Security for Recommender Systems

Some slides are from Dietmar Jannach, Markus Zanker, Alexander Felfernig, Gerhard Friedrich

Introduction

- **Why recommender systems**
  - Addressing information overload
  - Match users with items
- **Categories**
  - Collaborative filtering
  - Content based
  - Hybrid

Collaborative Filtering (CF)

- **The most prominent approach to generate recommendations**
  - used by large, commercial websites
  - 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 items (implicitly or explicitly)
  - Customers who had similar tastes in the past, will have similar tastes in the future

Problem setup

- **Input**
  - Only a matrix of given user–item ratings
- **Output types**
  - A (numerical) prediction indicating to what degree the current user will like or dislike a certain item
  - A top-N list of recommended items

Explicit ratings

- Probably the most precise ratings
- Most commonly used : 1 to 5
- Main problems
  - Users not always willing to rate many items
    - number of available ratings could be too small → sparse rating matrices → poor recommendation quality

Implicit ratings

- Typically collected by the web service or application in which the recommender system is embedded
- When a customer buys an item, for instance, many recommender systems interpret this behavior as a positive rating
- Clicks, page views, time spent on some page, demo downloads …
- Implicit ratings can be collected constantly and do not require additional efforts from the side of the user
- Main problem
  - One cannot be sure whether the user behavior is correctly interpreted
  - For example, a user might not like all the books he or she has bought; the user also might have bought a book for someone else

Collaborative Filtering Approaches

- User-based nearest-neighbor
- Item-based nearest-neighbor
- Graph-based
- Matrix factorization
- Association Rule Mining
- Neural network

User-based nearest-neighbor collaborative filtering (1)

- **The basic technique**
  - Given an "active user" (Alice) and an item i not yet seen by Alice
    - find a set of users (peers/nearest neighbors) 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
- **Basic assumption and idea**
  - If users had similar tastes in the past they will have similar tastes in the future
  - User preferences remain stable and consistent over time

User-based nearest-neighbor collaborative filtering (2)

- **Example**
  - A database of ratings of the current user, Alice, and some other users is given:

	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

  - Determine whether Alice will like or dislike *Item5*, which Alice has not yet rated or seen

User-based nearest-neighbor collaborative filtering (3)

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

Measuring user similarity (1)

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

  a, b : users

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

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

  - Possible similarity values between -1 and 1

Measuring user similarity (2)

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

  a, b : users

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

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

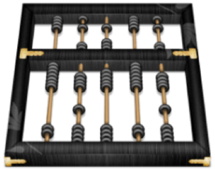
  – Possible similarity values between -1 and 1

	Item1	Item2	Item3	Item4	Item5	
Alice	5	3	4	4	?	
User1	3	1	2	3	3	sim = 0,85
User2	4	3	4	3	5	sim = 0,00
User3	3	3	1	5	4	sim = 0,70
User4	1	5	5	2	1	sim = -0,79

Making predictions

- A common prediction function:

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



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

Item-based collaborative filtering

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

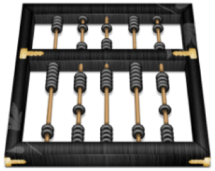
	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

The cosine similarity measure

- **Produces better results in item-to-item filtering**
- **Ratings are seen as vector in n-dimensional space**
- **Similarity is calculated based on the angle between the vectors**

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

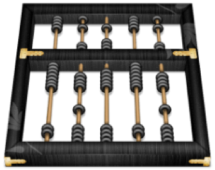
- **Adjusted cosine similarity**
  - take average user ratings into account, transform the original ratings
  - U: set of users who have rated both items a and b



Making predictions

- A common prediction function:

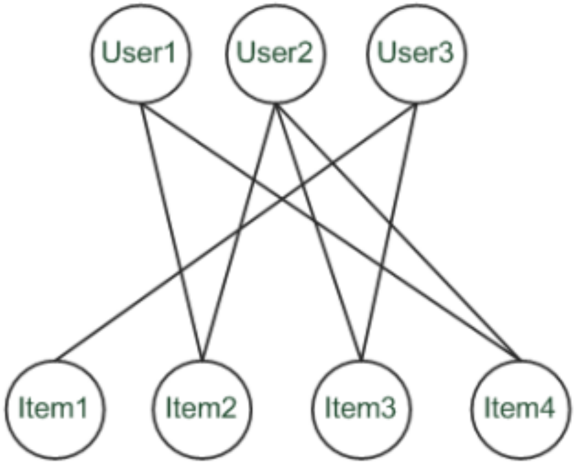
$$pred(u,p) = \frac{\sum_{i \in ratedItem(u)} sim(i,p) * r_{u,i}}{\sum_{i \in ratedItem(u)} sim(i,p)}$$



- Neighborhood size is typically also limited to a specific size
- Not all neighbors are taken into account for the prediction
- An analysis of the MovieLens dataset indicates that "in most real-world situations, a neighborhood of 20 to 50 neighbors seems reasonable" (Herlocker et al. 2002)

Graph-based methods

- Use graph to model user-item interactions
- Compute graph-based similarity scores



Matrix factorization

- Informally, the SVD theorem (Golub and Kahan 1965) states that a given matrix M can be decomposed into a product of three matrices as follows

$$M = U \times \Sigma \times V^T$$

  - where U and V are called *left* and *right singular vectors* and the values of the diagonal of Σ are called the *singular values*
- We can approximate the full matrix by observing only the most important features – those with the largest singular values

Example for SVD-based recommendation

- **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	Terminator	Die Hard	Twins	Eat Pray Love	Harry Potter
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

Threat model for poisoning attacks

- **Attacker's goal**
  - Individuals may be interested to push some items by manipulating the recommender system
  - Individuals might be interested to decrease the rank of competitors' items
  - Some simply might may want to sabotage the system ..
  - Manipulation of the "Internet opinion"
- **Attacker's background knowledge**
  - Complete/partial user-item rating matrix
  - Recommendation algorithm
- **Attacker's capability**
  - (Automatically) create numerous fake accounts / profiles
- **Different names**
  - Shilling attacks
  - Poisoning attacks

Key challenge

- **How to craft rating scores for the fake accounts**

- **Not detected**

Example attack

- **Assume that a user-based nearest neighbor is used with:**
  - Pearson correlation as similarity measure
  - Neighborhood size of 1
    - Only opinion of most similar user will be used to make prediction

	Item1	Item2	Item3	Item4	…	Target	Pearson
Alice	5	3	4	1	…	**?**	
User1	3	1	2	5	…	5	-0.54
User2	4	3	3	3	…	2	0.68
User3	3	3	1	5	…	4	-0.72
User4	1	5	5	2	…	1	-0.02

Example attack

- **Assume that a user-based nearest neighbor is used with:**
  - Pearson correlation as similarity measure
  - Neighborhood size of 1
    - Only opinion of most similar user will be used to make prediction

	Item1	Item2	Item3	Item4	…	Target	Pearson
Alice	5	3	4	1	…	**?**	
User1	3	1	2	5	…	5	-0.54
User2	4	3	3	3	…	2	0.68
User3	3	3	1	5	…	4	-0.72
User4	1	5	5	2	…	1	-0.02

← **User2** most similar to Alice

Example attack

- **Assume that a user-based nearest neighbor is used with:**
  - Pearson correlation as similarity measure
  - Neighborhood size of 1
    - Only opinion of most similar user will be used to make prediction

	Item1	Item2	Item3	Item4	…	Target	Pearson
Alice	5	3	4	1	…	**?**	
User1	3	1	2	5	…	5	-0.54
User2	4	3	3	3	…	2	0.68
User3	3	3	1	5	…	4	-0.72
User4	1	5	5	2	…	1	-0.02
Attack	**5**	**3**	**4**	**3**	**…**	**5**	**0.87**

← **User2** most similar to Alice

Attack

Example attack

- **Assume that a user-based nearest neighbor is used with:**
  - Pearson correlation as similarity measure
  - Neighborhood size of 1
    - Only opinion of most similar user will be used to make prediction

	Item1	Item2	Item3	Item4	…	Target	Pearson
Alice	5	3	4	1	…	**?**	
User1	3	1	2	5	…	5	-0.54
User2	4	3	3	3	…	2	[illegible]
User3	3	3	1	5	…	4	-0.72
User4	1	5	5	2	…	1	-0.02
Attack	**5**	**3**	**4**	**3**	**…**	**5**	**0.87**

← **User2** most similar to Alice

Attack

← **Attack** most similar to Alice

Algorithm-independent attacks: The Random Attack

- **General scheme of an attack profile**

Item1	…	ItemL	…	ItemN	Target
r_1	…	r_l	…	r_n	X
filler items			unrated items		

  – Attack models mainly differ in the way the profile sections are filled

- **Random attack model**
  – Take random values for filler items
    - Typical distribution of ratings is known, e.g., for the movie domain (Average 3.6, standard deviation around 1.1)
  – Idea:
    - generate profiles with "typical" ratings so they are considered as neighbors to many other real profiles
  – High/low ratings for target items
  – Limited effect compared with more advanced models

Algorithm-independent attacks: The Average Attack

- use the individual item's rating average for the filler items
- intuitively, there should be more neighbors
- additional cost involved: find out the average rating of an item

Algorithm-dependent attacks

- **Formulating as a bi-level optimization problem**

- **Objective: maximizing #users the target item is recommended to**
- **Constraints**
  - n fake users
  - Each user rates m filler items
  - Recommendation is calculated by a specific algorithm