Collaborative filtering

Nisheeth

Collaborative Filtering (CF)

- The most prominent approach to generate recommendations
  - used by large, commercial e-commerce sites
  - well-understood, various algorithms and variations exist
  - applicable in many domains (book, movies, DVDs, ..)
- Approach
  - use the "wisdom of the crowd" to recommend items
- Basic assumption and idea
  - Users give ratings to catalog items (implicitly or explicitly)
  - Customers who had similar tastes in the past, will have similar tastes in the future

User-based nearest-neighbor collaborative filtering

- The basic technique:
  - Given an "active user" (Alice) and an item I not yet seen by Alice
  - The *goal is to estimate Alice's rating for this item*, e.g., by
    - find a set of users (peers) who liked the same items as Alice in the past **and** who have rated item I
    - use, e.g. the average of their ratings to predict, if Alice will like item I
    - do this for all items Alice has not seen and recommend the best-rated

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User 1	3	1	2	3	3
User 2	4	3	4	3	5
User 3	3	3	1	5	4
User 4	1	5	5	2	1

User-based nearest-neighbor collaborative filtering

- Some first questions
  - How do we measure similarity?
  - How many neighbors should we consider?
  - How do we generate a prediction from the neighbors' ratings?

Measuring user similarity

- A popular similarity measure in user-based CF: **Pearson correlation**

$$sim(a,b) = \frac{\sum_{p \in P}(r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P}(r_{a,p} - \bar{r}_a)^2} \sqrt{\sum_{p \in P}(r_{b,p} - \bar{r}_b)^2}}$$

a, b : users

$\overline{r_a}, \overline{r_b}$

$r_{a,p}$: rating of user a for item p

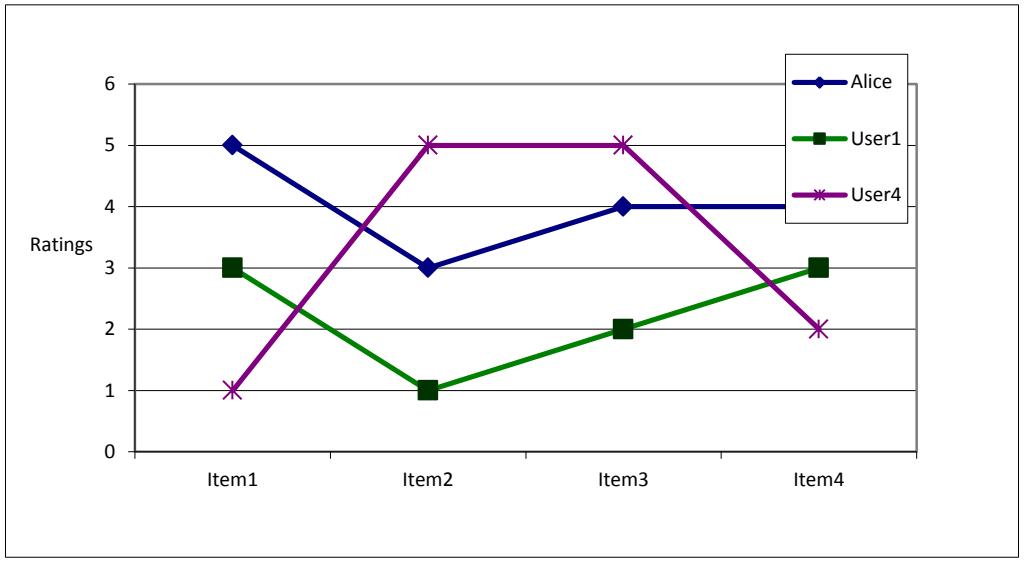
P : set of items, rated both by a and b

Possible similarity values between -1 and 1;

= user's average ratings

Pearson correlation

- Takes differences in rating behavior into account



- Works well in usual domains, compared with alternative measures
  - such as cosine similarity

Making predictions

- A common prediction function:

$$pred(a,p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a,b) * (r_{b,p} - \overline{r_b})}{\sum_{b \in N} sim(a,b)}$$

- Calculate, whether the neighbors' ratings for the unseen item *i* are higher or lower than their average
- Combine the rating differences – use the similarity as a weight
- Add/subtract the neighbors' bias from the active user's average and use this as a prediction

Making recommendations

- Making predictions is typically not the ultimate goal
- Usual approach (in academia)
  - Rank items based on their predicted ratings
- However
  - This might lead to the inclusion of (only) niche items
  - **In practice also:** Take item popularity into account
- Approaches
  - "Learning to rank"
    - Optimize according to a given rank evaluation metric (see later)

Improving the metrics / prediction function

- Not all neighbor ratings might be equally "valuable"
  - Agreement on commonly liked items is not so informative as agreement on controversial items
  - **Possible solution**: Give more weight to items that have a higher variance
- Value of number of co-rated items
  - Use "significance weighting", by e.g., linearly reducing the weight when the number of co-rated items is low
- Case amplification
  - Intuition: Give more weight to "very similar" neighbors, i.e., where the similarity value is close to 1.
- Neighborhood selection
  - Use similarity threshold or fixed number of neighbors

Item-based CF

- Scalability issues arise with U2U if many more users than items
  (m >> n , m = |users|, n = |items|)
  - e.g. Amazon.com
  - Space complexity $O(m^2)$ when pre-computed
  - Time complexity for computing Pearson $O(m^2n)$

- High sparsity leads to few common ratings between two users

- Basic idea: "Item-based CF exploits relationships between items first, instead of relationships between users"

Item-based collaborative filtering

- Basic idea:
  - Use the similarity between items (and not users) to make predictions
  - Treat ratings as item features (big assumption)
- Example:
  - Look for items that are similar to Item5
  - Take Alice's ratings for these items to predict the rating for Item5

	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

The cosine similarity measure

- Produces better results in item-to-item filtering
  - for some datasets, no consistent picture in literature
- Ratings are seen as vector in n-dimensional space
- Similarity is calculated based on the angle between the vectors

$$sim(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| * |\vec{b}|}$$

- Adjusted cosine similarity
  - take average user ratings into account, transform the original ratings
  - U: set of users who have rated both items a and b

$$sim(a, b) = \frac{\sum_{u \in U}(r_{u,a} - \overline{r_u})(r_{u,b} - \overline{r_u})}{\sqrt{\sum_{u \in U}(r_{u,a} - \overline{r_u})^2}\sqrt{\sum_{u \in U}(r_{u,b} - \overline{r_u})^2}}$$

Pre-processing for item-based filtering

- Item-based filtering does not solve the scalability problem itself
- Pre-processing approach by Amazon.com (in 2003)
  - Calculate all pair-wise item similarities in advance
  - The neighborhood to be used at run-time is typically rather small, because only items are taken into account which the user has rated
  - Item similarities are supposed to be more stable than user similarities
- Memory requirements
  - Up to N^2 pair-wise similarities to be memorized (N = number of items) in theory
  - In practice, this is significantly lower (items with no co-ratings)
  - Further reductions possible
    - Minimum threshold for co-ratings (items, which are rated at least by *n* users)
    - Limit the size of the neighborhood (might affect recommendation accuracy)

More about ratings

- Pure CF-based systems only rely on the rating matrix
- Explicit ratings
  - Most commonly used (1 to 5, 1 to 7 Likert response scales)
  - Research topics
    - "Optimal" granularity of scale; indication that 10-point scale is better accepted in movie domain
    - Multidimensional ratings (multiple ratings per movie)
  - Challenge
    - Users not always willing to rate many items; sparse rating matrices
    - How to stimulate users to rate more items?
- Implicit ratings
  - clicks, page views, time spent on some page, demo downloads …
  - Can be used in addition to explicit ones; question of correctness of interpretation

Data sparsity problems

- Cold start problem
  - How to recommend new items? What to recommend to new users?
- Straightforward approaches
  - Ask/force users to rate a set of items
  - Use another method (e.g., content-based, demographic or simply non-personalized) in the initial phase
- Alternatives
  - Use better algorithms (beyond nearest-neighbor approaches)
  - Example:
    - In nearest-neighbor approaches, the set of sufficiently similar neighbors might be to small to make good predictions
    - Assume "transitivity" of neighborhoods

Example algorithms for sparse datasets

- Recursive CF
  - Assume there is a very close neighbor ***n*** of u who however has not rated the target item ***i*** yet.
  - Idea:
    - Apply CF-method recursively and predict a rating for item ***i*** for the neighbor
    - Use this predicted rating instead of the rating of a more distant direct neighbor

Contrast – u2u vs i2i

	Avg. neighbors	Avg. ratings
User-based	$(\lvert\mathcal{U}\rvert - 1)\left(1 - \left(\frac{\lvert\mathcal{I}\rvert - p}{\lvert\mathcal{I}\rvert}\right)^{p}\right)$	$\frac{p^2}{\lvert\mathcal{I}\rvert}$
Item-based	$(\lvert\mathcal{I}\rvert - 1)\left(1 - \left(\frac{\lvert\mathcal{U}\rvert - q}{\lvert\mathcal{U}\rvert}\right)^{q}\right)$	$\frac{q^2}{\lvert\mathcal{U}\rvert}$

p = avg number of ratings per user
q = avg number of ratings per item

Contrast – u2u vs i2i

	U2U	I2I
Accuracy	|U|<<|I|	|U|>>|I|
Efficiency	|U|<<|I|	|U|>>|I|
Stability	Static users	Static items
Justifiability		Always better
Serendipity	Always better	

Shared problems: coverage, sparsity

Memory-based vs model-based approaches

- Both u2u and i2i CF are "memory-based"
  - the rating matrix is directly used to find neighbors / make predictions
  - does not scale for most real-world scenarios
  - large e-commerce sites have tens of millions of customers and millions of items
- Model-based approaches
  - based on an offline pre-processing or "model-learning" phase
  - at run-time, only the learned model is used to make predictions
  - models are updated / re-trained periodically
  - large variety of techniques used
  - model-building and updating can be computationally expensive

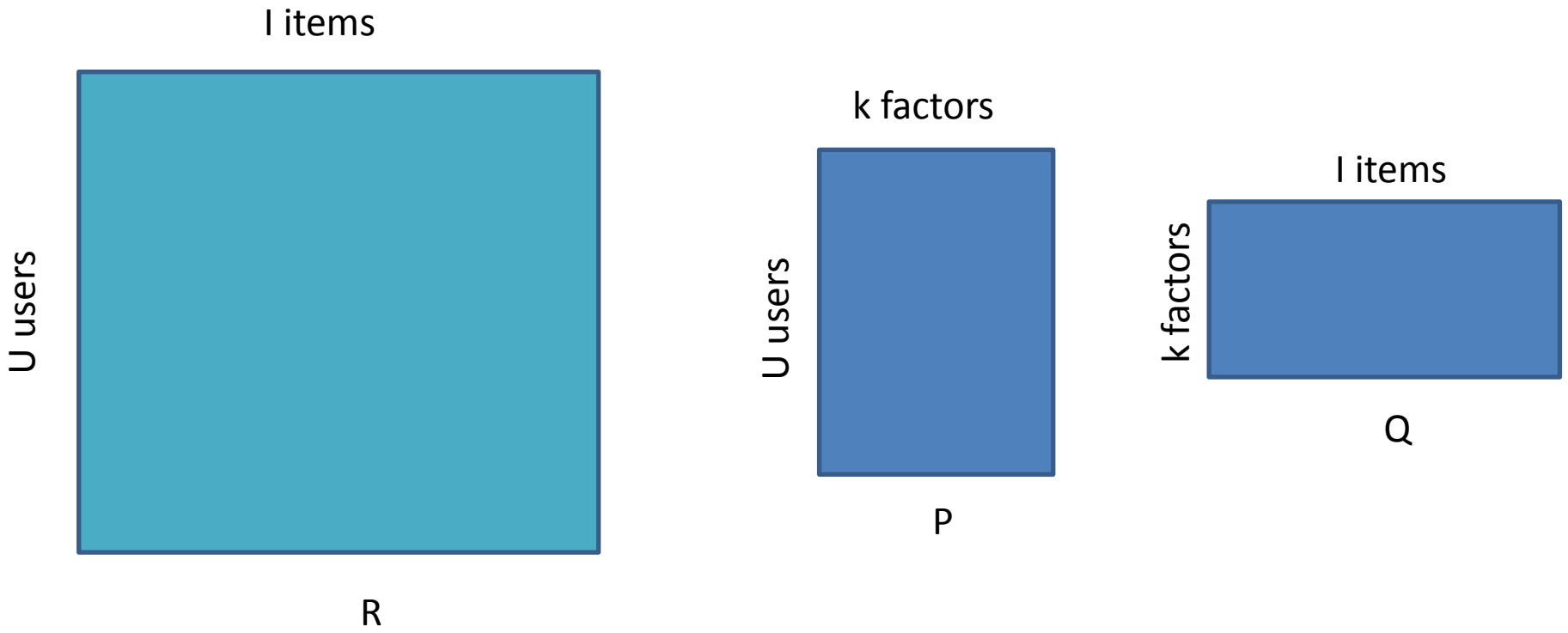
Model-based approaches

- Plethora of different techniques proposed in the last years, e.g.,
  - Matrix factorization techniques, statistics
    - singular value decomposition, principal component analysis
  - Association rule mining
    - compare: shopping basket analysis
  - Probabilistic models
    - clustering models, Bayesian networks, probabilistic Latent Semantic Analysis
  - Various other machine learning approaches
- Costs of pre-processing
  - Usually not discussed
  - Incremental updates possible?

CFs using matrix factorization

- Basic idea: Trade more complex offline model building for faster online prediction generation
- Singular Value Decomposition for dimensionality reduction of rating matrices
  - Captures important factors/aspects and their weights in the data
  - factors can be genre, actors but also non-understandable ones
  - Assumption that k dimensions capture the signals and filter out noise (K = 20 to 100)
- Constant time to make recommendations
- Approach also popular in IR (Latent Semantic Indexing), data compression, …

SVD: basic intuition



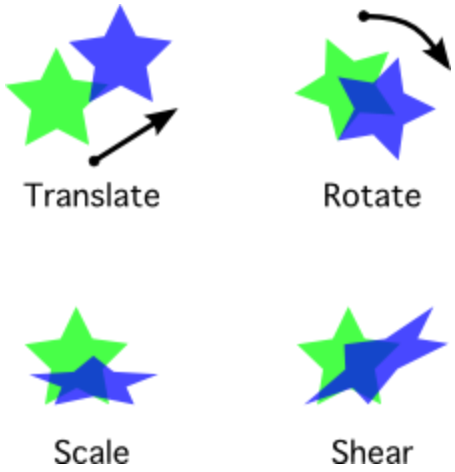
Minimize err(k) = $|R - PQ|^2_F$

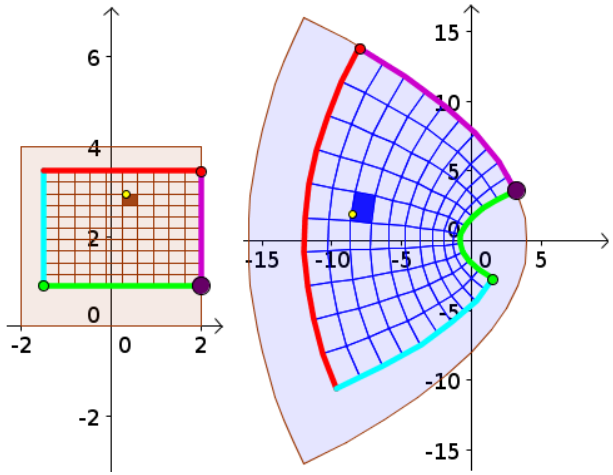
$$= \sum_{u,i} (r_{ui} - p_u q_i)^2$$

Equivalent to SVD

Linear transformations

- F(u + v) = F(u) +F(v)
- F(cu) = cF(u)





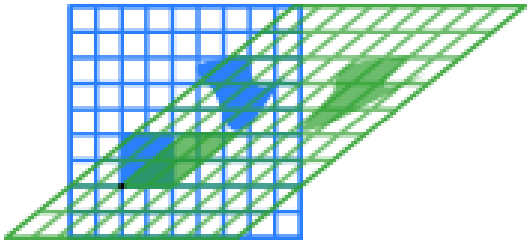
Matrices represent linear transforms

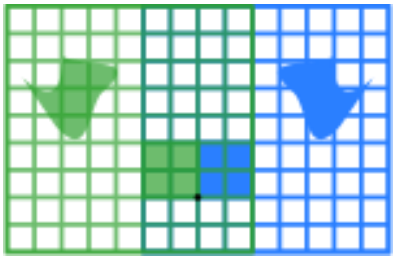
- A linear map Rn → Rm is equivalent to an n-by-m matrix
- What does this matrix do?

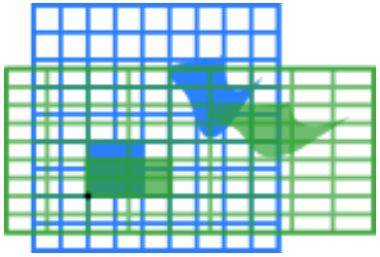
$$\mathbf{A} = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

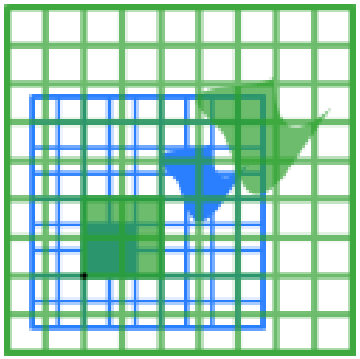
- It transforms the unit square
  - Transformation clarifies the role of matrices as linear maps

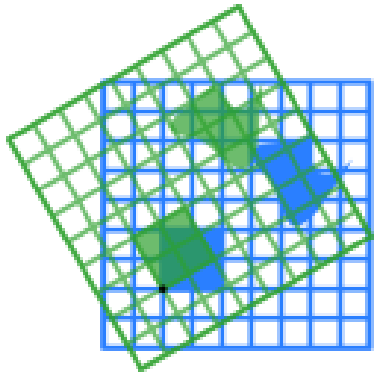
Matrices as instruction lists











Eigenvalues and eigenvectors

- Eigenvalue definition $Ax = \lambda x$
- Calculation: solutions of $|A - \lambda I| = 0$
- Apply solutions to original equation to calculate eigenvectors $(A- \lambda I)v = 0$
- Set of eigenvectors, along with the null set, forms the *eigenbasis* of a matrix
- What does it mean?
- Special simplified instruction set that is equivalent to the original instruction set
  - but implemented using only scaling

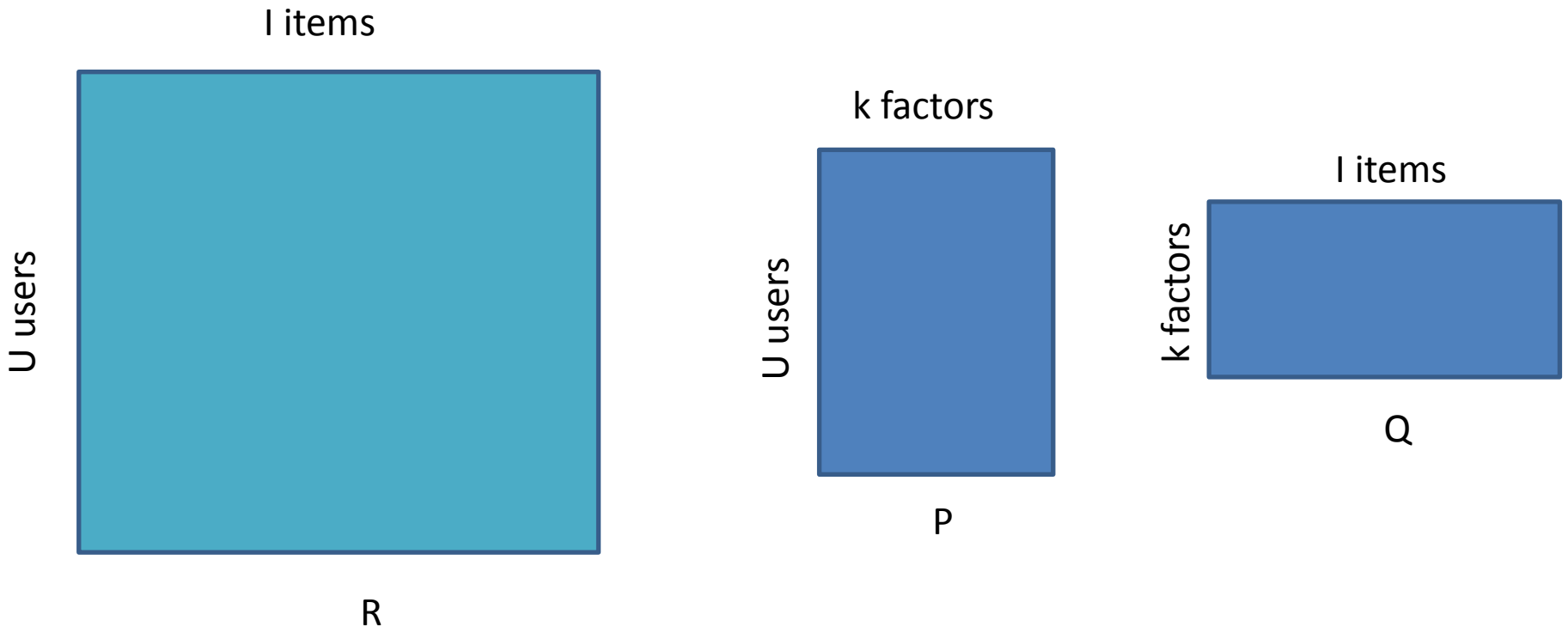
Eigendecomposition

- Writing out the transformation operation using its basic components
- $A = ELE^{-1}$
  - E is a matrix with each column an eigenvector
  - L is a diagonal matrix, with each non-zero element an eigenvalue
- Can arrange this in decreasing order of eigenvalue magnitude
- What happens when we set the smaller eigenvalues to zero?

Latent factor models via matrices

- Factorize n x n matrix A as a product of two matrices B,C such that
  - $B = EL^{1/2}$
  - $C^T = E^{-1}L^{1/2}$
- Can approximate A using B_k and C_k that consider only the top k eigenvalues
  - $err(k) = |A - BC^T|^2_F$
- Can generalize to non-square matrices
  - Singular value decomposition
  - Decompose A into $U\Sigma V^T$
  - U,V are matrices of singular vectors, Σ is a diagonal matrix that contains singular values
    - Singular vectors of A are eigenvectors of A^TA and AA^T
    - Non-zero singular values of A are square roots of non-zero eigenvalues of AA^T

SVD: basic intuition



Minimize err(k) = $|R - PQ|^2_F$

$$= \sum_{u,i} (r_{ui} - p_u q_i)^2$$

Equivalent to SVD

Typical SVD application

- SVD: $M_k = U_k \times \Sigma_k \times V_k^T$

U_k	Dim1	Dim2
Alice	0.47	-0.30
Bob	-0.44	0.23
Mary	0.70	-0.06
Sue	0.31	0.93

V_k^T					
Dim1	-0.44	-0.57	0.06	0.38	0.57
Dim2	0.58	-0.66	0.26	0.18	-0.36

Σ_k	Dim1	Dim2
Dim1	5.63	0
Dim2	0	3.23

- Prediction: $\hat{r}_{ui} = \bar{r}_u + U_k(Alice) \times \Sigma_k \times V_k^T(EPL)$

  = 3 + 0.84 = 3.84

SVD concerns

- User-rating matrix is often sparse
- How to interpret missing values?
  - Interpreting as 0 induces bias
  - Filling with default values also induces bias
- One solution- regularized learning
- $err(k) = \sum_{u,i \in R} (r_{ui} - p_u q_i)^2 + \gamma \left(\|\boldsymbol{p}_u\|^2 + \|\boldsymbol{q}_i\|^2 \right)$

Collaborative Filtering Issues

- Pros: 👍
  - well-understood, works well in some domains, no knowledge engineering required
- Cons: 👎
  - requires user community, sparsity problems, no integration of other knowledge sources, no explanation of results
- What is the best CF method?
  - In which situation and which domain? Inconsistent findings; always the same domains and data sets; differences between methods are often very small (1/100)
- How to evaluate the prediction quality?
  - MAE / RMSE: What does an MAE of 0.7 actually mean?
  - Serendipity: Not yet fully understood
- What about multi-dimensional ratings?

Other methods: association rule mining

- Commonly used for shopping behavior analysis
  - aims at detection of rules such as

    "If a customer purchases baby-food then he also buys diapers
    in 70% of the cases"
- Association rule mining algorithms
  - can detect rules of the form X => Y (e.g., baby-food => diapers) from a set of sales transactions D = {t_1, t_2, … t_n}
  - measure of quality: support, confidence

Other methods: Probabilistic methods

- Treat users as mixtures of topics
- Treat topics as distribution over item sample frequencies
- Run LDA?
- What could go wrong?