This chapter is from *Social Media Mining: An Introduction*.
By Reza Zafarani, Mohammad Ali Abbasi, and Huan Liu.
Cambridge University Press, 2014. Draft version: April 20, 2014.
Complete Draft and Slides Available at: http://dmml.asu.edu/smm

Chapter 9

Recommendation in Social Media

Individuals in social media make a variety of decisions on a daily basis. These decisions are about buying a product, purchasing a service, adding a friend, and renting a movie, among others. The individual often faces many options to choose from. These diverse options, the pursuit of optimality, and the limited knowledge that each individual has create a desire for external help. At times, we resort to search engines for recommendations; however, the results in search engines are rarely tailored to our particular tastes and are query-dependent, independent of the individuals who search for them.

Applications and algorithms are developed to help individuals decide easily, rapidly, and more accurately. These algorithms are tailored to individuals' tastes such that customized recommendations are available for them. These algorithms are called *recommendation algorithms* or *recommender systems*.

Recommender System

Recommender systems are commonly used for product recommendation. Their goal is to recommend products that would be interesting to individuals. Formally, a recommendation algorithm takes a set of users U and a set of items I and learns a function f such that

$$f : U \times I \rightarrow \mathbb{R} \tag{9.1}$$

In other words, the algorithm learns a function that assigns a real value to each user-item pair (u, i), where this value indicates how interested user u is in item i. This value denotes the *rating* given by user u to item i. The recommendation algorithm is not limited to item recommendation and

can be generalized to recommending people and material, such as, ads or content.

Recommendation vs. Search

When individuals seek recommendations, they often use web search engines. However, search engines are rarely tailored to individuals' needs and often retrieve the same results as long as the search query stays the same. To receive accurate recommendation from a search engine, one needs to send accurate keywords to the search engine. For instance, the query ``best 2013 movie to watch'' issued by an 8-year old and an adult will result in the same set of movies, whereas their individual tastes dictate different movies.

Recommendation systems are designed to recommend individual-based choices. Thus, the same query issued by different individuals should result in different recommendations. These systems commonly employ browsing history, product purchases, user profile information, and friends information to make customized recommendations. As simple as this process may look, a recommendation system algorithm actually has to deal with many challenges.

9.1 Challenges

Recommendation systems face many challenges, some of which are presented next:

- **Cold-Start Problem.** Many recommendation systems use historical data or information provided by the user to recommend items, products, and the like. However, when individuals first join sites, they have not yet bought any product: they have no history. This makes it hard to infer what they are going to like when they start on a site. The problem is referred to as the *cold-start* problem. As an example, consider an online movie rental store. This store has no idea what recently joined users prefer to watch and therefore cannot recommend something close to their tastes. To address this issue, these sites often ask users to rate a couple of movies before they begin recommend others to them. Other sites ask users to fill in profile

information, such as interests. This information serves as an input to the recommendation algorithm.

- **Data Sparsity.** Similar to the cold-start problem, data sparsity occurs when not enough historical or prior information is available. Unlike the cold-start problem, data sparsity relates to the system as a whole and is not specific to an individual. In general, data sparsity occurs when a few individuals rate many items while many other individuals rate only a few items. Recommender systems often use information provided by other users to help offer better recommendations to an individual. When this information is not reasonably available, then it is said that a *data sparsity* problem exists. The problem is more promi- nent in sites that are recently launched or ones that are not popular.

- **Attacks.** The recommender system may be attacked to recommend items otherwise not recommended. For instance, consider a system that recommends items based on similarity between ratings (e.g., lens A is recommended for camera B because they both have rating 4). Now, an attacker that has knowledge of the recommendation algorithm can create a set of fake user accounts and rate lens C (which is not as good as lens A) highly such that it can get rating 4. This way the recommendation system will recommend C with camera B as well as A. This attack is called a *push attack*, because it pushes the ratings up such that the system starts recommending items that would otherwise not be recommended. Other attacks such as *nuke* attacks attempt to stop the whole recommendation system algorithm and make it unstable. A recommendation system should have the means to stop such attacks.

  Nuke Attack and Push Attack

- **Privacy.** The more information a recommender system has about the users, the better the recommendations it provides to the users. However, users often avoid revealing information about themselves due to privacy concerns. Recommender systems should address this challenge while protecting individuals' privacy.

- **Explanation.** Recommendation systems often recommend items without having an explanation why they did so. For instance, when

several items are bought together by many users, the system recommends these to new users items together. However, the system does not know why these items are bought together. Individuals may prefer some reasons for buying items; therefore, recommendation algorithms should provide explanation when possible.

9.2 Classical Recommendation Algorithms

Classical recommendation algorithms have a long history on the web. In recent years, with the emergence of social media sites, these algorithms have been provided new information, such as friendship information, interactions, and so on. We review these algorithms in this section.

9.2.1 Content-Based Methods

Content-based recommendation systems are based on the fact that a user's interest should match the description of the items that are recommended by the system. In other words, the more similar the item's description to the user's interest, the higher the likelihood that the user is going to find the item's recommendation interesting. *Content-based* recommender systems implement this idea by measuring the similarity between an item's description and the user's profile information. The higher this similarity, the higher the chance that the item is recommended.

To formalize a content-based method, we first represent both user profiles and item descriptions by vectorizing (see Chapter 5) them using a set of k keywords. After vectorization, item j can be represented as a k-dimensional vector $I_j = (i_{j,1}, i_{j,2}, \ldots, i_{j,k})$ and user i as $U_i = (u_{i,1}, u_{i,2}, \ldots, u_{i,k})$. To compute the similarity between user i and item j, we can use cosine similarity between the two vectors U_i and I_j:

$$sim(U_i, I_j) = cos(U_i, I_j) = \frac{\sum_{l=1}^{k} u_{i,l} i_{j,l}}{\sqrt{\sum_{l=1}^{k} {u_{i,l}}^2} \sqrt{\sum_{l=1}^{k} {i_{j,l}}^2}} \tag{9.2}$$

In content-based recommendation, we compute the topmost similar items to a user j and then recommend these items in the order of similarity. Algorithm 9.1 shows the main steps of content-based recommendation.

Algorithm 9.1 Content-based recommendation

Require: User i's Profile Information, Item descriptions for items $j \in \{1, 2, \ldots, n\}$, k keywords, r number of recommendations.

1: **return** r recommended items.
2: $U_i = (u_1, u_2, \ldots, u_k)$ = user i's profile vector;
3: $\{I_j\}_{j=1}^{n} = \{(i_{j,1}, i_{j,2}, \ldots, i_{j,k})$ = item j's description vector$\}_{j=1}^{n}$;
4: $s_{i,j} = sim(U_i, I_j), 1 \leq j \leq n$;
5: Return top r items with maximum similarity $s_{i,j}$.

Table 9.1: User-Item Matrix

	Lion King	Aladdin	Mulan	Anastasia
John	3	0	3	3
Joe	5	4	0	2
Jill	1	2	4	2
Jane	3	?	1	0
Jorge	2	2	0	1

9.2.2 Collaborative Filtering (CF)

Collaborative filtering is another set of classical recommendation techniques. In collaborative filtering, one is commonly given a user-item matrix where each entry is either unknown or is the rating assigned by the user to an item. Table 9.1 is an user-item matrix where ratings for some cartoons are known and unknown for others (question marks). For instance, on a review scale of 5, where 5 is the best and 0 is the worst, if an entry (i, j) in the user-item matrix is 4, that means that user i liked item j.

In collaborative filtering, one aims to predict the missing ratings and possibly recommend the cartoon with the highest predicted rating to the user. This prediction can be performed directly by using previous ratings in the matrix. This approach is called *memory-based* collaborative filtering because it employs historical data available in the matrix. Alternatively, one can assume that an underlying model (hypothesis) governs the way users rate items. This model can be approximated and learned. After the model is learned, one can use it to predict other ratings. The second approach is called *model-based* collaborative filtering.

Memory-Based Collaborative Filtering

In memory-based collaborative filtering, one assumes one of the following (or both) to be true:

- Users with similar **previous** ratings for items are likely to rate future items similarly.
- Items that have received similar ratings **previously** from users are likely to receive similar ratings from future users.

If one follows the first assumption, the memory-based technique is a *user-based* CF algorithm, and if one follows the latter, it is an *item-based* CF algorithm. In both cases, users (or items) collaboratively help filter out irrelevant content (dissimilar users or items). To determine similarity between users or items, in collaborative filtering, two commonly used similarity measures are cosine similarity and Pearson correlation. Let $r_{u,i}$ denote the rating that user u assigns to item i, let $\bar{r}_u$ denote the average rating for user u, and let $\bar{r}_i$ be the average rating for item i. Cosine similarity between users u and v is

$$sim(U_u, U_v) = cos(U_u, U_v) = \frac{U_u \cdot U_v}{||U_u|| \, ||U_v||} = \frac{\sum_i r_{u,i} r_{v,i}}{\sqrt{\sum_i {r_{u,i}}^2} \sqrt{\sum_i {r_{v,i}}^2}}. \tag{9.3}$$

And the Pearson correlation coefficient is defined as

$$sim(U_u, U_v) = \frac{\sum_i (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_i (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_i (r_{v,i} - \bar{r}_v)^2}}. \tag{9.4}$$

Next, we discuss user- and item-based collaborative filtering.

User-Based Collaborative Filtering. In this method, we predict the rating of user u for item i by (1) finding users most similar to u and (2) using a combination of the ratings of these users for item i as the predicted rating of user u for item i. To remove noise and reduce computation, we often limit the number of similar users to some fixed number. These most similar users are called the *neighborhood* for user u, $N(u)$. In user-based collaborative filtering, the rating of user u for item i is calculated as

Neighborhood

$$r_{u,i} = \bar{r}_u + \frac{\sum_{v \in N(u)} sim(u, v)(r_{v,i} - \bar{r}_v)}{\sum_{v \in N(u)} sim(u, v)}, \tag{9.5}$$

where the number of members of $N(u)$ is predetermined (e.g., top 10 most similar members).

Example 9.1. *In Table 9.1, $r_{Jane,Aladdin}$ is missing. The average ratings are the following:*

$$\bar{r}_{John} = \frac{3+3+0+3}{4} = 2.25 \tag{9.6}$$

$$\bar{r}_{Joe} = \frac{5+4+0+2}{4} = 2.75 \tag{9.7}$$

$$\bar{r}_{Jill} = \frac{1+2+4+2}{4} = 2.25 \tag{9.8}$$

$$\bar{r}_{Jane} = \frac{3+1+0}{3} = 1.33 \tag{9.9}$$

$$\bar{r}_{Jorge} = \frac{2+2+0+1}{4} = 1.25. \tag{9.10}$$

Using cosine similarity (or Pearson correlation), the similarity between Jane and others can be computed:

$$sim(Jane, John) = \frac{3\times3+1\times3+0\times3}{\sqrt{10}\sqrt{27}} = 0.73 \tag{9.11}$$

$$sim(Jane, Joe) = \frac{3\times5+1\times0+0\times2}{\sqrt{10}\sqrt{29}} = 0.88 \tag{9.12}$$

$$sim(Jane, Jill) = \frac{3\times1+1\times4+0\times2}{\sqrt{10}\sqrt{21}} = 0.48 \tag{9.13}$$

$$sim(Jane, Jorge) = \frac{3\times2+1\times0+0\times1}{\sqrt{10}\sqrt{5}} = 0.84. \tag{9.14}$$

Now, assuming that the neighborhood size is 2, then Jorge and Joe are the two most similar neighbors. Then, Jane's rating for Aladdin computed from user-based collaborative filtering is

$$\begin{aligned} r_{Jane,Aladdin} &= \bar{r}_{Jane} + \frac{sim(Jane, Joe)(r_{Joe,Aladdin} - \bar{r}_{Joe})}{sim(Jane, Joe) + sim(Jane, Jorge)} \\ &\quad + \frac{sim(Jane, Jorge)(r_{Jorge,Aladdin} - \bar{r}_{Jorge})}{sim(Jane, Joe) + sim(Jane, Jorge)} \\ &= 1.33 + \frac{0.88(4-2.75) + 0.84(2-1.25)}{0.88+0.84} = 2.33 \end{aligned} \tag{9.15}$$

Item-based Collaborative Filtering. In user-based collaborative filtering, we compute the average rating for different users and find the most similar users to the users for whom we are seeking recommendations. Unfortunately, in most online systems, users do not have many ratings; therefore, the averages and similarities may be unreliable. This often results in a different set of similar users when new ratings are added to the system. On the other hand, products usually have many ratings and their average and the similarity between them are more stable. In item-based CF, we perform collaborative filtering by finding the most similar items. The rating of user u for item i is calculated as

$$r_{u,i} = \bar{r}_i + \frac{\sum_{j \in N(i)} sim(i,j)(r_{u,j} - \bar{r}_j)}{\sum_{j \in N(i)} sim(i,j)}, \tag{9.16}$$

where $\bar{r}_i$ and $\bar{r}_j$ are the average ratings for items i and j, respectively.

Example 9.2. *In Table 9.1, $r_{Jane,Aladdin}$ is missing. The average ratings for items are*

$$\bar{r}_{Lion\,King} = \frac{3+5+1+3+2}{5} = 2.8. \tag{9.17}$$

$$\bar{r}_{Aladdin} = \frac{0+4+2+2}{4} = 2. \tag{9.18}$$

$$\bar{r}_{Mulan} = \frac{3+0+4+1+0}{5} = 1.6. \tag{9.19}$$

$$\bar{r}_{Anastasia} = \frac{3+2+2+0+1}{5} = 1.6. \tag{9.20}$$

Using cosine similarity (or Pearson correlation), the similarity between Aladdin and others can be computed:

$$sim(Aladdin, Lion\,King) = \frac{0\times3+4\times5+2\times1+2\times2}{\sqrt{24}\sqrt{39}} = 0.84. \tag{9.21}$$

$$sim(Aladdin, Mulan) = \frac{0\times3+4\times0+2\times4+2\times0}{\sqrt{24}\sqrt{25}} = 0.32. \tag{9.22}$$

$$sim(Aladdin, Anastasia) = \frac{0\times3+4\times2+2\times2+2\times1}{\sqrt{24}\sqrt{18}} = 0.67. \tag{9.23}$$

Now, assuming that the neighborhood size is 2, then Lion King and Anastasia are the two most similar neighbors. Then, Jane's rating for Aladdin computed from item-based collaborative filtering is

$$\begin{aligned} r_{Jane,Aladdin} &= \bar{r}_{Aladdin} + \frac{sim(Aladdin, Lion\ King)(r_{Jane,Lion\ King} - \bar{r}_{Lion\ King})}{sim(Aladdin, Lion\ King) + sim(Aladdin, Anastasia)} \\ &\quad + \frac{sim(Aladdin, Anastasia)(r_{Jane,Anastasia} - \bar{r}_{Anastasia})}{sim(Aladdin, Lion\ King) + sim(Aladdin, Anastasia)} \\ &= 2 + \frac{0.84(3 - 2.8) + 0.67(0 - 1.6)}{0.84 + 0.67} = 1.40. \end{aligned} \tag{9.24}$$

Model-Based Collaborative Filtering

In memory-based methods (either item-based or user-based), one aims to predict the missing ratings based on similarities between users or items. In model-based collaborative filtering, one assumes that an underlying model governs the way users rate. We aim to learn that model and then use that model to predict the missing ratings. Among a variety of model-based techniques, we focus on a well-established model-based technique that is based on singular value decomposition (SVD).

Singular Value Decomposition

SVD is a linear algebra technique that, given a real matrix $X \in \mathbb{R}^{m \times n}$, $m \geq n$, factorizes it into three matrices,

$$X = U\Sigma V^T, \tag{9.25}$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix. The product of these matrices is equivalent to the original matrix; therefore, no information is lost. Hence, the process is *lossless*.

Lossless Matrix Factorization

Let $||X||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij}^2}$ denote the Frobenius norm of matrix X. A low-rank matrix approximation of matrix X is another matrix $C \in \mathbb{R}^{m \times n}$. C approximates X, and C's rank (the maximum number of linearly independent columns) is a fixed number $k \ll min(m, n)$:

Frobenius Norm

$$rank(C) = k. \tag{9.26}$$

The best low-rank matrix approximation is a matrix C that minimizes $||X - C||_F$. Low-rank approximations of matrices remove noise by assuming that the matrix is not generated at random and has an underlying structure.

SVD can help remove noise by computing a low-rank approximation of a matrix. Consider the following matrix X_k, which we construct from matrix X after computing the SVD of $X = U\Sigma V^T$:

1. Create Σ_k from Σ by keeping only the first k elements on the diagonal. This way, $\Sigma_k \in \mathbb{R}^{k \times k}$.

2. Keep only the first k columns of U and denote it as $U_k \in \mathbb{R}^{m \times k}$, and keep only the first k rows of V^T and denote it as ${V_k}^T \in \mathbb{R}^{k \times n}$.

3. Let $X_k = U_k \Sigma_k {V_k}^T$, $X_k \in \mathbb{R}^{m \times n}$.

Eckart-Young-Mirsky Theorem

As it turns out, X_k is the *best low-rank approximation* of a matrix X. The following Eckart-Young-Mirsky theorem outlines this result.

Theorem 9.1 (Eckart-Young-Mirsky Low-Rank Matrix Approximation)**.** *Let X be a matrix and C be the best low-rank approximation of X; if $\|X - C\|_F$ is minimized, and rank$(C) = k$, then $C = X_k$.*

To summarize, the best rank-k approximation of the matrix can be easily computed by calculating the SVD of the matrix and then taking the first k columns of U, truncating Σ to the the first k entries, and taking the first k rows of V^T.

As mentioned, low-rank approximation helps remove noise from a matrix by assuming that the matrix is low rank. In low-rank approximation using SVD, if $X \in \mathbb{R}^{m \times n}$, then $U_k \in \mathbb{R}^{m \times k}$, $\Sigma_k \in \mathbb{R}^{k \times k}$, and $V_k^T \in \mathbb{R}^{k \times n}$. Hence, U_k has the same number of rows as X, but in a k-dimensional space. Therefore, U_k represents rows of X, but in a transformed k-dimensional space. The same holds for V_k^T because it has the same number of columns as X, but in a k-dimensional space. To summarize, U_k and V_k^T can be thought of as k-dimensional representations of rows and columns of X. In this k-dimensional space, noise is removed and more similar points should be closer.

Now, given the user-item matrix X, we can remove its noise by computing X_k from X and getting the new k-dimensional user space U_k or the k-dimensional item space V_k^T. This way, we can compute the most similar neighbors based on distances in this k-dimensional space. The similarity in the k-dimensional space can be computed using cosine similarity or Pearson correlation. We demonstrate this via Example 9.3.

Table 9.2: An User-Item Matrix

	Lion King	Aladdin	Mulan
John	3	0	3
Joe	5	4	0
Jill	1	2	4
Jorge	2	2	0

Example 9.3. *Consider the user-item matrix, in Table 9.2. Assuming this matrix is X, then by computing the SVD of* $X = U\Sigma V^T$,[1] *we have*

$$U = \begin{bmatrix} -0.4151 & -0.4754 & -0.7679 & 0.1093 \\ -0.7437 & 0.5278 & 0.0169 & -0.4099 \\ -0.4110 & -0.6626 & 0.6207 & -0.0820 \\ -0.3251 & 0.2373 & 0.1572 & 0.9018 \end{bmatrix} \tag{9.27}$$

$$\Sigma = \begin{bmatrix} 8.0265 & 0 & 0 \\ 0 & 4.3886 & 0 \\ 0 & 0 & 2.0777 \\ 0 & 0 & 0 \end{bmatrix} \tag{9.28}$$

$$V^T = \begin{bmatrix} -0.7506 & -0.5540 & -0.3600 \\ 0.2335 & 0.2872 & -0.9290 \\ -0.6181 & 0.7814 & 0.0863 \end{bmatrix} \tag{9.29}$$

Considering a rank 2 approximation (i.e., $k = 2$*), we truncate all three matrices:*

$$U_k = \begin{bmatrix} -0.4151 & -0.4754 \\ -0.7437 & 0.5278 \\ -0.4110 & -0.6626 \\ -0.3251 & 0.2373 \end{bmatrix} \tag{9.30}$$

$$\Sigma_k = \begin{bmatrix} 8.0265 & 0 \\ 0 & 4.3886 \end{bmatrix} \tag{9.31}$$

$$V_k^T = \begin{bmatrix} -0.7506 & -0.5540 & -0.3600 \\ 0.2335 & 0.2872 & -0.9290 \end{bmatrix}. \tag{9.32}$$

The rows of U_k *represent users. Similarly the columns of* V_k^T *(or rows of* V_k*) represent items. Thus, we can plot users and items in a 2-D figure. By plotting*

[1] In Matlab, this can be performed using the svd command.

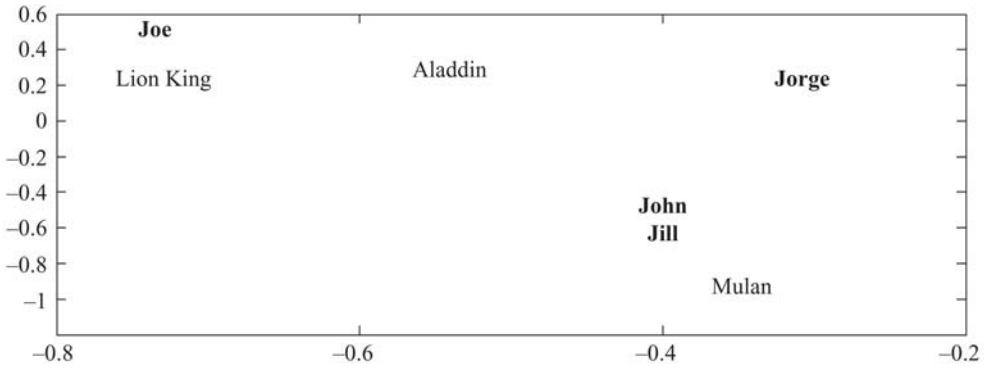

Figure 9.1: Users and Items in the 2-D Space.

user rows or item columns, we avoid computing distances between them and can visually inspect items or users that are most similar to one another. Figure 9.1 depicts users and items depicted in a 2-D space. As shown, to recommend for Jill, John is the most similar individual to her. Similarly, the most similar item to Lion King is Aladdin.

After most similar items or users are found in the lower k-dimensional space, one can follow the same process outlined in user-based or item-based collaborative filtering to find the ratings for an unknown item. For instance, we showed in Example 9.3 (see Figure 9.1) that if we are predicting the rating $r_{Jill,Lion\ King}$ and assume that neighborhood size is 1, item-based CF uses $r_{Jill,Aladdin}$, because Aladdin is closest to Lion King. Similarly, user-based collaborative filtering uses $r_{John,Lion\ King}$, because John is the closest user to Jill.

9.2.3 Extending Individual Recommendation to Groups of Individuals

All methods discussed thus far are used to predict a rating for item i for an individual u. Advertisements that individuals receive via email marketing are examples of this type of recommendation on social media. However, consider ads displayed on the starting page of a social media site. These ads are shown to a large population of individuals. The goal when showing these ads is to ensure that they are interesting to the individuals

who observe them. In other words, the site is advertising to a group of individuals.

Our goal in this section is to formalize how existing methods for recommending to a single individual can be extended to a group of individuals. Consider a group of individuals $G = \{u_1, u_2, \ldots, u_n\}$ and a set of products $I = \{i_1, i_2, \ldots, i_m\}$. From the products in I, we aim to recommend products to our group of individuals G such the recommendation satisfies the group being recommended to as much as possible. One approach is to first consider the ratings predicted for each individual in the group and then devise methods that can aggregate ratings for the individuals in the group. Products that have the highest aggregated ratings are selected for recommendation. Next, we discuss these aggregation strategies for individuals in the group.

Aggregation Strategies for a Group of Individuals

We discuss three major aggregation strategies for individuals in the group. Each aggregation strategy considers an assumption based on which ratings are aggregated. Let $r_{u,i}$ denote the rating of user $u \in G$ for item $i \in I$. Denote R_i as the group-aggregated rating for item i.

Maximizing Average Satisfaction. We assume that products that satisfy each member of the group on average are the best to be recommended to the group. Then, R_i group rating based on the maximizing average satisfaction strategy is given as

$$R_i = \frac{1}{n} \sum_{u \in G} r_{u,i}. \tag{9.33}$$

After we compute R_i for all items $i \in I$, we recommend the items that have the highest R_i's to members of the group.

Least Misery. This strategy combines ratings by taking the minimum of them. In other words, we want to guarantee that no individuals is being recommended an item that he or she strongly dislikes. In least misery, the aggregated rating R_i of an item is given as

$$R_i = \min_{u \in G} r_{u,i}. \tag{9.34}$$

Similar to the previous strategy, we compute R_i for all items $i \in I$ and recommend the items with the highest R_i values. In other words, we prefer recommending items to the group such that no member of the group strongly dislikes them.

Most Pleasure. Unlike the least misery strategy, in the most pleasure approach, we take the maximum rating in the group as the group rating:

$$R_i = \max_{u \in G} r_{u,i}. \tag{9.35}$$

Since we recommend items that have the highest R_i values, this strategy guarantees that the items that are being recommended to the group are enjoyed the most by at least one member of the group.

Example 9.4. *Consider the user-item matrix in Table 9.3. Consider group $G = \{John, Jill, Juan\}$. For this group, the aggregated ratings for all products using average satisfaction, least misery, and most pleasure are as follows.*

Table 9.3: User-Item Matrix

	Soda	Water	Tea	Coffee
John	1	3	1	1
Joe	4	3	1	2
Jill	2	2	4	2
Jorge	1	1	3	5
Juan	3	3	4	5

Average Satisfaction:

$$R_{Soda} = \frac{1+2+3}{3} = 2. \tag{9.36}$$

$$R_{Water} = \frac{3+2+3}{3} = 2.66. \tag{9.37}$$

$$R_{Tea} = \frac{1+4+4}{3} = 3. \tag{9.38}$$

$$R_{Coffee} = \frac{1+2+5}{3} = 2.66. \tag{9.39}$$

Least Misery:

$$R_{Soda} = \min\{1,2,3\} = 1. \tag{9.40}$$

$$R_{Water} = \min\{3,2,3\} = 2. \quad (9.41)$$
$$R_{Tea} = \min\{1,4,4\} = 1. \quad (9.42)$$
$$R_{Coffee} = \min\{1,2,5\} = 1. \quad (9.43)$$

Most Pleasure:

$$R_{Soda} = \max\{1,2,3\} = 3. \quad (9.44)$$
$$R_{Water} = \max\{3,2,3\} = 3. \quad (9.45)$$
$$R_{Tea} = \max\{1,4,4\} = 4. \quad (9.46)$$
$$R_{Coffee} = \max\{1,2,5\} = 5. \quad (9.47)$$

Thus, the first recommended items are tea, water, and coffee based on average satisfaction, least misery, and most pleasure, respectively.

9.3 Recommendation Using Social Context

In social media, in addition to ratings of products, there is additional information available, such as the friendship network among individuals. This information can be used to improve recommendations, based on the assumption that an individual's friends have an impact on the ratings ascribed to the individual. This impact can be due to homophily, influence, or confounding, discussed in Chapter 8. When utilizing this social information (i.e., *social context*) we can (1) use friendship information alone, (2) use social information in addition to ratings, or (3) constrain recommendations using social information. Figure 9.2 compactly represents these three approaches.

9.3.1 Using Social Context Alone

Consider a network of friendships for which no user-item rating matrix is provided. In this network, we can still recommend users from the network to other users for friendship. This is an example of *friend recommendation* in social networks. For instance, in social networking sites, users are often provided with a list of individuals they may know and are asked if they wish to befriend them. How can we recommend such friends?

There are many methods that can be used to recommend friends in social networks. One such method is *link prediction*, which we discuss

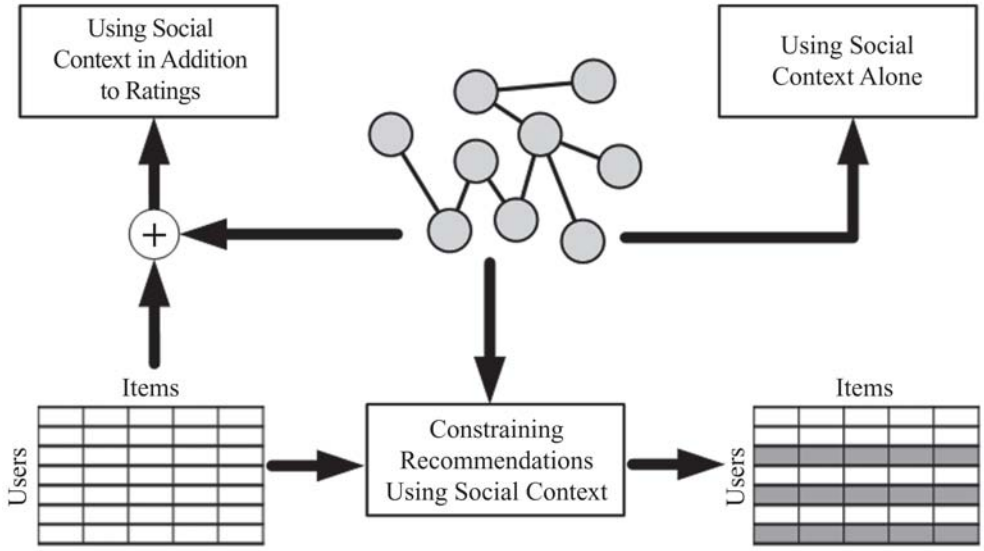

Figure 9.2: Recommendation using Social Context. When utilizing social information, we can 1) utilize this information independently, 2) add it to user-rating matrix, or 3) constrain recommendations with it.

in detail in Chapter 10. We can also use the structure of the network to recommend friends. For example, it is well known that individuals often form triads of friendships on social networks. In other words, two friends of an individual are often friends with one another. A triad of three individuals a, b, and c consists of three edges $e(a,b)$, $e(b,c)$, and $e(c,a)$. A triad that is missing one of these edges is denoted as an *open triad*. To recommend friends, we can find open triads and recommend individuals who are not connected as friends to one another.

9.3.2 Extending Classical Methods with Social Context

Social information can also be used in addition to a user-item rating matrix to improve recommendation. Addition of social information can be performed by assuming that users that are connected (i.e., friends) have similar tastes in rating items. We can model the taste of user U_i using a k-dimensional vector $U_i \in \mathbb{R}^{k\times 1}$. We can also model items in the k-dimensional space. Let $V_j \in \mathbb{R}^{k\times 1}$ denote the item representation in k-dimensional space. We can assume that rating R_{ij} given by user i to item j

can be computed as

$$R_{ij} = U_i^T V_i. \tag{9.48}$$

To compute U_i and V_i, we can use matrix factorization. We can rewrite Equation 9.48 in matrix format as

$$R = U^T V, \tag{9.49}$$

where $R \in \mathbb{R}^{n \times m}$, $U \in \mathbb{R}^{k \times n}$, $V \in \mathbb{R}^{k \times m}$, n is the number of users, and m is the number of items. Similar to model-based CF discussed in Section 9.2.2, matrix factorization methods can be used to find U and V, given user-item rating matrix R. In mathematical terms, in this matrix factorization, we are finding U and V by solving the following optimization problem:

$$\min_{U,V} \frac{1}{2} \|R - U^T V\|_F^2. \tag{9.50}$$

Users often have only a few ratings for items; therefore, the R matrix is very sparse and has many missing values. Since we compute U and V only for nonmissing ratings, we can change Equation 9.50 to

$$\min_{U,V} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} I_{ij} (R_{ij} - U_i^T V_j)^2, \tag{9.51}$$

where $I_{ij} \in \{0, 1\}$ and $I_{ij} = 1$ when user i has rated item j and is equal to 0 otherwise. This ensures that nonrated items do not contribute to the summations being minimized in Equation 9.51. Often, when solving this optimization problem, the computed U and V can estimate ratings for the **already rated** items accurately, but fail at predicting ratings for unrated items. This is known as the *overfitting problem*. The overfitting problem can be mitigated by allowing both U and V to only consider important features required to represent the data. In mathematical terms, this is equivalent to both U and V having small matrix norms. Thus, we can change Equation 9.51 to

Overfitting

$$\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} I_{ij} (R_{ij} - U_i^T V_j)^2 + \frac{\lambda_1}{2} \|U\|_F^2 + \frac{\lambda_2}{2} \|V\|_F^2, \tag{9.52}$$

where $\lambda_1, \lambda_2 > 0$ are predetermined constants that control the effects of matrix norms. The terms $\frac{\lambda_1}{2} \|U\|_F^2$ and $\frac{\lambda_2}{2} \|V\|_F^2$ are denoted as *regularization*

terms. Note that to minimize Equation 9.52, we need to minimize all terms in the equation, including the regularization terms. Thus, whenever one needs to minimize some other constraint, it can be introduced as a new additive term in Equation 9.52. Equation 9.52 lacks a term that incorporates the social network of users. For that, we can add another regularization term,

Regularization Term

$$\sum_{i=1}^{n} \sum_{j \in F(i)} sim(i, j) \|U_i - U_j\|_F^2, \tag{9.53}$$

where $sim(i, j)$ denotes the similarity between user i and j (e.g., cosine similarity or Pearson correlation between their ratings) and $F(i)$ denotes the friends of i. When this term is minimized, it ensures that the taste for user i is close to that of all his friends $j \in F(i)$. As we did with previous regularization terms, we can add this term to Equation 9.51. Hence, our final goal is to solve the following optimization problem:

$$\min_{U,V} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} I_{ij} (R_{ij} - U_i^T V_j)^2 + \beta \sum_{i=1}^{n} \sum_{j \in F(i)} sim(i, j) \|U_i - U_j\|_F^2 + \frac{\lambda_1}{2} \|U\|_F^2 + \frac{\lambda_2}{2} \|V\|_F^2, \tag{9.54}$$

where β is the constant that controls the effect of social network regularization. A local minimum for this optimization problem can be obtained using gradient-descent-based approaches. To solve this problem, we can compute the gradient with respect to U_i's and V_i's and perform a gradient-descent-based method.

9.3.3 Recommendation Constrained by Social Context

In classical recommendation, to estimate ratings of an item, one determines similar users or items. In other words, *any user* similar to the individual can contribute to the predicted ratings for the individual. We can limit the set of individuals that can contribute to the ratings of a user to the set of friends of the user. For instance, in user-based collaborative filtering, we determine a neighborhood of most similar individuals. We can take the intersection of this neighborhood with the set of friends of the individual

Table 9.4: User-Item Matrix

	Lion King	Aladdin	Mulan	Anastasia
John	4	3	2	2
Joe	5	2	1	5
Jill	2	5	?	0
Jane	1	3	4	3
Jorge	3	1	1	2

to attract recommendations only from friends who are *similar enough*:

$$r_{u,i} = \bar{r}_u + \frac{\sum_{v \in N(u) \cap F(u)} sim(u,v)(r_{v,i} - \bar{r}_v)}{\sum_{v \in N(u) \cap F(u)} sim(u,v)}. \tag{9.55}$$

This approach has its own shortcomings. When there is no intersection between the set of friends and the neighborhood of most similar individuals, the ratings cannot be computed. To mitigate this, one can use the set of k *most similar* friends of an individual $S(i)$ to predict the ratings,

$$r_{u,i} = \bar{r}_u + \frac{\sum_{v \in S(u)} sim(u,v)(r_{v,i} - \bar{r}_v)}{\sum_{v \in S(u)} sim(u,v)}. \tag{9.56}$$

Similarly, when friends are not very similar to the individual, the predicted rating can be different from the rating predicted using most similar users. Depending on the context, both equations can be utilized.

Example 9.5. *Consider the user-item matrix in Table 9.4 and the following adjacency matrix denoting the friendship among these individuals.*

$$A = \begin{bmatrix} & John & Joe & Jill & Jane & Jorge \\ John & 0 & 1 & 0 & 0 & 1 \\ Joe & 1 & 0 & 1 & 0 & 0 \\ Jill & 0 & 1 & 0 & 1 & 1 \\ Jane & 0 & 0 & 1 & 0 & 0 \\ Jorge & 1 & 0 & 1 & 0 & 0 \end{bmatrix}, \tag{9.57}$$

We wish to predict $r_{Jill,Mulan}$. We compute the average ratings and similarity between Jill and other individuals using cosine similarity:

$$\bar{r}_{John} = \frac{4+3+2+2}{4} = 2.75. \tag{9.58}$$

$$\bar{r}_{Joe} = \frac{5+2+1+5}{4} = 3.25. \quad (9.59)$$

$$\bar{r}_{Jill} = \frac{2+5+0}{3} = 2.33. \quad (9.60)$$

$$\bar{r}_{Jane} = \frac{1+3+4+3}{4} = 2.75. \quad (9.61)$$

$$\bar{r}_{Jorge} = \frac{3+1+1+2}{4} = 1.75. \quad (9.62)$$

The similarities are

$$sim(Jill, John) = \frac{2\times 4+5\times 3+0\times 2}{\sqrt{29}\sqrt{29}} = 0.79. \quad (9.63)$$

$$sim(Jill, Joe) = \frac{2\times 5+5\times 2+0\times 5}{\sqrt{29}\sqrt{54}} = 0.50. \quad (9.64)$$

$$sim(Jill, Jane) = \frac{2\times 1+5\times 3+0\times 3}{\sqrt{29}\sqrt{19}} = 0.72. \quad (9.65)$$

$$sim(Jill, Jorge) = \frac{2\times 3+5\times 1+0\times 2}{\sqrt{29}\sqrt{14}} = 0.54. \quad (9.66)$$

Considering a neighborhood of size 2, the most similar users to Jill are John and Jane:

$$N(Jill) = \{John, Jane\}. \quad (9.67)$$

We also know that friends of Jill are

$$F(Jill) = \{Joe, Jane, Jorge\}. \quad (9.68)$$

We can use Equation 9.55 to predict the missing rating by taking the intersection of friends and neighbors:

$$\begin{aligned} r_{Jill,Mulan} &= \bar{r}_{Jill} + \frac{sim(Jill, Jane)(r_{Jane,Mulan} - \bar{r}_{Jane})}{sim(Jill, Jane)} \\ &= 2.33 + (4 - 2.75) = 3.58. \end{aligned} \quad (9.69)$$

Similarly, we can utilize Equation 9.56 to compute the missing rating. Here, we take Jill's two most similar neighbors: Jane and Jorge.

$$r_{Jill,Mulan} = \bar{r}_{Jill} + \frac{sim(Jill, Jane)(r_{Jane,Mulan} - \bar{r}_{Jane})}{sim(Jill, Jane) + sim(Jill, Jorge)}$$

$$+ \frac{sim(Jill, Jorge)(r_{Jorge,Mulan} - \bar{r}_{Jorge})}{sim(Jill, Jane) + sim(Jill, Jorge)}$$
$$= 2.33 + \frac{0.72(4 - 2.75) + 0.54(1 - 1.75)}{0.72 + 0.54} = 2.72 \tag{9.70}$$

9.4 Evaluating Recommendations

When a recommendation algorithm predicts ratings for items, one must evaluate how accurate its recommendations are. One can evaluate the (1) accuracy of predictions, (2) relevancy of recommendations, or (3) rankings of recommendations.

9.4.1 Evaluating Accuracy of Predictions

When evaluating the accuracy of predictions, we measure how close predicted ratings are to the true ratings. Similar to the evaluation of supervised learning, we often predict the ratings of some items with known ratings (i.e., true ratings) and compute how close the predictions are to the true ratings. One of the simplest methods, mean absolute error (MAE), computes the average absolute difference between the predicted ratings and true ratings,

$$MAE = \frac{\sum_{ij} |\hat{r}_{ij} - r_{ij}|}{n}, \tag{9.71}$$

where n is the number of predicted ratings, $\hat{r}_{ij}$ is the predicted rating, and r_{ij} is the true rating. Normalized mean absolute error (NMAE) normalizes MAE by dividing it by the range ratings can take,

$$NMAE = \frac{MAE}{r_{\max} - r_{\min}}, \tag{9.72}$$

where $r_{\max}$ is the maximum rating items can take and $r_{\min}$ is the minimum. In MAE, error linearly contributes to the MAE value. We can increase this contribution by considering the summation of squared errors in the root mean squared error (RMSE):

$$RMSE = \sqrt{\frac{1}{n} \sum_{i,j} (\hat{r}_{ij} - r_{ij})^2}. \tag{9.73}$$

Example 9.6. *Consider the following table with both the predicted ratings and true ratings of five items:*

Item	*Predicted Rating*	*True Rating*
1	*1*	*3*
2	*2*	*5*
3	*3*	*3*
4	*4*	*2*
5	*4*	*1*

The MAE, NMAE, and RMSE values are

$$MAE = \frac{|1-3|+|2-5|+|3-3|+|4-2|+|4-1|}{5} = 2. \tag{9.74}$$

$$NMAE = \frac{MAE}{5-1} = 0.5. \tag{9.75}$$

$$RMSE = \sqrt{\frac{(1-3)^2+(2-5)^2+(3-3)^2+(4-2)^2+(4-1)^2}{5}}. \tag{9.76}$$
$$= 2.28.$$

9.4.2 Evaluating Relevancy of Recommendations

When evaluating recommendations based on relevancy, we ask users if they find the recommended items relevant to their interests. Given a set of recommendations to a user, the user describes each recommendation as *relevant* or *irrelevant*. Based on the selection of items for recommendations and their relevancy, we can have the four types of items outlined in Table 9.5. Given this table, we can define measures that use relevancy

Table 9.5: Partitioning of Items with Respect to Their Selection for Recommendation and Their Relevancy

	Selected	Not Selected	Total
Relevant	N_{rs}	N_{rn}	N_r
Irrelevant	N_{is}	N_{in}	N_i
Total	N_s	N_n	N

information provided by users. *Precision* is one such measure. It defines the fraction of relevant items among recommended items:

$$P = \frac{N_{rs}}{N_s}. \tag{9.77}$$

Similarly, we can use *recall* to evaluate a recommender algorithm, which provides the probability of selecting a relevant item for recommendation:

$$R = \frac{N_{rs}}{N_r}. \tag{9.78}$$

We can also combine both precision and recall by taking their harmonic mean in the *F-measure*:

$$F = \frac{2PR}{P+R}. \tag{9.79}$$

Example 9.7. *Consider the following recommendation relevancy matrix for a set of 40 items. For this table, the precision, recall, and F-measure values are*

	Selected	*Not Selected*	*Total*
Relevant	*9*	*15*	*24*
Irrelevant	*3*	*13*	*16*
Total	*12*	*28*	*40*

$$P = \frac{9}{12} = 0.75. \tag{9.80}$$

$$R = \frac{9}{24} = 0.375. \tag{9.81}$$

$$F = \frac{2 \times 0.75 \times 0.375}{0.75 + 0.375} = 0.5. \tag{9.82}$$

9.4.3 Evaluating Ranking of Recommendations

Often, we predict ratings for multiple products for a user. Based on the predicted ratings, we can rank products based on their levels of interestingness to the user and then evaluate this ranking. Given the true ranking of interestingness of items, we can compare this ranking with it and report a value. Rank correlation measures the correlation between the predicted ranking and the true ranking. One such technique is the Spearman's rank

correlation discussed in Chapter 8. Let x_i, $1 \leq x_i \leq n$, denote the rank predicted for item i, $1 \leq i \leq n$. Similarly, let y_i, $1 \leq y_i \leq n$, denote the true rank of item i from the user's perspective. Spearman's rank correlation is defined as

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} (x_i - y_i)^2}{n^3 - n}, \tag{9.83}$$

where n is the total number of items.

Here, we discuss another rank correlation measure: Kendall's tau. We say that the pair of items (i, j) are *concordant* if their ranks $\{x_i, y_i\}$ and $\{x_j, y_j\}$ are in order:

Kendall's Tau

$$x_i > x_j, \quad y_i > y_j \quad \text{or} \quad x_i < x_j, \quad y_i < y_j. \tag{9.84}$$

A pair of items is *discordant* if their corresponding ranks are not in order:

$$x_i > x_j, \quad y_i < y_j \quad \text{or} \quad x_i < x_j, \quad y_i > y_j. \tag{9.85}$$

When $x_i = x_j$ or $y_i = y_j$, the pair is neither concordant nor discordant. Let c denote the total number of concordant item pairs and d the total number of discordant item pairs. Kendall's tau computes the difference between the two, normalized by the total number of item pairs $\binom{n}{2}$:

$$\tau = \frac{c - d}{\binom{n}{2}}. \tag{9.86}$$

Kendall's tau takes value in range $[-1, 1]$. When the ranks completely agree, all pairs are concordant and Kendall's tau takes value 1, and when the ranks completely disagree, all pairs are discordant and Kendall's tau takes value -1.

Example 9.8. *Consider a set of four items $I = \{i_1, i_2, i_3, i_4\}$ for which the predicted and true rankings are as follows:*

	Predicted Rank	*True Rank*
i_1	*1*	*1*
i_2	*2*	*4*
i_3	*3*	*2*
i_4	*4*	*3*

The pair of items and their status {concordant/discordant} are

$$(i_1, i_2) \quad : \quad \textit{concordant} \tag{9.87}$$
$$(i_1, i_3) \quad : \quad \textit{concordant} \tag{9.88}$$
$$(i_1, i_4) \quad : \quad \textit{concordant} \tag{9.89}$$
$$(i_2, i_3) \quad : \quad \textit{discordant} \tag{9.90}$$
$$(i_2, i_4) \quad : \quad \textit{discordant} \tag{9.91}$$
$$(i_3, i_4) \quad : \quad \textit{concordant} \tag{9.92}$$

Thus, Kendall's tau for the rankings is

$$\tau = \frac{4-2}{6} = 0.33. \tag{9.93}$$

9.5 Summary

In social media, recommendations are constantly being provided. Friend recommendation, product recommendation, and video recommendation, among others, are all examples of recommendations taking place in social media. Unlike web search, recommendation is tailored to individuals' interests and can help recommend more relevant items. Recommendation is challenging due to the cold-start problem, data sparsity, attacks on these systems, privacy concerns, and the need for an explanation for why items are being recommended.

In social media, sites often resort to classical recommendation algorithms to recommend items or products. These techniques can be divided into content-based methods and collaborative filtering techniques. In content-based methods, we use the similarity between the content (e.g., item description) of items and user profiles to recommend items. In collaborative filtering (CF), we use historical ratings of individuals in the form of a user-item matrix to recommend items. CF methods can be categorized into memory-based and model-based techniques. In memory-based techniques, we use the similarity between users (user-based) or items (item-based) to predict missing ratings. In model-based techniques, we assume that an underlying model describes how users rate items. Using matrix factorization techniques we approximate this model to predict missing ratings. Classical recommendation algorithms often predict ratings for individuals. We discussed ways to extend these techniques to groups of individuals.

In social media, we can also use friendship information to give recommendations. These friendships alone can help recommend (e.g., friend recommendation), can be added as complementary information to classical techniques, or can be used to constrain the recommendations provided by classical techniques.

Finally, we discussed the evaluation of recommendation techniques. Evaluation can be performed in terms of accuracy, relevancy, and rank of recommended items. We discussed MAE, NMAE, and RMSE as methods that evaluate accuracy, precision, recall, and F-measure from relevancy-based methods, and Kendall's tau from rank-based methods.

9.6 Bibliographic Notes

General references for the content provided in this chapter can be found in [138, 237, 249, 5]. In social media, recommendation is utilized for various items, including blogs [16], news [177, 63], videos [66], and tags [257]. For example, YouTube video recommendation system employs co-visitation counts to compute the similarity between videos (items). To perform recommendations, videos with high similarity to a seed set of videos are recommended to the user. The seed set consists of the videos that users watched on YouTube (beyond a certain threshold), as well as videos that are explicitly favorited, "liked," rated, or added to playlists.

Among classical techniques, more on content-based recommendation can be found in [226], and more on collaborative filtering can be found in [268, 246, 248]. Content-based and CF methods can be combined into *hybrid* methods, which are not discussed in this chapter. A survey of hybrid methods is available in [48]. More details on extending classical techniques to groups are provided in [137].

When making recommendations using social context, we can use additional information such as tags [116, 254] or trust [102, 222, 190, 180]. For instance, in [272], the authors discern multiple facets of trust and apply multifaceted trust in social recommendation. In another work, Tang et al. [273] exploit the evolution of both rating and trust relations for social recommendation. Users in the physical world are likely to ask for suggestions from their local friends while they also tend to seek suggestions from users with high global reputations (e.g., reviews by vine voice reviewers of Amazon.com). Therefore, in addition to friends, one can also use global network information for better recommendations. In [274], the authors exploit both local and global social relations for recommendation.

When recommending people (potential friends), we can use all these types of information. A comparison of different people recommendation techniques can be found in the work of Chen et al. [52]. Methods that extend classical techniques with social context are discussed in [181, 182, 152].

9.7 Exercises

Classical Recommendation Algorithm

1. Discuss one difference between content-based recommendation and collaborative filtering.

2. Compute the missing rating in this table using user-based collaborative filtering (CF). Use cosine similarity to find the nearest neighbors.

	God	*Le Cercle Rouge*	*Cidade de Deu*	*Rashomon*	*La vita e bella*	$\bar{r}_u$
Newton	3	0	3	3	2	
Einstein	5	4	0	2	3	
Gauss	1	2	4	2	0	
Aristotle	3	?	1	0	2	1.5
Euclid	2	2	0	1	5	

   Assuming that you have computed similarity values in the following table, calculate Aristotle's rating by completing these four tasks:

	Newton	Einstein	Gauss	Euclid
Aristotle	0.76	?	0.40	0.78

   - Calculate the similarity value between Aristotle and Einstein.
   - Identify Aristotle's two nearest neighbors.
   - Calculate $\bar{r}_u$ values for everyone (Aristotle's is given).
   - Calculate Aristotle's rating for *Le Cercle Rouge*.

3. In an item-based CF recommendation, describe how the recommender finds and recommends items to the given user.

Recommendation Using Social Context

4. Provide two examples where social context can help improve classical recommendation algorithms in social media.

5. In Equation 9.54, the term $\beta \sum_{i=1}^{n} \sum_{j \in F(i)} sim(i, j) \|U_i - U_j\|_F^2$ is added to model the similarity between friends' tastes. Let $T \in \mathbb{R}^{n \times n}$ denote the pairwise trust matrix, in which $0 \leq T_{ij} \leq 1$ denotes how much user i trusts user j. Using your intuition on how trustworthiness of individuals should affect recommendations received from them, modify Equation 9.54 using trust matrix T.

Evaluating Recommendation Algorithms

6. What does "high precision" mean? Why is precision alone insufficient to measure performance under normal circumstances? Provide an example to show that both precision and recall are important.

7. When is Kendall's tau equal to -1? In other words, how is the predicted ranking different from the true ranking?