the desired characteristics/price?
- From filtering approach to conversational approach. RS is not a filter but more a personalized guide.
- Issues:
  - How to acquire user profile knowledge in domains with no purchase history?
  - What is the most adequate interaction pattern? Sometime users want to define their own requirements (price lower than...).
- (+) no need of user history

KNOWLEDGE BASED RECOMMENDATION

In KB the user must specify the requirements, and the system tries to identify a solution and provide explanations. If no solution can be found, the user must change the requirements.
Two types of KB approaches

- Constraint-based: rely on an explicitly defined set of recommendation rules and attack the recommendation as a constraint satisfaction problem. Implemented as a filtering operation on relational table
- Case-based: focus on the retrieval of similar items (with a predefined threshold) on the basis of different types of similarity measures. Notion of distance product/requirement. Given a set R describing all the customer requirements it is possible to define for each product p a similarity

$$S(R, p) = \frac{\sum_{r \in R} w_r s(r, p)}{\sum_{r \in R} w_r}$$

where w_r is the importance weight of the requirement r and $s(r, p)$ is the similarity between the product p and that specific requirement r.

MORE VS. LESS IS BETTER

Given a requirement r expressed as a range and the related value of the product $\phi_r(p)$ the similarity depends on the nature of the re

- For MIB (More is Better) properties

$$s(r,p) = \frac{\phi_r(p) - \min(r)}{\max(r) - \min(r)}$$

- For LIB (Lower is Better) properties

$$s(r,p) = \frac{\max(r) - \phi_r(p)}{\max(r) - \min(r)}$$

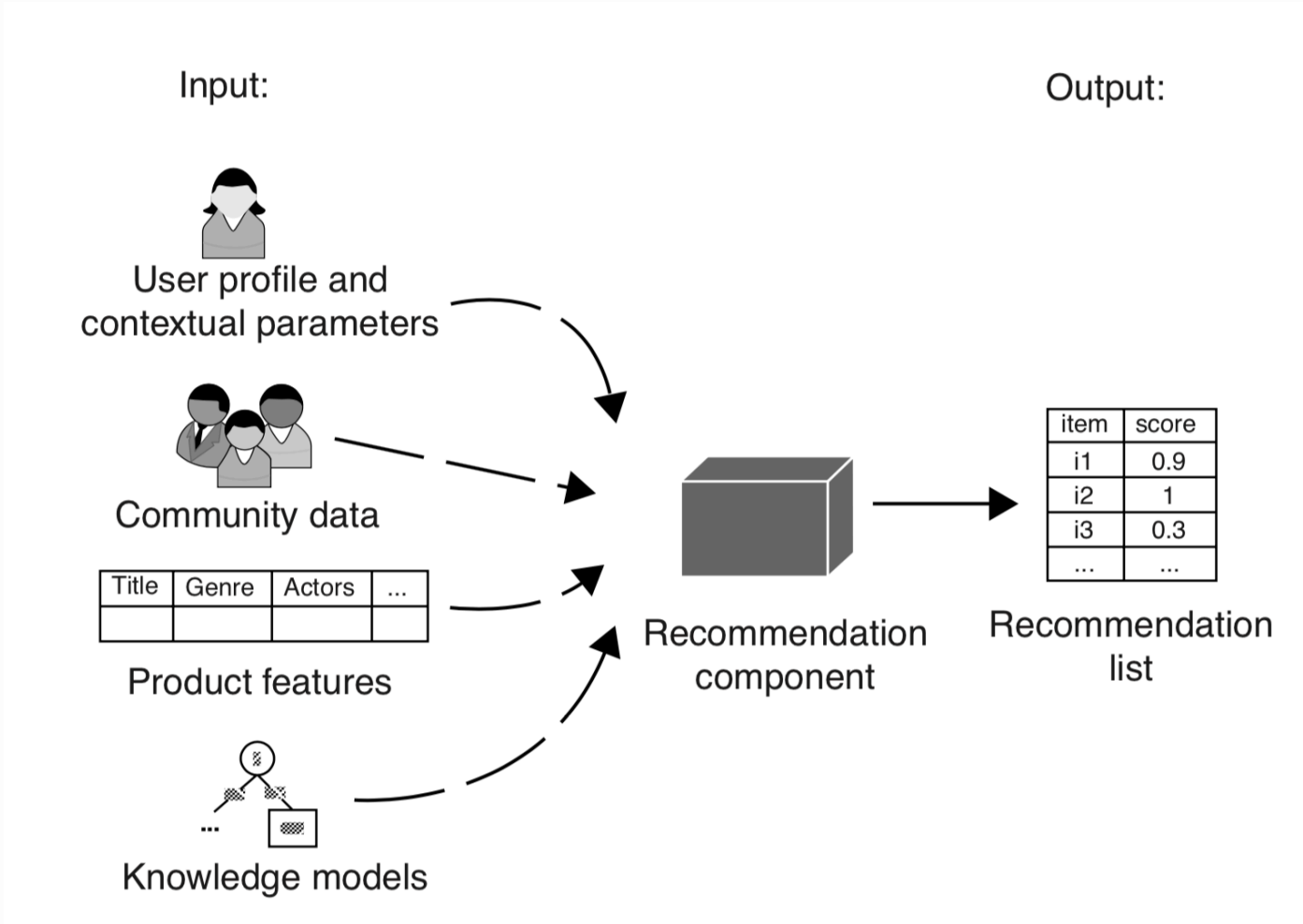
- For CIB (Closer is better)

$$s(r,p) = 1 - \frac{|\max(r) - \phi_r(p)|}{\max(r) - \min(r)}$$

INTERACTION ISSUES

- Need of graphical (e.g. web based) interaction
- Interaction pattern
  1. Users first specify their preferences (in a form or by a preference elicitation process)
  2. User is presented with a set of matching items
  3. User revise the requirements
- Issues: no adequate item, alternative proposals, use of defaults, adaptation to user search (learning on interaction logs), selection of next questions, space exploration, use of information of related customers (e.g. most popular price)

HYBRID APPROACHES



HYBRID APPROACHES

- Rationale: all previous methods have pros and cons. Why not to combine them?
- Strategies for hybridization design:
  - Monolithic: a single algorithmic implementation that incorporates all the available features (feature augmentation)
  - Parallelized: alternative recommender systems operate independently and produce separate recommendation lists which are combined into a final set
  - Pipelined: the output of a recommender is used as input of a subsequent one.

INTERESTING ISSUES

- Adversarial attacks
- Privacy aspects

These aspects are common to many similar big data applications [1] .

ATTACKING RECOMMENDATION SYSTEMS

Interesting research question

- Attacks on collaborative systems: recommendation systems has an important business impact. What if users are not honest and want to influence the buying behavior (e.g. damage a competitor, sabotage the system)?
- Fake user accounts may be created to be as close as possible to other profiles in order to influence their recommendation
- Push (increase the recommendation of an item) vs nuke (decrease the recommendation) attacks
- Random attack, average attack, bandwagon (i.e. high ratings for blockbusters products so to have many neighbors), segment attack (targeting a specific segment of the population, e.g. fans of science-fiction), love/hate attack (rate some item at the minimum value while rating the other items at the highest possible values)
- automated crawler that simulates web browsing sessions with the goal of associating a target item (the page to be pushed) with other pages in such a way that the target items appear on recommendation lists more

ATTACKING RS: COUNTERMEASURES

- Using model-based (or hybrid) techniques instead of memory-based techniques to reduce vulnerability to alteration of the training set
- Use of RS that do not rely only on rating profiles.
- Use of trust measures to attribute weights to different profiles
- Increase the cost of injecting new profiles (use of Captcha, limitations based on IP addresses)
- Automated attack detection (detecting shilling behavior, e.g. cluster of users rating in short time in suspiciously similar manner)
- Detection fake profiles in unsupervised (based on agreement with other users, degree of similarity with top neighbors, or rating deviation from average) or supervised (labeling of genuine profiles) manner. Risk of false positives.
- Monitoring rating time evolution.

PRIVACY ASPECTS

- Rating databases of collaborative filtering recommender systems contain detailed information about the individual tastes and preferences of their users.
- Sensitive and valuable (also in monetary terms) information
- Need to avoid leaks to preserve customers confidence and trust: CF cannot exist without users willing to rate.
- Naive architecture (single central server storing plain (unobfuscated and nonencrypted) ratings) provides a central target point of attack
- Few report of attacks on real-world systems are made public to avoid economic backlash.

PRIVACY COUNTERMEASURES

Two main strategies

- Data distribution: avoid storing the information in a central place (e.g. use of P2P architectures in which every node maintains its information and communicates only on demand)
- Data perturbation: data obfuscation by applying random data perturbation (e.g. add random noise or transform quantitative rating into concordances) in such a manner that the outcome of the analytics (e.g. compute average statistics or scalar products) is robust wrt perturbation. Trade off obfuscation/accuracy. Problems with perturbation size not adequate for all the ratings

Combination of the two strategies are recommended too.


D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich.
Recommender Systems.
Cambridge, 2012.


Anand Rajaraman, Jure Leskovec, and Jeffrey D. Ullman.
Mining Massive Datasets.
2014.