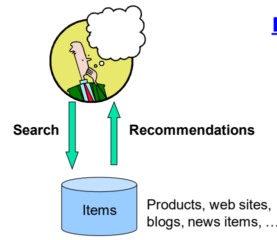
Data Mining
Recommendation Models

Prof. Dr. Şule Gündüz Öğüdücü

Recommendations



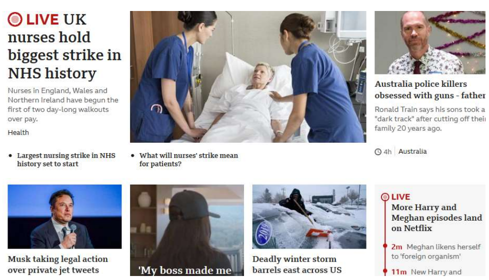
Examples:

amazon.com, Pandora, StumbleUpon, del.icio.us, Netflix, movielens (helping you find the right movies), last.fm, Google News, YouTube, Xbox Live

http://www.mmds.org

Recommender Systems

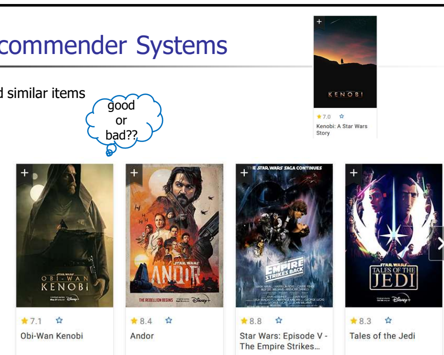
- Recommend new content



Recommender Systems

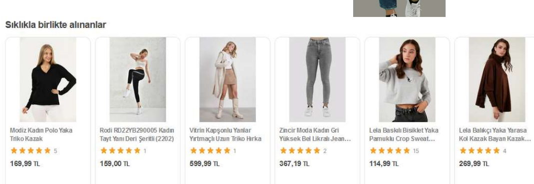
- Recommend similar items

good or bad??



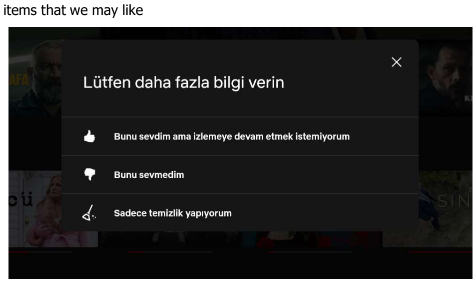
Recommender Systems

- Recommend co-occurred items



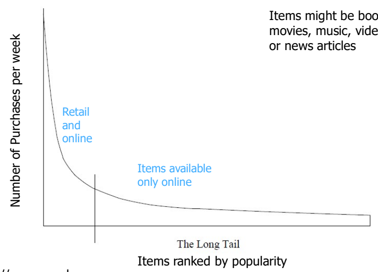
Recommender Systems

- identify items that we may like



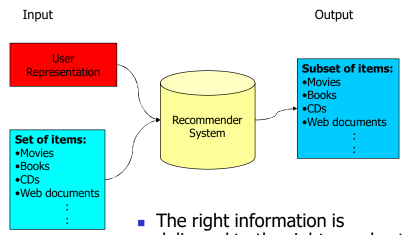
The Long Tail

Items might be books, movies, music, videos or news articles



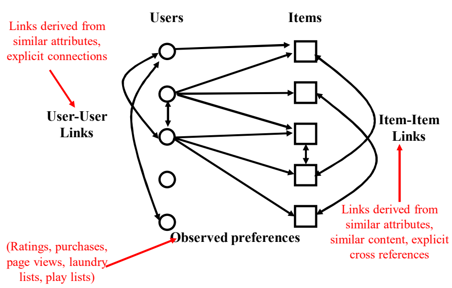
http://www.mmds.org

Recommendation Process

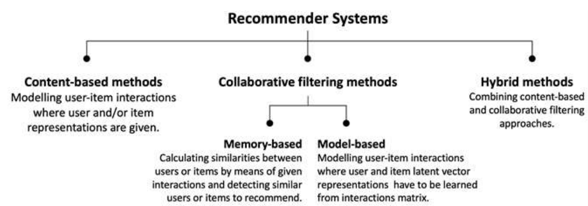


- The right information is delivered to the right people at the right time.

The Recommendation System Space



Recommender Systems: Methods



How it Works?

- Each user has a profile
- Users rate items
  - Explicitly: score from 1..5
  - Implicitly: web usage mining
    - Time spent in viewing the item
    - Navigation path
    - Etc…
- System does the rest, How?
  - This is what we will show today

Formal Model

- ***X*** = set of **Customers**
- ***S*** = set of **Items**
- **Utility function** ***u***: ***X*** × ***S*** → ***R***
  - ***R*** = set of ratings
  - ***R*** is a totally ordered set
  - e.g., **0-5** stars, real number in **[0,1]**

http://www.mmds.org

Utility Matrix

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

http://www.mmds.org

Key Problems

(1) Gathering "known" ratings for matrix
- How to collect the data in the utility matrix

(2) Extrapolate unknown ratings from the known ones
- Mainly interested in high unknown ratings
- We are not interested in knowing what you don't like but what you like

(3) Evaluating extrapolation methods
- How to measure success/performance of recommendation methods

(1) Gathering Ratings

- **Explicit**
  - Ask people to rate items
  - Doesn't work well in practice – people can't be bothered
- **Implicit**
  - Learn ratings from user actions
    - E.g., purchase implies high rating
  - What about low ratings?

http://www.mmds.org

(2) Extrapolating Utilities

- **Key problem:** Utility matrix ***U*** is **sparse**
  - Most people have not rated most items
  - **Cold start:**
    - New items have no ratings
    - New users have no history
- **Three approaches to recommender systems:**

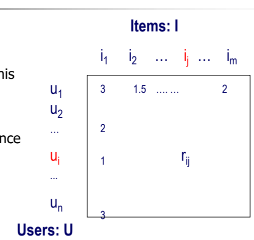
  1) Content-based
  2) Collaborative
  3) Hybrid Models

Collaborative Filtering

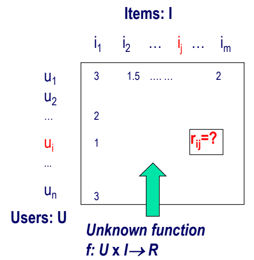
- Method
  - Correlation between user's interests (such as votes and trails)
    - Results are captured in a generally sparse matrix (users x items)
  - Similarities between items
- Problems
  - Sparsity
  - Cold-start
  - Diversity

Data Structures

- Each user has a profile
- Users rate items
  - Explicitly: a user consciously express his or her preference for a title
    - score from 1..5
  - Implicitly: interprete user behavior or selections to impute a vote or preference using web usage mining
    - Time spent in viewing the item
    - Navigation path
    - purchase history
    - Etc…



Collaborative Filtering: A Framework



The task:
Q1: Find Unknown ratings?
Q2: Which items should we recommend to this user?
.
.
.

Collaborative Filtering

- User-Centric
  - The preferences of a large group of people are registered. These preferences could be items bought by the user on a e-commerce Web site, or Web pages visited by the user
  - According to a similarity metric a subgroup of people are selected whose preferences are similar to the current user's preferences.
  - An average of the preferences of that subgroup is calculated.
  - Options on which the current user has no experience yet are selected to generate a recommendation set.
- Item Centric
  - Build an item-item matrix
  - Determine the relationship between pairs of items

Algorithms for CF

- r_{ij}= vote of user i on item j
- I_i = items for which user i has voted
- Mean vote for user i is

$$\bar{r}_i = \frac{1}{|I_i|}\sum_{j \in I_i} r_{ij}$$

- Predicted vote for "active user" a is weighted sum

$$r_{aj} = \bar{r}_a + \kappa \sum_{i=1}^{n} \underbrace{w(a,i)}_{\text{similarity between user } i \text{ and active user } a}(r_{ij} - \bar{r}_i)$$

normalizer

How to Measure Similarity?

- k-nearest neighbour:
  - Compute distance between all other users and active user

$$d_{ai} = \sum_{j=1}^{M}(r_{aj} - r_{ij})^2$$

- aggregate ratings from K nearest neighbors to predict active user's rating

$$w_k(a,i) = \begin{cases} 1 & \text{if } i \in neighbors(a) \\ 0 & else \end{cases}$$

Finding "Similar" Users

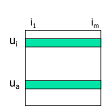
r_a = [*, _, _, *, ***]
r_i = [*, _, **, **, _]

- Let r_a be the vector of user a's ratings
- **Jaccard similarity measure**
  - **Problem:** Ignores the value of the rating
- **Cosine similarity measure**
  - $w_c(a,i) = \frac{r_a \cdot r_i}{||r_a|| \cdot ||r_i||}$
  - **Problem:** Treats missing ratings as "negative"
- **Pearson correlation coefficient**
  - $S_{ai} = I_a \cap I_i$ = items rated by both users a and i

r_a, r_i as sets:
r_a = {1, 4, 5}
r_i = {1, 3, 4}

r_a, r_i as points:
r_a = {1, 0, 0, 1, 3}
r_i = {1, 0, 2, 2, 0}

$$w_p(a,i) = \frac{\sum_{j \in S_{ai}}(r_{aj} - \bar{r}_a)(r_{ij} - \bar{r}_i)}{\sqrt{\sum_{j \in S_{ai}}(r_{aj} - \bar{r}_a)^2 \sum_{j \in S_{ai}}(r_{ij} - \bar{r}_i)^2}}$$



http://www.mmds.org

Item-Item Collaborative Filtering

- **So far: User-user collaborative filtering**
- **Another view: Item-item**
  - For item i, find other similar items
  - Estimate rating for item i based on ratings for similar items
  - Can use same similarity metrics and prediction functions as in user-user model

$$\boxed{r_{ai} = \frac{\sum_{j \in N(i;a)} s_{ij} \cdot r_{aj}}{\sum_{j \in N(i;a)} s_{ij}}}$$

s_{ij}… similarity of items i and j
r_{aj}…rating of user a on item j
$N(i;a)$… set items rated by a similar to i

http://www.mmds.org

Item-Item CF (|N|=2)

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3			5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

- unknown rating
- rating between 1 to 5

http://www.mmds.org

Item-Item CF (|N|=2)

users

movies	1	2	3	4	**5**	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

- estimate rating of movie **1** by user **5**

http://www.mmds.org

Item-Item CF (|N|=2)

users

movies	1	2	3	4	**5**	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Neighbor selection:
Identify movies similar to movie **1**, **rated by user 5**

Here we use Pearson correlation as similarity:
1) Subtract mean rating m_i from each movie i
$m_1 = (1+3+5+5+4)/5 = 3.6$
row 1: [-2.6, 0, -0.6, 0, 0, 1.4, 0, 0, 1.4, 0, 0.4, 0]
2) Compute cosine similarities between rows

http://www.mmds.org

Item-Item CF (|N|=2)

users

movies	1	2	3	4	5	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Compute similarity weights:
$s_{1,3}=0.41$, $s_{1,6}=0.59$

http://www.mmds.org

Item-Item CF (|N|=2)

users

movies	1	2	3	4	5	6	7	8	9	10	11	12	
1	1		3		2.6	5			5		4		
2			5	4			4			2	1	3	
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		
5			4	3	4	2					2	5	
6	1		3		3			2			4		0.59

Predict by taking weighted average:

$r_{5,1}$ = (0.41*2 + 0.59*3) / (0.41+0.59) = 2.6

$$r_{ai} = \frac{\sum_{j \in N(i;a)} s_{ij} \cdot r_{aj}}{\sum s_{ij}}$$

http://www.mmds.org

CF: Common Practice

Before:

$$r_{ai} = \frac{\sum_{j \in N(i;a)} s_{ij} r_{aj}}{\sum_{j \in N(i;a)} s_{ij}}$$

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

Global Effect

Local Effect

$$r_{ai} = b_{ai} + \frac{\sum_{j \in N(i;a)} s_{ij} \cdot (r_{aj} - b_{aj})}{\sum_{j \in N(i;a)} s_{ij}}$$

baseline estimate for r_{ai}

$$b_{ai} = \mu + b_a + b_i$$

- μ = overall mean movie rating
- b_a = rating deviation of user x = (*avg. rating of user* a) $-\mu$
- b_i = rating deviation of movie i

http://www.mmds.org

Modeling Local & Global Effects

- **Global:**
  - Mean movie rating: **3.7 stars**
  - *The Sixth Sense* is **0.5** stars above avg.
  - Joe rates **0.2** stars below avg.
    ⇒ Baseline estimation:
    ***Joe* will rate *The Sixth Sense* 4 stars**
- **Local neighborhood (CF/NN):**
  - *Joe* didn't like related movie *Signs*
  - **⇒ Final estimate:**
    ***Joe* will rate *The Sixth Sense* 3.8 stars**

$b_{ai} = \mu + b_a + b_i$=3.7+(-0.2)+0.5=4

- μ = overall mean movie rating
- b_a = rating deviation of user x
  = (*avg. rating of user* a) $- \mu$
- b_i = rating deviation of movie i

http://www.mmds.org

Pros/Cons of Collaborative Filtering

$$r_{ai} = b_{ai} + \frac{\sum_{j \in N(i;a)} s_{ij} \cdot (r_{aj} - b_{aj})}{\sum_{j \in N(i;a)} s_{ij}}$$

- **+ Works for any kind of item**
  - No feature selection needed
- **- Cold Start:**
  - Need enough users in the system to find a match
- **- Sparsity:**
  - The user/ratings matrix is sparse
  - Hard to find users that have rated the same items
- **- First rater:**
  - Cannot recommend an item that has not been previously rated
  - New items, Esoteric items
- **- Popularity bias:**
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items
- **- Similarity:**
  - Similarity measures are "arbitrary": Pairwise similarities neglect interdependencies among users

http://www.mmds.org

Idea: Interpolation Weights w_{ij}

- Use a **weighted sum** rather than **weighted avg.**:

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

- **A few notes:**
  - $N(i;x)$ … set of movies rated by user x that are similar to movie i
  - w_{ij} is the interpolation weight (some real number)
    - We allow: $\sum_{j \in N(i,x)} w_{ij} \neq 1$
  - w_{ij} models interaction between pairs of movies (it does not depend on user x)

http://www.mmds.org

Recommendations via Optimization

- **Idea: Let's set values w such that they work well on known (user, item) ratings**
- **How to find such values w?**
- **Idea:** Define an objective function and solve the optimization problem

- Find w_{ij} that minimize **SSE** on **training data**!

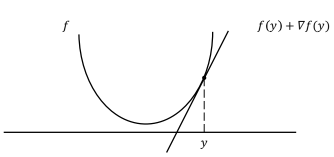
$$J(w) = \sum_{x,i} \Big(\underbrace{\Big[b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj}) \Big]}_{\text{Predicted rating}} - \underbrace{r_{xi}}_{\text{True rating}} \Big)^2$$

- Think of w as a vector of numbers

http://www.mmds.org

Detour: Minimizing a function

- **A simple way to minimize a function $f(x)$:**
  - Compute the take a derivative ∇f
  - **Start at some point y and evaluate $\nabla f(y)$**
  - **Make a step in the reverse direction of the gradient: $y = y - \nabla f(y)$**
  - **Repeat until converged**



http://www.mmds.org

Interpolation Weights

- **We have the optimization problem, now what?**

$$J(w) = \sum_{x} \Big(\Big[b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj}) \Big] - r_{xi} \Big)^2$$

- **Gradient decent:**
  - **Iterate until convergence:** $w \leftarrow w - \eta \nabla_w J$
  - **where $\nabla_w J$ is the gradient (derivative evaluated on data):**

η … learning rate

$$\nabla_w J = \left[\frac{\partial J(w)}{\partial w_{ij}} \right]$$

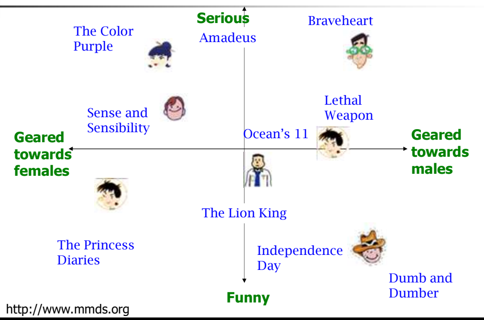
$$= 2 \sum_{x,i} \Big(\Big[b_{xi} + \sum_{k \in N(i;x)} w_{ik}(r_{xk} - b_{xk}) \Big] - r_{xi} \Big) (r_{xj} - b_{xj})$$

for $j \in \{N(i;x), \forall i, \forall x\}$
else $\frac{\partial J(w)}{\partial w_{ij}} = 0$

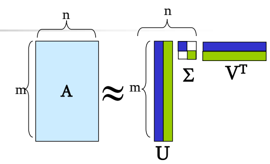
  - **Note:** We fix movie i, go over all r_{xi}, for every movie $j \in N(i;x)$, we compute $\frac{\partial J(w)}{\partial w_{ij}}$

while $|w_{new} - w_{old}| > \varepsilon$:
  $w_{old} = w_{new}$
  $w_{new} = w_{old} - \eta \cdot \nabla w_{old}$

Latent Factor Models (e.g., SVD)



Recap: SVD



- **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: $R \approx Q \cdot P^T$

$A = R,\ Q = U,\ P^T = \Sigma\, V^T$

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

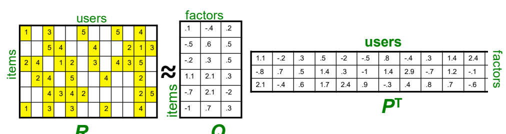
$$\min_{U,V,\Sigma} \sum_{ij \in A} \left(A_{ij} - [U\Sigma V^T]_{ij}\right)^2$$

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

Latent Factor Models

- "SVD" on Netflix data: **R** $\approx$ ***Q* · *P*^{T}**

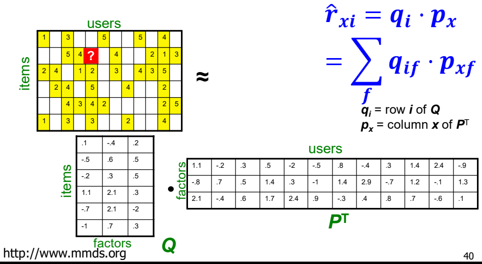
SVD: $A = U\,\Sigma\,V^T$



- For now let's assume we can approximate the rating matrix ***R*** as a product of "thin" ***Q* · *P*^{T}**
  - ***R*** has missing entries but let's ignore that for now!
    - Basically, we will want the reconstruction error to be small on known ratings and we don't care about the values on the missing ones

Ratings as Products of Factors

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

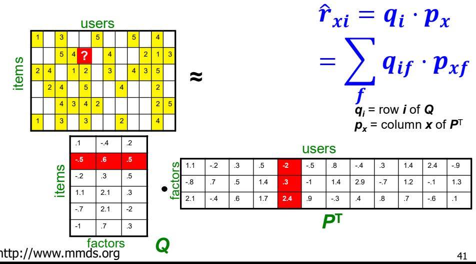


$$\hat{r}_{xi} = q_i \cdot p_x = \sum_f q_{if} \cdot p_{xf}$$

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

Ratings as Products of Factors

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

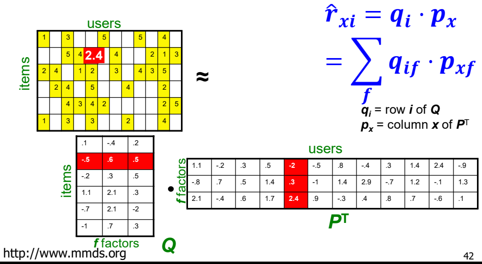


$$\hat{r}_{xi} = q_i \cdot p_x = \sum_f q_{if} \cdot p_{xf}$$

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

Ratings as Products of Factors

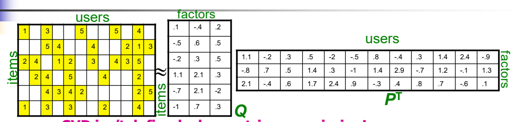
- **How to estimate the missing rating of user *x* for item *i*?**



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

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

Latent Factor Models



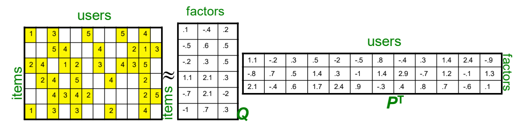
- **SVD isn't defined when entries are missing!**
- **Use specialized methods to find *P, Q***
  - $\min_{P,Q} \sum_{(i,x)\in R} (r_{xi} - q_i \cdot p_x)^2$
  - **Note:**
    - We don't require cols of ***P, Q*** to be orthogonal/unit length
    - ***P, Q*** map users/movies to a latent space
    - The most popular model among Netflix contestants

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

Latent Factor Models

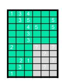
- **Our goal is to find P and Q such tat:**

$$\min_{P,Q} \sum_{(i,x)\in R} (r_{xi} - q_i \cdot p_x)^2$$

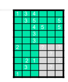


Back to Our Problem

- **Want to minimize SSE for unseen test data**
- **Idea: Minimize SSE on training data**
  - Want large ***k*** (# of factors) to capture all the signals
  - But, **SSE** on **test data** begins to rise for ***k*** > 2
- This is a classical example of **overfitting:**
  - With too much freedom (too many free parameters) the model starts fitting noise
    - That is it fits too well the training data and thus **not generalizing** well to unseen test data



Dealing with Missing Entries



- **To solve overfitting we introduce regularization:**
  - Allow rich model where there are sufficient data
  - Shrink aggressively where data are scarce

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

λ_1, λ_2 … user set regularization parameters

Note: We do not care about the "raw" value of the objective function, but we care in P,Q that achieve the minimum of the objective

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

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

Stochastic Gradient Descent

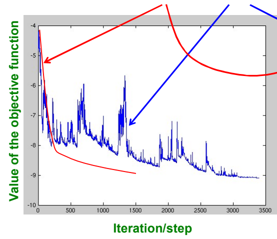
- **Gradient Descent (GD) vs. Stochastic GD**
  - **Observation:** $\nabla Q = [\nabla q_{if}]$ where

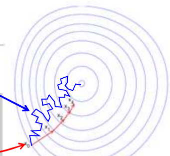
$$\nabla q_{if} = \sum_{x,i} -2(r_{xi} - q_{if} p_{xf}) p_{xf} + 2\lambda q_{if} = \sum_{x,i} \nabla \boldsymbol{Q}(\boldsymbol{r_{xi}})$$

    - Here $\boldsymbol{q_{if}}$ is entry ***f*** of row $\boldsymbol{q_i}$ of matrix ***Q***
  - $\boldsymbol{Q} = \boldsymbol{Q} - \eta \nabla \boldsymbol{Q} = \boldsymbol{Q} - \eta \left[\sum_{x,i} \nabla \boldsymbol{Q}(\boldsymbol{r_{xi}}) \right]$
  - **Idea:** Instead of evaluating gradient over all ratings evaluate it for each individual rating and make a step
- **GD:** $\boldsymbol{Q} \leftarrow \boldsymbol{Q} - \eta \left[\sum_{\boldsymbol{r_{xi}}} \nabla \boldsymbol{Q}(\boldsymbol{r_{xi}}) \right]$
- **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

SGD vs. GD

- **Convergence of GD vs. SGD**





GD improves the value of the objective function at every step.
SGD improves the value but in a "noisy" way.
GD takes fewer steps to converge but each step takes much longer to compute.
In practice, **SGD** is much faster!

Stochastic Gradient Descent

- **Stochastic gradient decent:**
  - Initialize P and Q (using SVD, pretend missing ratings are 0)
  - Then iterate over the ratings (multiple times if necessary) and update factors:

  For each r_{xi}:
    - $\varepsilon_{xi} = 2(r_{xi} - q_i \cdot p_x)$ (derivative of the "error")
    - $q_i \leftarrow q_i + \eta_1 (\varepsilon_{xi}\, p_x - \lambda_2\, q_i)$ (update equation)
    - $p_x \leftarrow p_x + \eta_2 (\varepsilon_{xi}\, q_i - \lambda_1\, p_x)$ (update equation)
- **2 for loops:**
  - For until convergence:
    - For each $\mathbf{r_{xi}}$
      - Compute gradient, do a "step"

η… learning rate

Putting It All Together

$$r_{xi} = \underbrace{\mu}_{\text{Overall mean rating}} + \underbrace{b_x}_{\text{Bias for user } x} + \underbrace{b_i}_{\text{Bias for movie } i} + \underbrace{q_i \cdot p_x}_{\text{User-Movie interaction}}$$

- **Example:**
  - Mean rating: $\mu = 3.7$
  - You are a critical reviewer: your ratings are 1 star lower than the mean: $b_x = -1$
  - Star Wars gets a mean rating of *0.5* higher than average movie: $b_i = + 0.5$
  - Predicted rating for you on Star Wars: $= 3.7 - 1 + 0.5 = 3.2$

Fitting the New Model

- **Solve:**

$$\min_{Q,P} \sum_{(x,i)\in R} \left(r_{xi} - (\mu + b_x + b_i + q_i\, p_x)\right)^2$$

goodness of fit

$$+ \left(\lambda_1 \sum_i \|q_i\|^2 + \lambda_2 \sum_x \|p_x\|^2 + \lambda_3 \sum_x \|b_x\|^2 + \lambda_4 \sum_i \|b_i\|^2\right)$$

regularization

λ is selected via grid-search on a validation set

- **Stochastic gradient decent to find parameters**
  - **Note:** Both biases b_x, b_i as well as interactions q_i, p_x are treated as parameters (we estimate them)

Evaluation Metrics for Recommendation Systems

- $Recall@K = \frac{Number\ of\ Relevant\ Items\ in\ Top\ K}{Total\ Number\ of\ Relevant\ Items}$
- $Precision@K = \frac{Number\ of\ Relevant\ Items\ in\ Top\ K}{K}$
- $F1@K = \frac{2 \times Precision@K \times Recall@K}{Precision@K + Recall@K}$

A movie recommender system: Recommend 10 movies for every user. A user has seen 5 movies, the recommendation list has 3 of them.

$$Recall@K = \frac{3}{5} = 0.6$$
$$Precision@K = \frac{3}{10} = 0.3$$

- Mean Average Precision

$$MAP = \frac{1}{|Users|} \sum_{u=1}^{|Users|} AP_u$$
$$AP = \frac{1}{Q} \sum_{q=1}^{K} P@q$$

A movie recommender system: Recommend K movies, number of relevant items Q
Recommendation List: {Star Wars-Return of Jedi, Back To the Future, The Matrix}
Ground Truth: {Terminator 2, Back To the Future, The Matrix}

$$Precision@K = [0, 1/2, 2/3]$$
$$AP = (1/3)[(1/2) + (2/3)] = 0.38$$

Evaluation Metrics for Recommendation Systems

- Mean Reciprocal Rank: the position of the first relevant item in the recommendation list

$$MRR = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{rank_i}$$

A movie recommender system: Recommend K movies, number of relevant items N
Recommendation List: {Star Wars-Return of Jedi, Back To the Future, The Matrix}
Ground Truth: {Back To the Future, The Matrix}

$$MRR = 1/2 \times [(1/2) + (1/3)] = 0.41$$

- Normalized Cumulative Discounted Gain (NDCG): a measure of how good a ranked list is
- $NDCG@k = \frac{DCG@k}{IDCG@k}$
- $DCG@k = \sum_{i=1}^{K} \frac{rel(i)}{\log_2(i+1)}$

$rel(i)$: relevancy score of item i
$IDCG@k$: $DCG@k$ of the "ideal" recommendation algorithm
Ground Truth: {Terminator 2, Back To the Future, The Matrix}
rel(Terminator 2) = 1 rel(Back To the Future) = 1 rel(The Matrix) = 1
List of Recommendations: {Star Wars-Return of Jedi, Back To the Future, The Matrix}

$$DCG@3 = \frac{0}{\log_2(1+1)} + \frac{1}{\log_2(2+1)} + \frac{1}{\log_2(3+1)}$$
$$IDCG@3 = \frac{1}{\log_2(1+1)} + \frac{1}{\log_2(2+1)} + \frac{1}{\log_2(3+1)}$$
$$NDCG@3 = \frac{DCG@3}{IDCG@3}$$

References

- Lecture slides of Mining Massive Datasets

55