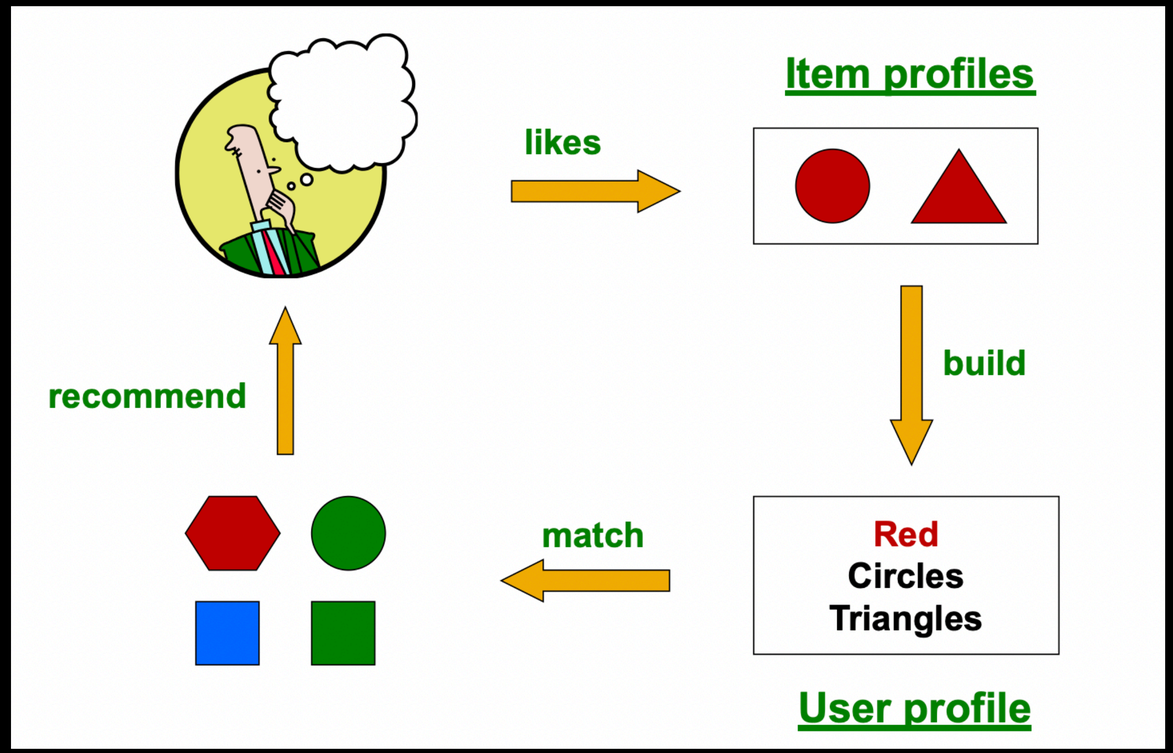
Recommender Systems

An abridging of Jure Leskovec’s research

Pablo Rodriguez Bertorello

Sketch of Plan

- How much will user x like item y?
- Recommend top k items user x may like
- Update user from further action of all users

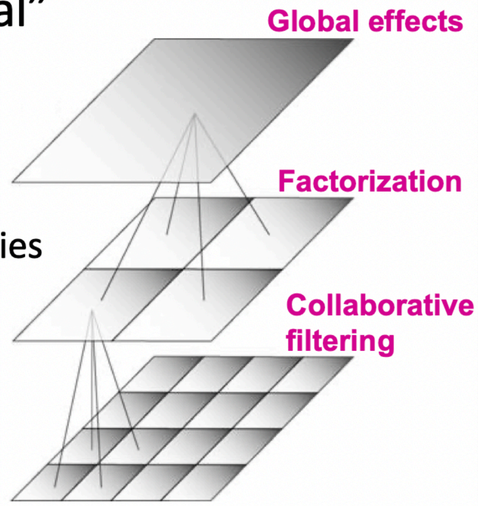


Approaches

BellKor Recommendation Engine

- Compare to content-based: eg TF-IDF
- Collaborative: user-based, or item-baed
- Latent factor

- **The winner of the Netflix Challenge**
- **Multi-scale modeling of the data:**
  Combine top level, “regional” modeling of the data, with a refined, local view:
  - **Global:**
    - Overall deviations of users/movies
  - **Factorization:**
    - Addressing “regional” effects
  - **Collaborative filtering:**
    - Extract local patterns



Collaborative Filtering

User Collaborative Filtering

- Consider user x
- Find set N of other users whose ratings are “similar” to x’s ratings
- Estimate x’s ratings based on ratings of users in N
- Alternatively: Item Collaborative Filtering

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.2	
Bob		0.5		0.3
Carol	0.2		1	
David				0.4

Which Similarity?

- Let rx be the vector of user x’s ratings
  - Jaccard similarity measure
  - Cosine similarity measure
  - Pearson correlation coefficient

Prediction

Via Cosine Similarity

- Let rx be the vector of user x's ratings
- Let N be the set of k users most similar to x who have rated item i
- Prediction for item i of user x:

- $r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi}$
- Or even better: $r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$

Similarity to Nearest Neighbors

- Define **similarity** s_{ij} of items ***i*** and ***j***
- Select ***k*** nearest neighbors $N(i; x)$
  - Items most similar to ***i***, that were rated by ***x***
- Estimate rating r_{xi} as the weighted average:

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

baseline estimate for r_{xi}

$$b_{xi} = \mu + b_x + b_i$$

- μ = overall mean movie rating
- b_x = rating deviation of user ***x***
  = (*avg. rating of user **x***) – μ
- b_i = rating deviation of movie ***i***

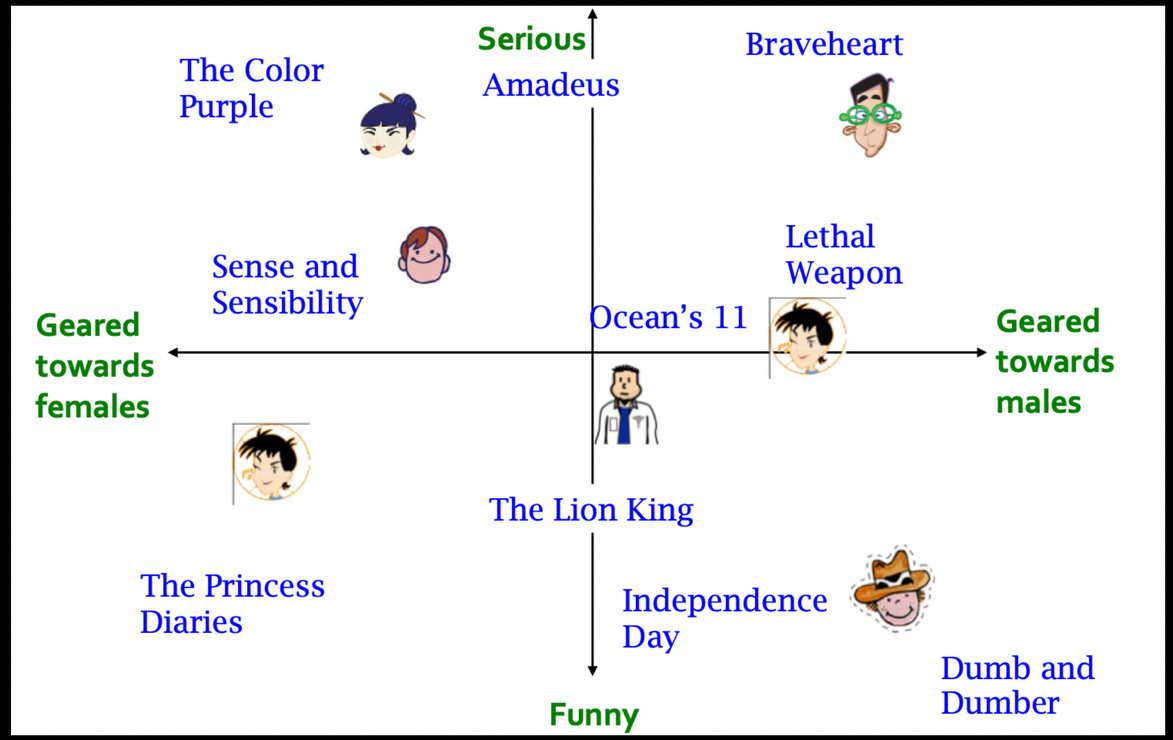
Pro/Cons of Collaborative Filtering

- Works for any kind of item
- But relies only on local knonwledge
- Cold start, need enough users in the system to find a match
- Sparsity, hard to find users that have rated the same items
- Popularity bias, cannot recommend items to someone with unique taste

Latent Factor

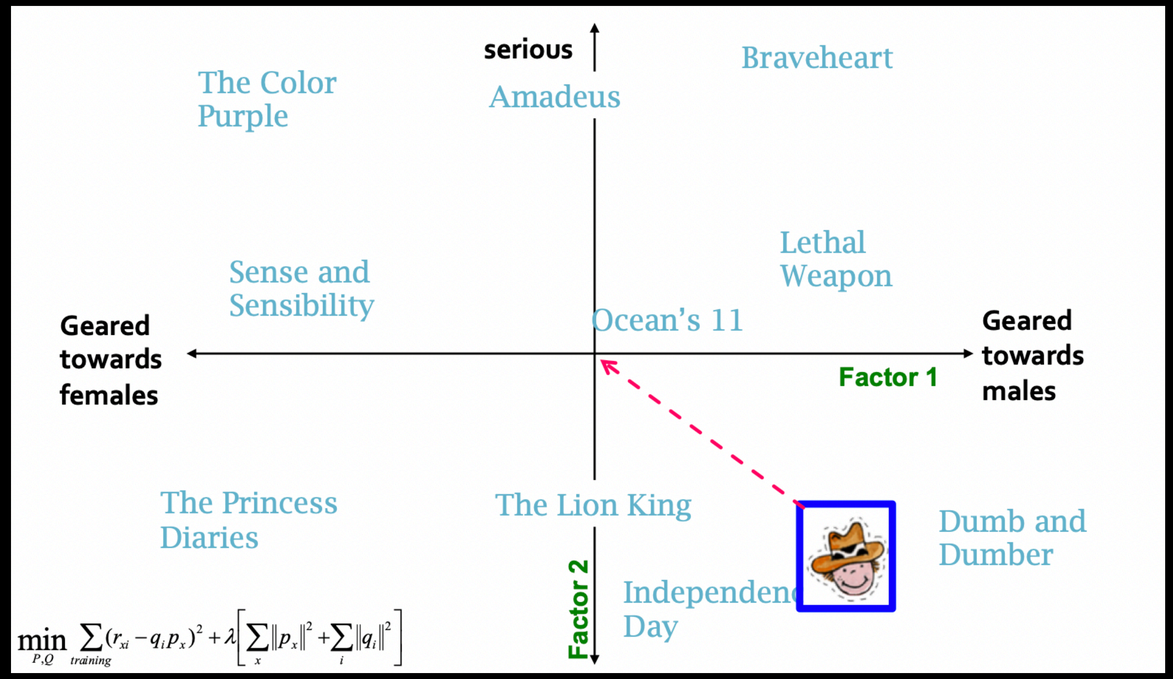
The Plan (I)

Fature Space



Enhanced Plan (I)

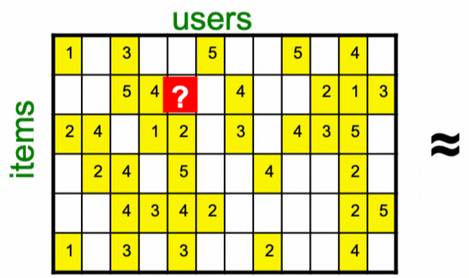
Principal concept-space



Predict

How much will user like item?

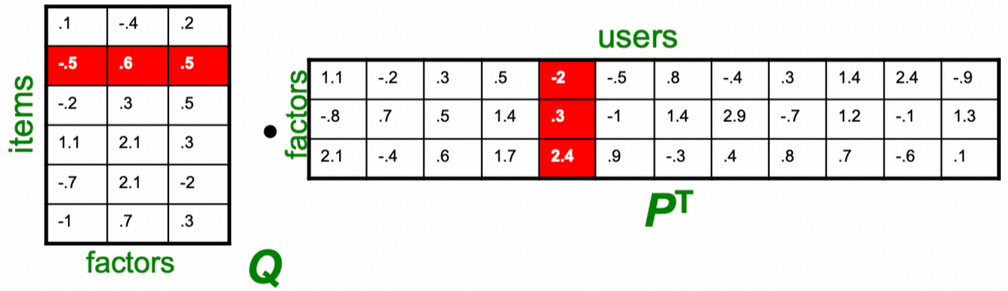
- How to estimate the missing rating of user *x* for item *i*?



≈

$$\hat{r}_{xi} = q_i \cdot p_x = \sum_f q_{if} \cdot p_{xf}$$

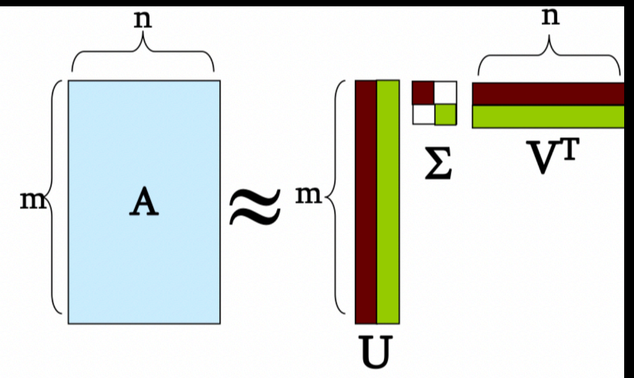
q_i = row i of Q
p_x = column x of P^T



SVD

Factor the User-concept,
and Item-concept eigen-vectors

- **Remember SVD:**
  - **A**: Input data matrix
  - **U**: Left singular vecs
  - **V**: Right singular vecs
  - Σ: Singular values



- **So in our case:**
  "SVD" on Netflix data: $R \approx Q \cdot P^T$
  $A = R,\ Q = U,\ P^T = \Sigma V^T$

$$\hat{r}_{xi} = q_i \cdot p_x$$

Latent Factor Model

- Machine-learn the factorization
- Minimize rating matrix reconstruction error
- Loss excludes un-rated cells (empty/null). No assumptions:
  - zero
  - mean

- **We already know that SVD gives minimum reconstruction error (Sum of Squared Errors):**

$$\min_{U,V,\Sigma} \sum_{ij \in A} \left(A_{ij} - [U\Sigma V^{T}]_{ij}\right)^2$$

- **Note two things:**
  - **SSE** and **RMSE** are monotonically related:
    - $RMSE = \frac{1}{c}\sqrt{SSE}$ **Great news: SVD is minimizing RMSE!**
  - **Complication:** The sum in SVD error term is over all entries (no-rating is interpreted as zero-rating). But our ***R*** has missing entries!

Learning the SVD factorization

Gradient Descent

- **Want to find matrices *P* and *Q*:**

$$\min_{P,Q} \sum_{training} (r_{xi} - q_i p_x)^2 + \left[\lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2 \right]$$

- **Gradient descent:**
  - Initialize ***P*** and ***Q*** (using SVD, pretend missing ratings are 0)
  - Do gradient descent:
    - $P \leftarrow P - \eta \cdot \nabla P$
    - $Q \leftarrow Q - \eta \cdot \nabla Q$
    - where ∇Q is gradient/derivative of matrix ***Q***:
      $\nabla Q = [\nabla q_{if}]$ and $\nabla q_{if} = \sum_{x:(x,i) \in training} -2(r_{xi} - q_i p_x) p_{xf} + 2\lambda_2 q_{if}$
      - Here q_{if} is entry ***f*** of row q_i of matrix ***Q***

How to compute gradient of a matrix?
Compute gradient of every element independently!

Learning the SVD Factorization

Stochastic Gradient Descent

- **Gradient Descent (GD) vs. Stochastic GD**
  - **Observation:** $\nabla Q = [\nabla q_{if}]$ where
    $$\nabla q_{if} = \sum_{x:(x,i)\in training} -2(r_{xi} - q_{if}p_{xf})p_{xf} + 2\lambda q_{if} = \sum_{x:(x,i)\in training} \nabla \boldsymbol{Q}(\boldsymbol{r_{xi}})$$
    - Here $\boldsymbol{q_{if}}$ is entry $\boldsymbol{f}$ of row $\boldsymbol{q_i}$ of matrix $\boldsymbol{Q}$
  - $\boldsymbol{Q} \leftarrow \boldsymbol{Q} - \eta\nabla\boldsymbol{Q} = \boldsymbol{Q} - \eta[\sum_{\boldsymbol{x}} \nabla \boldsymbol{Q}(\boldsymbol{r_{xi}})]$
  - **Idea:** Instead of evaluating gradient over all ratings evaluate it for each individual rating and make a step
- **GD:** $\boldsymbol{Q} \leftarrow \boldsymbol{Q} - \eta[\sum_{\boldsymbol{r_{xi}}} \nabla \boldsymbol{Q}(\boldsymbol{r_{xi}})]$
- **SGD:** $\boldsymbol{Q} \leftarrow \boldsymbol{Q} - \mu\nabla \boldsymbol{Q}(\boldsymbol{r_{xi}})$
  - **Faster convergence!**
    - Need more steps but each step is computed much faster

Backup

Term Frequency - Inverse Document Frequency

f_{ij} = frequency of term (feature) i in doc (item) j

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

Note: we normalize TF to discount for "longer" documents

n_i = number of docs that mention term i
N = total number of docs

$$IDF_i = \log \frac{N}{n_i}$$

TF-IDF score: $w_{ij} = TF_{ij} \times IDF_i$

Doc profile = set of words with highest **TF-IDF** scores, together with their scores