Chapter 4
A Comprehensive Survey of Neighborhood-based Recommendation Methods

Christian Desrosiers and George Karypis

Abstract Among collaborative recommendation approaches, methods based on nearest-neighbors still enjoy a huge amount of popularity, due to their simplicity, their efficiency, and their ability to produce accurate and personalized recommendations. This chapter presents a comprehensive survey of neighborhood-based methods for the item recommendation problem. In particular, the main benefits of such methods, as well as their principal characteristics, are described. Furthermore, this document addresses the essential decisions that are required while implementing a neighborhood-based recommender system, and gives practical information on how to make such decisions. Finally, the problems of sparsity and limited coverage, often observed in large commercial recommender systems, are discussed, and a few solutions to overcome these problems are presented.

4.1 Introduction

The appearance and growth of online markets has had a considerable impact on the habits of consumers, providing them access to a greater variety of products and information on these goods. While this freedom of purchase has made online commerce into a multi-billion dollar industry, it also made it more difficult for consumers to select the products that best fit their needs. One of the main solutions proposed for this information overload problem are recommender systems, which provide automated and personalized suggestions of products to consumers. Recommender systems have been used in a wide variety of applications, such as the

Christian Desrosiers
Department of Software Engineering and IT, École de Technologie Supérieure, Montreal, Canada
e-mail: christian.desrosiers@etsmtl.ca

George Karypis
Department of Computer Science & Engineering, University of Minnesota, Minneapolis, USA
e-mail: karypis@cs.umn.edu

F. Ricci et al. (eds.), *Recommender Systems Handbook*,
DOI 10.1007/978-0-387-85820-3_4, © Springer Science+Business Media, LLC 2011

recommendation of books and CDs [47, 53], music [45, 70], movies [31, 51, 5], news [6, 41, 76], jokes [23], and web pages [7, 52].

The recommendation problem can be defined as estimating the response of a user for new items, based on historical information stored in the system, and suggesting to this user *novel* and *original* items for which the predicted response is *high*. The type of user-item responses varies from one application to the next, and falls in one of three categories: scalar, binary and unary. Scalar responses, also known as *ratings*, are numerical (e.g., 1-5 stars) or ordinal (e.g., strongly agree, agree, neutral, disagree, strongly disagree) values representing the possible levels of appreciation of users for items. Binary responses, on the other hand, only have two possible values encoding opposite levels of appreciation (e.g., like/dislike or interested/not interested). Finally, unary responses capture the interaction of a user with an item (e.g., purchase, online access, etc.) without giving explicit information on the appreciation of the user for this item. Since most users tend to interact with items that they find interesting, unary responses still provide useful information on the preferences of users.

The way in which user responses are obtained can also differ. For instance, in a movie recommendation application, users can enter ratings explicitly after watching a movie, giving their opinion on this movie. User responses can also be obtained implicitly from purchase history or access patterns [41, 76]. For example, the amount of time spent by a user browsing a specific type of item can be used as an indicator of the user's interest for this item type. For the purpose of simplicity, from this point on, we will call rating any type of user-item response.

4.1.1 Formal Definition of the Problem

In order to give a formal definition of the item recommendation task, we need to introduce some notation. Thus, the set of users in the system will be denoted by $\mathcal{U}$, and the set of items by $\mathcal{I}$. Moreover, we denote by $\mathcal{R}$ the set of ratings recorded in the system, and write $\mathcal{S}$ the set of possible values for a rating (e.g., $\mathcal{S} = [1,5]$ or $\mathcal{S} = \{\text{like}, \text{dislike}\}$). Also, we suppose that no more than one rating can be made by any user $u \in \mathcal{U}$ for a particular item $i \in \mathcal{I}$ and write r_{ui} this rating. To identify the subset of users that have rated an item i, we use the notation $\mathcal{U}_i$. Likewise, $\mathcal{I}_u$ represents the subset of items that have been rated by a user u. Finally, the items that have been rated by two users u and v, i.e. $\mathcal{I}_u \cap \mathcal{I}_v$, is an important concept in our presentation, and we use $\mathcal{I}_{uv}$ to denote this concept. In a similar fashion, $\mathcal{U}_{ij}$ is used to denote the set of users that have rated both items i and j.

Two of the most important problems associated with recommender systems are the *best item* and *top-N* recommendation problems. The first problem consists in finding, for a particular user u, the new item $i \in \mathcal{I} \setminus \mathcal{I}_u$ for which u is most likely to be interested in. When ratings are available, this task is most often defined as a regression or (multi-class) classification problem where the goal is to learn a function $f : \mathcal{U} \times \mathcal{I} \rightarrow \mathcal{S}$ that predicts the rating $f(u,i)$ of a user u for a new item i. This

function is then used to recommend to the active user u_a an item i^* for which the estimated rating has the highest value:

$$i^* = \underset{j \in \mathcal{I} \setminus \mathcal{I}_u}{\arg\max}\, f(u_a, j). \tag{4.1}$$

Accuracy is commonly used to evaluate the performance of the recommendation method. Typically, the ratings $\mathcal{R}$ are divided into a *training* set $\mathcal{R}_{\text{train}}$ used to learn f, and a *test* set $\mathcal{R}_{\text{test}}$ used to evaluate the prediction accuracy. Two popular measures of accuracy are the *Mean Absolute Error* (MAE):

$$\text{MAE}(f) = \frac{1}{|\mathcal{R}_{\text{test}}|} \sum_{r_{ui} \in \mathcal{R}_{\text{test}}} |f(u,i) - r_{ui}|, \tag{4.2}$$

and the *Root Mean Squared Error* (RMSE):

$$\text{RMSE}(f) = \sqrt{\frac{1}{|\mathcal{R}_{\text{test}}|} \sum_{r_{ui} \in \mathcal{R}_{\text{test}}} (f(u,i) - r_{ui})^2}. \tag{4.3}$$

When ratings are not available, for instance, if only the list of items purchased by each user is known, measuring the rating prediction accuracy is not possible. In such cases, the problem of finding the best item is usually transformed into the task of recommending to an active user u_a a list $L(u_a)$ containing N items likely to interest him or her [18, 45]. The quality of such method can be evaluated by splitting the items of $\mathcal{I}$ into a set $\mathcal{I}_{\text{train}}$, used to learn L, and a test set $\mathcal{I}_{\text{test}}$. Let $T(u) \subset \mathcal{I}_u \cap \mathcal{I}_{\text{test}}$ be the subset of test items that a user u found relevant. If the user responses are binary, these can be the items that u has rated positively. Otherwise, if only a list of purchased or accessed items is given for each user u, then these items can be used as $T(u)$. The performance of the method is then computed using the measures of *precision* and *recall*:

$$\text{Precision}(L) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} |L(u) \cap T(u)| \,/\, |L(u)| \tag{4.4}$$

$$\text{Recall}(L) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} |L(u) \cap T(u)| \,/\, |T(u)|. \tag{4.5}$$

A drawback of this task is that all items of a recommendation list $L(u)$ are considered equally interesting to user u. An alternative setting, described in [18], consists in learning a function L that maps each user u to a list $L(u)$ where items are *ordered* by their "interestingness" to u. If the test set is built by randomly selecting, for each user u, a single item i_u of $\mathcal{I}_u$, the performance of L can be evaluated with the *Average Reciprocal Hit-Rank* (ARHR):

$$\text{ARHR}(L) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{\text{rank}(i_u, L(u))}, \tag{4.6}$$

where $\text{rank}(i_u, L(u))$ is the rank of item i_u in $L(u)$, equal to ∞ if $i_u \notin L(u)$. A more extensive description of evaluation measures for recommender systems can be found in Chapter 8 of this book.

4.1.2 Overview of Recommendation Approaches

While the recommendation problem truly emerged as an independent area of research in the mid 1990's, it has deeper roots in several other fields like cognitive science [42] and information retrieval [65]. Approaches for this problem are normally divided in two broad categories: *content-based* and *collaborative filtering* approaches.

4.1.2.1 Content-based approaches

The general principle of content-based (or cognitive) approaches [7, 8, 44, 58] is to identify the common characteristics of items that have received a favorable rating from a user u, and then recommend to u new items that share these characteristics. In content-based recommender systems, rich information describing the nature of each item i is assumed to be available in the form of a feature vector $\mathbf{x}_i$. For items in the form of text documents, such as news articles [8, 44] or Web documents [7, 58], this vector often contains the *Term Frequency-Inverse Document Frequency* (TF-IDF) [65] weights of the most informative keywords. Moreover, for each user u, a preference profile vector $\mathbf{x}_u$ is usually obtained from the contents of items of $\mathcal{I}_u$. A technique to compute these profiles, used in several content-based recommender systems such as Newsweeder [44] and Fab [7], is the Rocchio algorithm [78, 12]. This technique updates the profile $\mathbf{x}_u$ of user u whenever this user rates an item i by adding the weights of $\mathbf{x}_i$ to $\mathbf{x}_u$, in proportion to the appreciation of u for i:

$$\mathbf{x}_u = \sum_{i \in \mathcal{I}_u} r_{ui} \mathbf{x}_i.$$

The user profiles can then be used to recommend new items to a user u, by suggesting the item whose feature vector $\mathbf{x}_i$ is most similar to the profile vector $\mathbf{x}_u$, for example, using the cosine similarity [7, 8, 44] or the *Minimum Description Length* (MDL) [44, 62]. This approach can also be used to predict the rating of user u for a new item i [44], by building for each rating value $r \in \mathcal{S}$ a content profile vector $\mathbf{x}_u^{(r)}$ as the average of the feature vectors of items that have received this rating value from u. The predicted rating $\hat{r}_{ui}$ for item i is the value r for which $\mathbf{x}_u^{(r)}$ is most similar to $\mathbf{x}_i$. Bayesian approaches using content information have also been proposed to predict ratings [8, 53, 58].

Recommender systems based purely on content generally suffer from the problems of *limited content analysis* and *over-specialization* [70]. Limited content anal-

ysis stems from the fact that the system may have only a limited amount of information on its users or the content of its items. The reasons for this lack of information can be numerous. For instance, privacy issues might refrain a user from providing personal information, or the precise content of items may be difficult or costly to obtain for some types of items, such as music or images. Finally, the content of an item is often insufficient to determine its quality. For example, it may be impossible to distinguish between a well written and a badly written article if both use the same terms. Over-specialization, on the other hand, is a side effect of the way in which content-based systems recommend new items, where the predicted rating of a user for an item is high if this item is similar to the ones liked by this user. For example, in a movie recommendation application, the system may recommend to a user a movie of the same genre or having the same actors as movies already seen by this user. Because of this, the system may fail to recommend items that are different but still interesting to the user. Solutions proposed for this problem include adding some randomness [71] or filtering out items that are too similar [8, 77]. More information on content-based recommendation approaches can be found in Chapter 3 of this book.

4.1.2.2 Collaborative filtering approaches

Unlike content-based approaches, which use the content of items previously rated by a user u, collaborative (or social) filtering approaches [18, 31, 41, 47, 60, 45, 70] rely on the ratings of u as well as those of other users in the system. The key idea is that the rating of u for a new item i is likely to be similar to that of another user v, if u and v have rated other items in a similar way. Likewise, u is likely to rate two items i and j in a similar fashion, if other users have given similar ratings to these two items.

Collaborative approaches overcome some of the limitations of content-based ones. For instance, items for which the content is not available or difficult to obtain can still be recommended to users through the feedback of other users. Furthermore, collaborative recommendations are based on the quality of items as evaluated by peers, instead of relying on content that may be a bad indicator of quality. Finally, unlike content-based systems, collaborative filtering ones can recommend items with very different content, as long as other users have already shown interest for these different items.

Following [1, 5, 10, 18], collaborative filtering methods can be grouped in the two general classes of *neighborhood* and *model*-based methods. In neighborhood-based (memory-based [10] or heuristic-based [1]) collaborative filtering [17, 18, 31, 41, 47, 54, 60, 45, 70], the user-item ratings stored in the system are directly used to predict ratings for new items. This can be done in two ways known as *user-based* or *item-based* recommendation. User-based systems, such as GroupLens [41], Bellcore video [31], and Ringo [70], evaluate the interest of a user u for an item i using the ratings for this item by other users, called *neighbors*, that have similar rating patterns. The neighbors of user u are typically the users v whose ratings on

the items rated by both u and v, i.e. $\mathcal{I}_{uv}$, are most correlated to those of u. Item-based approaches [18, 47, 45], on the other hand, predict the rating of a user u for an item i based on the ratings of u for items similar to i. In such approaches, two items are similar if several users of the system have rated these items in a similar fashion.

In contrast to neighborhood-based systems, which use the stored ratings directly in the prediction, model-based approaches use these ratings to learn a predictive model. The general idea is to model the user-item interactions with factors representing latent characteristics of the users and items in the system, like the preference class of users and the category class of items. This model is then trained using the available data, and later used to predict ratings of users for new items. Model-based approaches for the task of recommending items are numerous and include Bayesian Clustering [10], Latent Semantic Analysis [32], Latent Dirichlet Allocation [9], Maximum Entropy [78], Boltzmann Machines [64], Support Vector Machines [27], and Singular Value Decomposition [4, 42, 57, 74, 75]. A survey of state-of-the-art model-based methods can be found in Chapter 5 of this book.

4.1.3 Advantages of Neighborhood Approaches

While recent investigations show that state-of-the-art model-based approaches are superior to neighborhood ones in the task of predicting ratings [42, 73], there is also an emerging understanding that good prediction accuracy alone does not guarantee users an effective and satisfying experience [25]. Another factor that has been identified as playing an important role in the appreciation of users for the recommender system is *serendipity* [25, 45]. Serendipity extends the concept of novelty by helping a user find an interesting item he might not have otherwise discovered. For example, recommending to a user a movie directed by his favorite director constitutes a novel recommendation if the user was not aware of that movie, but is likely not serendipitous since the user would have discovered that movie on his own.

Model-based approaches excel at characterizing the preferences of a user with latent factors. For example, in a movie recommender system, such methods may determine that a given user is a fan of movies that are both funny and romantic, without having to actually define the notions "funny" and "romantic". This system would be able to recommend to the user a romantic comedy that may not have been known to this user. However, it may be difficult for this system to recommend a movie that does not quite fit this high-level genre, for instance, a funny parody of horror movies. Neighborhood approaches, on the other hand, capture local associations in the data. Consequently, it is possible for a movie recommender system based on this type of approach to recommend a movie very different from the users usual taste or a movie that is not well known (e.g. repertoire film), if one of his closest neighbors has given it a strong rating. This recommendation may not be a guaranteed success, as would be a romantic comedy, but it may help the user discover a whole new genre or a new favorite actor/director.

The main advantages of neighborhood-based methods are:

- **Simplicity:** Neighborhood-based methods are intuitive and relatively simple to implement. In their simplest form, only one parameter (the number of neighbors used in the prediction) requires tuning.
- **Justifiability:** Such methods also provide a concise and intuitive justification for the computed predictions. For example, in item-based recommendation, the list of neighbor items, as well as the ratings given by the user to these items, can be presented to the user as a justification for the recommendation. This can help the user better understand the recommendation and its relevance, and could serve as basis for an interactive system where users can select the neighbors for which a greater importance should be given in the recommendation [4]. The necessity of explaining recommendations to users is addressed in Chapter 15 of this book.
- **Efficiency:** One of the strong points of neighborhood-based systems is their efficiency. Unlike most model-based systems, they require no costly training phases, which need to be carried out at frequent intervals in large commercial applications. While the recommendation phase is usually more expensive than for model-based methods, the nearest-neighbors can be pre-computed in an offline step, providing near instantaneous recommendations. Moreover, storing these nearest neighbors requires very little memory, making such approaches scalable to applications having millions of users and items.
- **Stability:** Another useful property of recommender systems based on this approach is that they are little affected by the constant addition of users, items and ratings, which are typically observed in large commercial applications. For instance, once item similarities have been computed, an item-based system can readily make recommendations to new users, without having to re-train the system. Moreover, once a few ratings have been entered for a new item, only the similarities between this item and the ones already in the system need to be computed.

4.1.4 Objectives and Outline

This chapter has two main objectives. It first serves as a general guide on neighborhood-based recommender systems, and presents practical information on how to implement such recommendation approaches. In particular, the main components of neighborhood-based methods will be described, as well as the benefits of the most common choices for each of these components. Secondly, it presents more specialized techniques addressing particular aspects of recommending items, such as data sparsity. Although such techniques are not required to implement a simple neighborhood-based system, having a broader view of the various difficulties and solutions for neighborhood methods may help with making appropriate decisions during the implementation process.

The rest of this document is structured as follows. In Secton 4.2, the principal neighborhood approaches, predicting user ratings for new items based on regres-

sion or classification, are introduced, and the main advantages and flaws of these approaches are described. This section also presents two complementary ways of implementing such approaches, either based on user or item similarities, and analyses the impact of these two implementations on the accuracy, efficiency, stability, justfiability and serendipity of the recommender system. Section 4.3, on the other hand, focuses on the three main components of neighborhood-based recommendation methods: rating normalization, similarity weight computation, and neighborhood selection. For each of these components, the most common approaches are described, and their respective benefits compared. In Section 4.4, the problems of limited coverage and data sparsity are introduced, and several solutions are described to overcome these problems are described. In particular, several techniques based on dimensionality reduction and graphs are presented. Finally, the last section of this document summarizes the principal characteristics and methods of neighorhood-based recommendation, and gives a few more pointers on implementing such methods.

4.2 Neighborhood-based Recommendation

Recommender systems based on nearest-neighbors automate the common principle of *word-of-mouth*, where one relies on the opinion of like-minded people or other trusted sources to evaluate the value of an item (movie, book, articles, album, etc.) according to his own preferences. To illustrate this, consider the following example based on the ratings of Figure 4.1.

Example 4.1. User Eric has to decide whether or not to rent the movie "Titanic" that he has not yet seen. He knows that Lucy has very similar tastes when it comes to movies, as both of them hated "The Matrix" and loved "Forrest Gump", so he asks her opinion on this movie. On the other hand, Eric finds out he and Diane have different tastes, Diane likes action movies while he does not, and he discards her opinion or considers the opposite in his decision.

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
John	5	1		2	2
Lucy	1	5	2	5	5
Eric	2	?	3	5	4
Diane	4	3	5	3	

Fig. 4.1: A "toy example" showing the ratings of four users for five movies.

4.2.1 User-based Rating Prediction

User-based neighborhood recommendation methods predict the rating r_{ui} of a user u for a new item i using the ratings given to i by users most similar to u, called nearest-neighbors. Suppose we have for each user $v \neq u$ a value w_{uv} representing the preference similarity between u and v (how this similarity can be computed will be discussed in Section 4.3.2). The k-nearest-neighbors (k-NN) of u, denoted by $\mathcal{N}(u)$, are the k users v with the highest similarity w_{uv} to u. However, only the users who have rated item i can be used in the prediction of r_{ui}, and we instead consider the k users most similar to u that *have rated i*. We write this set of neighbors as $\mathcal{N}_i(u)$. The rating r_{ui} can be estimated as the average rating given to i by these neighbors:

$$\hat{r}_{ui} = \frac{1}{|\mathcal{N}_i(u)|} \sum_{v \in \mathcal{N}_i(u)} r_{vi}. \tag{4.7}$$

A problem with (4.7) is that is does not take into account the fact that the neighbors can have different levels of similarity. Consider once more the example of Figure 4.1. If the two nearest-neighbors of Eric are Lucy and Diane, it would be foolish to consider equally their ratings of the movie "Titanic", since Lucy's tastes are much closer to Eric's than Diane's. A common solution to this problem is to weigh the contribution of each neighbor by its similarity to u. However, if these weights do not sum to 1, the predicted ratings can be well outside the range of allowed values. Consequently, it is customary to normalize these weights, such that the predicted rating becomes

$$\hat{r}_{ui} = \frac{\sum_{v \in \mathcal{N}_i(u)} w_{uv}\, r_{vi}}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|}. \tag{4.8}$$

In the denominator of (4.8), $|w_{uv}|$ is used instead of w_{uv} because negative weights can produce ratings outside the allowed range. Also, w_{uv} can be replaced by w_{uv}^{α}, where $\alpha > 0$ is an amplification factor [10]. When $\alpha > 1$, as is it most often employed, an even greater importance is given to the neighbors that are the closest to u.

Example 4.2. Suppose we want to use (4.8) to predict Eric's rating of the movie "Titanic" using the ratings of Lucy and Diane for this movie. Moreover, suppose the similarity weights between these neighbors and Eric are respectively 0.75 and 0.15. The predicted rating would be

$$\hat{r} = \frac{0.75 \times 5 + 0.15 \times 3}{0.75 + 0.15} \simeq 4.67,$$

which is closer to Lucy's rating than to Diane's.

Equation (4.8) also has an important flaw: it does not consider the fact that users may use different rating values to quantify the same level of appreciation for an item. For example, one user may give the highest rating value to only a few outstanding

items, while a less difficult one may give this value to most of the items he likes. This problem is usually addressed by converting the neighbors' ratings r_{vi} to normalized ones $h(r_{vi})$ [10, 60], giving the following prediction:

$$\hat{r}_{ui} = h^{-1}\left(\frac{\sum_{v \in \mathcal{N}_i(u)} w_{uv}\, h(r_{vi})}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|}\right). \tag{4.9}$$

Note that the predicted rating must be converted back to the original scale, hence the h^{-1} in the equation. The most common approaches to normalize ratings will be presented in Section 4.3.1.

4.2.2 User-based Classification

The prediction approach just described, where the predicted ratings are computed as a weighted average of the neighbors' ratings, essentially solves a *regression* problem. Neighborhood-based *classification*, on the other hand, finds the most likely rating given by a user u to an item i, by having the nearest-neighbors of u vote on this value. The vote v_{ir} given by the k-NN of u for the rating $r \in \mathcal{S}$ can be obtained as the sum of the similarity weights of neighbors that have given this rating to i:

$$v_{ir} = \sum_{v \in \mathcal{N}_i(u)} \delta(r_{vi} = r)\, w_{uv}, \tag{4.10}$$

where $\delta(r_{vi} = r)$ is 1 if $r_{vi} = r$, and 0 otherwise. Once this has been computed for every possible rating value, the predicted rating is simply the value r for which v_{ir} is the greatest.

Example 4.3. Suppose once again that the two nearest-neighbors of Eric are Lucy and Diane with respective similarity weights 0.75 and 0.15. In this case, ratings 5 and 3 each have one vote. However, since Lucy's vote has a greater weight than Diane's, the predicted rating will be $\hat{r} = 5$.

A classification method that considers normalized ratings can also be defined. Let $\mathcal{S}'$ be the set of possible normalized values (that may require discretization), the predicted rating is obtained as:

$$\hat{r}_{ui} = h^{-1}\left(\arg\max_{r \in \mathcal{S}'} \sum_{v \in \mathcal{N}_i(u)} \delta(h(r_{vi}) = r)\, w_{uv}\right). \tag{4.11}$$

4.2.3 Regression VS Classification

The choice between implementing a neighborhood-based regression or classification method largely depends on the system's rating scale. Thus, if the rating scale is continuous, e.g. ratings in the *Jester* joke recommender system [23] can take any value between -10 and 10, then a regression method is more appropriate. On the contrary, if the rating scale has only a few discrete values, e.g. "good" or "bad", or if the values cannot be ordered in an obvious fashion, then a classification method might be preferable. Furthermore, since normalization tends to map ratings to a continuous scale, it may be harder to handle in a classification approach.

Another way to compare these two approaches is by considering the situation where all neighbors have the same similarity weight. As the number of neighbors used in the prediction increases, the rating r_{ui} predicted by the regression approach will tend toward the mean rating of item i. Suppose item i has only ratings at either end of the rating range, i.e. it is either loved or hated, then the regression approach will make the safe decision that the item's worth is average. This is also justified from a statistical point of view since the expected rating (estimated in this case) is the one that minimizes the RMSE. On the other hand, the classification approach will predict the rating as the most frequent one given to i. This is more risky as the item will be labeled as either "good" or "bad". However, as mentioned before, taking risk may be be desirable if it leads to serendipitous recommendations.

4.2.4 Item-based Recommendation

While user-based methods rely on the opinion of like-minded users to predict a rating, item-based approaches [18, 47, 45] look at ratings given to similar items. Let us illustrate this approach with our toy example.

Example 4.4. Instead of consulting with his peers, Eric instead determines whether the movie "Titanic" is right for him by considering the movies that he has already seen. He notices that people that have rated this movie have given similar ratings to the movies "Forrest Gump" and "Wall-E". Since Eric liked these two movies he concludes that he will also like the movie "Titanic".

This idea can be formalized as follows. Denote by $\mathcal{N}_u(i)$ the items rated by user u most similar to item i. The predicted rating of u for i is obtained as a weighted average of the ratings given by u to the items of $\mathcal{N}_u(i)$:

$$\hat{r}_{ui} = \frac{\sum\limits_{j \in \mathcal{N}_u(i)} w_{ij}\, r_{uj}}{\sum\limits_{j \in \mathcal{N}_u(i)} |w_{ij}|}. \tag{4.12}$$

Example 4.5. Suppose our prediction is again made using two nearest-neighbors, and that the items most similar to "Titanic" are "Forrest Gump" and "Wall-E", with

respective similarity weights 0.85 and 0.75. Since ratings of 5 and 4 were given by Eric to these two movies, the predicted rating is computed as

$$\hat{r} = \frac{0.85 \times 5 + 0.75 \times 4}{0.85 + 0.75} \simeq 4.53.$$

Again, the differences in the users' individual rating scales can be considered by normalizing ratings with a h:

$$\hat{r}_{ui} = h^{-1}\left(\frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij}\, h(r_{uj})}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}\right). \tag{4.13}$$

Moreover, we can also define an item-based classification approach. In this case, the items j rated by user u vote for the rating to be given to a new item i, and these votes are weighted by the similarity between i and j. The normalized version of this approach can be expressed as follows:

$$\hat{r}_{ui} = h^{-1}\left(\arg\max_{r \in \mathcal{S}'} \sum_{j \in \mathcal{N}_u(i)} \delta(h(r_{uj}) = r)\, w_{ij}\right). \tag{4.14}$$

4.2.5 User-based VS Item-based Recommendation

When choosing between the implementation of a user-based and an item-based neighborhood recommender system, five criteria should be considered:

- **Accuracy:** The accuracy of neighborhood recommendation methods depends mostly on the ratio between the number of users and items in the system. As will be presented in the Section 4.3.2, the similarity between two users in user-based methods, which determines the neighbors of a user, is normally obtained by comparing the ratings made by these users on the same items. Consider a system that has $10,000$ ratings made by $1,000$ users on 100 items, and suppose, for the purpose of this analysis, that the ratings are distributed uniformly over the items[1]. Following Table 4.1, the average number of users available as potential neighbors is roughly 650. However, the average number of common ratings used to compute the similarities is only 1. On the other hand, an item-based method usually computes the similarity between two items by comparing ratings made by the same user on these items. Assuming once more a uniform distribution of ratings, we find an average number of potential neighbors of 99 and an average number of ratings used to compute the similarities of 10.

[1] The distribution of ratings in real-life data is normally skewed, i.e. most ratings are given to a small proportion of items.

In general, a small number of high-confidence neighbors is by far preferable to a large number of neighbors for which the similarity weights are not trustworthy. In cases where the number of users is much greater than the number of items, such as large commercial systems like *Amazon.com*, item-based methods can therefore produce more accurate recommendations [19, 45]. Likewise, systems that have fewer users than items, e.g., a research paper recommender with thousands of users but hundreds of thousands of articles to recommend, may benefit more from user-based neighborhood methods [25].

Table 4.1: The average number of neighbors and average number of ratings used in the computation of similarities for user-based and item-based neighborhood methods. A uniform distribution of ratings is assumed with average number of ratings per user $p = |\mathcal{R}|/|\mathcal{U}|$, and average number of ratings per item $q = |\mathcal{R}|/|\mathcal{I}|$

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

- **Efficiency:** As shown in Table 4.2, the memory and computational efficiency of recommender systems also depends on the ratio between the number of users and items. Thus, when the number of users exceeds the number of items, as is it most often the case, item-based recommendation approaches require much less memory and time to compute the similarity weights (training phase) than user-based ones, making them more scalable. However, the time complexity of the online recommendation phase, which depends only on the number of available items and the maximum number of neighbors, is the same for user-based and item-based methods.

  In practice, computing the similarity weights is much less expensive than the worst-case complexity reported in Table 4.2, due to the fact that users rate only a few of the available items. Accordingly, only the non-zero similarity weights need to be stored, which is often much less than the number of user pairs. This number can be further reduced by storing for each user only the top N weights, where N is a parameter [45]. In the same manner, the non-zero weights can be computed efficiently without having to test each pair of users or items, which makes neighborhood methods scalable to very large systems.

- **Stability:** The choice between a user-based and an item-based approach also depends on the frequency and amount of change in the users and items of the system. If the list of available items is fairly static in comparison to the users of the system, an item-based method may be preferable since the item similarity weights could then be computed at infrequent time intervals while still being able to recommend items to new users. On the contrary, in applications where the list

Table 4.2: The space and time complexity of user-based and item-based neighborhood methods, as a function of the maximum number of ratings per user $p = \max_u |\mathcal{I}_u|$, the maximum number of ratings per item $q = \max_i |\mathcal{U}_i|$, and the maximum number of neighbors used in the rating predictions k.

	Space	Time	
		Training	Online
User-based	$O(\|\mathcal{U}\|^2)$	$O(\|\mathcal{U}\|^2 p)$	$O(\|\mathcal{I}\|k)$
Item-based	$O(\|\mathcal{I}\|^2)$	$O(\|\mathcal{I}\|^2 q)$	$O(\|\mathcal{I}\|k)$

of available items is constantly changing, e.g., an online article recommender, user-based methods could prove to be more stable.

- **Justifiability:** An advantage of item-based methods is that they can easily be used to justify a recommendation. Hence, the list of neighbor items used in the prediction, as well as their similarity weights, can be presented to the user as an explanation of the recommendation. By modifying the list of neighbors and/or their weights, it then becomes possible for the user to participate interactively in the recommendation process. User-based methods, however, are less amenable to this process because the active user does not know the other users serving as neighbors in the recommendation.

- **Serendipity:** In item-based methods, the rating predicted for an item is based on the ratings given to similar items. Consequently, recommender systems using this approach will tend to recommend to a user items that are related to those usually appreciated by this user. For instance, in a movie recommendation application, movies having the same genre, actors or director as those highly rated by the user are likely to be recommended. While this may lead to safe recommendations, it does less to help the user discover different types of items that he might like as much.

  Because they work with user similarity, on the other hand, user-based approaches are more likely to make serendipitous recommendations. This is particularly true if the recommendation is made with a small number of nearest-neighbors. For example, a user A that has watched only comedies may be very similar to a user B only by the ratings made on such movies. However, if B is fond of a movie in a different genre, this movie may be recommended to A through his similarity with B.

4.3 Components of Neighborhood Methods

In the previous section, we have seen that deciding between a regression and a classification rating prediction method, as well as choosing between a user-based or item-based recommendation approach, can have a significant impact on the accu-

racy, efficiency and overall quality of the recommender system. In addition to these crucial attributes, three very important considerations in the implementation of a neighborhood-based recommender system are 1) the normalization of ratings, 2) the computation of the similarity weights, and 3) the selection of neighbors. This section reviews some of the most common approaches for these three components, describes the main advantages and disadvantages of using each one of them, and gives indications on how to implement them.

4.3.1 Rating Normalization

When it comes to assigning a rating to an item, each user has its own personal scale. Even if an explicit definition of each of the possible ratings is supplied (e.g., 1="strongly disagree", 2="disagree", 3="neutral", etc.), some users might be reluctant to give high/low scores to items they liked/disliked. Two of the most popular rating normalization schemes that have been proposed to convert individual ratings to a more universal scale are *mean-centering* and *Z-score*.

4.3.1.1 Mean-centering

The idea of mean-centering [10, 60] is to determine whether a rating is positive or negative by comparing it to the mean rating. In user-based recommendation, a raw rating r_{ui} is transformation to a mean-centered one $h(r_{ui})$ by subtracting to r_{ui} the average $\overline{r}_u$ of the ratings given by user u to the items in $\mathcal{I}_u$:

$$h(r_{ui}) = r_{ui} - \overline{r}_u.$$

Using this approach the user-based prediction of a rating r_{ui} is obtained as

$$\hat{r}_{ui} = \overline{r}_u + \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv}\left(r_{vi} - \overline{r}_v\right)}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|}. \tag{4.15}$$

In the same way, the *item*-mean-centered normalization of r_{ui} is given by

$$h(r_{ui}) = r_{ui} - \overline{r}_i,$$

where $\overline{r}_i$ corresponds to the mean rating given to item i by user in $\mathcal{U}_i$. This normalization technique is most often used in item-based recommendation, where a rating r_{ui} is predicted as:

$$\hat{r}_{ui} = \overline{r}_i + \frac{\sum\limits_{j \in \mathcal{N}_u(i)} w_{ij}\left(r_{uj} - \overline{r}_j\right)}{\sum\limits_{j \in \mathcal{N}_u(i)} |w_{ij}|}. \tag{4.16}$$

An interesting property of mean-centering is that one can see right-away if the appreciation of a user for an item is positive or negative by looking at the sign of the normalized rating. Moreover, the module of this rating gives the level at which the user likes or dislikes the item.

Example 4.6. As shown in Figure 4.2, although Diane gave an average rating of 3 to the movies "Titanic" and "Forrest Gump", the user-mean-centered ratings show that her appreciation of these movies is in fact negative. This is because her ratings are high on average, and so, an average rating correspond to a low degree of appreciation. Differences are also visible while comparing the two types of mean-centering. For instance, the item-mean-centered rating of the movie "Titanic" is neutral, instead of negative, due to the fact that much lower ratings were given to that movie. Likewise, Diane's appreciation for "The Matrix" and John's distaste for "Forrest Gump" are more pronounced in the item-mean-centered ratings.

User mean-centering:

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
John	2.50	-1.50		-0.50	-0.50
Lucy	-2.60	1.40	-1.60	1.40	1.40
Eric	-1.50		-0.50	1.50	0.50
Diane	0.25	-0.75	1.25	-0.75	

Item mean-centering:

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
John	2.00	-2.00		-1.75	-1.67
Lucy	-2.00	2.00	-1.33	1.25	1.33
Eric	-1.00		-0.33	1.25	0.33
Diane	1.00	0.00	1.67	-0.75	

Fig. 4.2: The *user* and *item* mean-centered ratings of Figure 4.1.

4.3.1.2 Z-score normalization

Consider, two users A and B that both have an average rating of 3. Moreover, suppose that the ratings of A alternate between 1 and 5, while those of B are always 3. A rating of 5 given to an item by B is more exceptional than the same rating given by A, and, thus, reflects a greater appreciation for this item. While mean-centering removes the offsets caused by the different perceptions of an average rating, Z-score normalization [29] also considers the spread in the individual rating scales.

Once again, this is usually done differently in user-based than in item-based recommendation. In user-based methods, the normalization of a rating r_{ui} divides the *user*-mean-centered rating by the standard deviation σ_u of the ratings given by user u:

$$h(r_{ui}) = \frac{r_{ui} - \overline{r}_u}{\sigma_u}.$$

A user-based prediction of rating r_{ui} using this normalization approach would therefore be obtained as

$$\hat{r}_{ui} = \overline{r}_u + \sigma_u \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv} \left(r_{vi} - \overline{r}_v\right)/\sigma_v}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|}. \tag{4.17}$$

Likewise, the z-score normalization of r_{ui} in item-based methods divides the *item*-mean-centered rating by the standard deviation of ratings given to item i:

$$h(r_{ui}) = \frac{r_{ui} - \overline{r}_i}{\sigma_i}.$$

The item-based prediction of rating r_{ui} would then be

$$\hat{r}_{ui} = \overline{r}_i + \sigma_i \frac{\sum\limits_{j \in \mathcal{N}_u(i)} w_{ij} \left(r_{uj} - \overline{r}_j\right)/\sigma_j}{\sum\limits_{j \in \mathcal{N}_u(i)} |w_{ij}|}. \tag{4.18}$$

4.3.1.3 Choosing a normalization scheme

In some cases, rating normalization can have undesirable effects. For instance, imagine the case of a user that gave only the highest ratings to the items he has purchased. Mean-centering would consider this user as "easy to please" and any rating below this highest rating (whether it is a positive or negative rating) would be considered as negative. However, it is possible that this user is in fact "hard to please" and carefully selects only items that he will like for sure. Furthermore, normalizing on a few ratings can produce unexpected results. For example, if a user has entered a single rating or a few identical ratings, his rating standard deviation will be 0, leading to undefined prediction values. Nevertheless, if the rating data is not overly sparse, normalizing ratings has been found to consistently improve the predictions [29, 33].

Comparing mean-centering with Z-score, as mentioned, the second one has the additional benefit of considering the variance in the ratings of individual users or items. This is particularly useful if the rating scale has a wide range of discrete values or if it is continuous. On the other hand, because the ratings are divided and multiplied by possibly very different standard deviation values, Z-score can be more sensitive than mean-centering and, more often, predict ratings that are outside the rating scale. Lastly, while an initial investigation found mean-centering and Z-score

to give comparable results [29], a more recent one showed Z-score to have more significant benefits [33].

Finally, if rating normalization is not possible or does not improve the results, another possible approach to remove the problems caused by the individual rating scale is *preference-based filtering*. The particularity of this approach is that it focuses on predicting the relative preferences of users instead of absolute rating values. Since the rating scale does not change the preference order for items, predicting relative preferences removes the need to normalize the ratings. More information on this approach can be found in [13, 21, 37, 36].

4.3.2 Similarity Weight Computation

The similarity weights play a double role in neighborhood-based recommendation methods: 1) they allow the selection of trusted neighbors whose ratings are used in the prediction, and 2) they provide the means to give more or less importance to these neighbors in the prediction. The computation of the similarity weights is one of the most critical aspects of building a neighborhood-based recommender system, as it can have a significant impact on both its accuracy and its performance.

4.3.2.1 Correlation-based similarity

A measure of the similarity between two objects a and b, often used in information retrieval, consists in representing these objects in the form of two vectors $\mathbf{x}_a$ and $\mathbf{x}_b$ and computing the *Cosine Vector* (CV) (or *Vector Space*) similarity [7, 8, 44] between these vectors:

$$\cos(\mathbf{x}_a, \mathbf{x}_b) \;=\; \frac{\mathbf{x}_a^\top \mathbf{x}_b}{||\mathbf{x}_a||||\mathbf{x}_b||}.$$

In the context of item recommendation, this measure can be employed to compute user similarities by considering a user u as a vector $\mathbf{x}_u \in \mathbb{R}^{|I|}$, where $\mathbf{x}_{ui} = r_{ui}$ if user u has rated item i, and 0 otherwise. The similarity between two users u and v would then be computed as

$$CV(u,v) \;=\; \cos(\mathbf{x}_u, \mathbf{x}_v) \;=\; \frac{\sum\limits_{i \in \mathcal{I}_{uv}} r_{ui}\, r_{vi}}{\sqrt{\sum\limits_{i \in \mathcal{I}_u} r_{ui}^2 \sum\limits_{j \in \mathcal{I}_v} r_{vj}^2}}, \tag{4.19}$$

where I_{uv} once more denotes the items rated by both u and v. A problem with this measure is that is does not consider the differences in the mean and variance of the ratings made by users u and v.

A popular measure that compares ratings where the effects of mean and variance have been removed is the *Pearson Correlation* (PC) similarity:

$$\mathrm{PC}(u,v) = \frac{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)^2 \sum_{i \in \mathcal{I}_{uv}} (r_{vi} - \bar{r}_v)^2}}. \tag{4.20}$$

Note that this is different from computing the CV similarity on the Z-score normalized ratings, since the standard deviation of the ratings is evaluated only on the common items I_{uv}, not on the entire set of items rated by u and v, i.e. $\mathcal{I}_u$ and $\mathcal{I}_v$. The same idea can be used to obtain similarities between two items i and j [18, 45], this time by comparing the ratings made by users that have rated both of these items:

$$\mathrm{PC}(i,j) = \frac{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)^2 \sum_{u \in \mathcal{U}_{ij}} (r_{uj} - \bar{r}_j)^2}}. \tag{4.21}$$

While the sign of a similarity weight indicates whether the correlation is direct or inverse, its magnitude (ranging from 0 to 1) represents the strength of the correlation.

Example 4.7. The similarities between the pairs of users and items of our toy example, as computed using PC similarity, are shown in Figure 4.3. We can see that Lucy's taste in movies is very close to Eric's (similarity of 0.922) but very different from John's (similarity of -0.938). This means that Eric's ratings can be trusted to predict Lucy's, and that Lucy should discard John's opinion on movies or consider the opposite. We also find that the people that like "The Matrix" also like "Die Hard" but hate "Wall-E". Note that these relations were discovered without having any knowledge of the genre, director or actors of these movies.

The differences in the rating scales of individual users are often more pronounced than the differences in ratings given to individual items. Therefore, while computing the item similarities, it may be more appropriate to compare ratings that are centered on their *user* mean, instead of their *item* mean. The *Adjusted Cosine* (AC) similarity [45], is a modification of the PC item similarity which compares user-mean-centered ratings:

$$AC(i,j) = \frac{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_u)(r_{uj} - \bar{r}_u)}{\sqrt{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_u)^2 \sum_{u \in \mathcal{U}_{ij}} (r_{uj} - \bar{r}_u)^2}}.$$

In some cases, AC similarity has been found to outperform PC similarity on the prediction of ratings using an item-based method [45].

4.3.2.2 Other similarity measures

Several other measures have been proposed to compute similarities between users or items. One of them is the *Mean Squared Difference* (MSD) [70], which evaluates the

User-based Pearson correlation

	John	Lucy	Eric	Diane
John	1.000	-0.938	-0.839	0.659
Lucy	-0.938	1.000	0.922	-0.787
Eric	-0.839	0.922	1.000	-0.659
Diane	0.659	-0.787	-0.659	1.000

Item-based Pearson correlation

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
Matrix	1.000	-0.943	0.882	-0.974	-0.977
Titanic	-0.943	1.000	-0.625	0.931	0.994
Die Hard	0.882	-0.625	1.000	-0.804	-1.000
Forrest Gump	-0.974	0.931	-0.804	1.000	0.930
Wall-E	-0.977	0.994	-1.000	0.930	1.000

Fig. 4.3: The *user* and *item* PC similarity for the ratings of Figure 4.1.

similarity between two users u and v as the inverse of the average squared difference between the ratings given by u and v on the same items:

$$\mathrm{MSD}(u,v) \;=\; \frac{|\mathcal{I}_{uv}|}{\sum\limits_{i\in\mathcal{I}_{uv}} (r_{ui}-r_{vi})^2}. \tag{4.22}$$

While it could be modified to compute the differences on normalized ratings, the MSD similarity is limited compared to PC similarity because it does not capture negative correlations between user preferences or the appreciation of different items. Having such negative correlations may improve the rating prediction accuracy [28].

Another well-known similarity measure is the *Spearman Rank Correlation* (SRC) [39]. While PC uses the rating values directly, SRC instead considers the ranking of these ratings. Denote by k_{ui} the rating rank of item i in user u's list of rated items (tied ratings get the average rank of their spot). The SRC similarity between two users u and v is evaluated as:

$$\mathrm{SRC}(u,v) \;=\; \frac{\sum\limits_{i\in\mathcal{I}_{uv}} (k_{ui}-\overline{k}_u)(k_{vi}-\overline{k}_v)}{\sqrt{\sum\limits_{i\in\mathcal{I}_{uv}} (k_{ui}-\overline{k}_u)^2 \sum\limits_{i\in\mathcal{I}_{uv}} (k_{vi}-\overline{k}_v)^2}}, \tag{4.23}$$

where $\overline{k}_u$ is the average rank of items rated by u (which can differ from $|\mathcal{I}_u|+1$ if there are tied ratings).

The principal advantage of SRC is that it avoids the problem of rating normalization, described in the last section, by using rankings. On the other hand, this measure may not be the best one when the rating range has only a few possible values, since

that would create a large number of tied ratings. Moreover, this measure is typically more expensive than PC as ratings need to be sorted in order to compute their rank.

Table 4.3 shows the user-based prediction accuracy (MAE) obtained with MSD, SRC and PC similarity measures, on the *MovieLens*[2] dataset [28]. Results are given for different values of k, which represents the maximum number of neighbors used in the predictions. For this data, we notice that MSD leads to the least accurate predictions, possibly due to the fact that it does not take into account negative correlations. Also, these results show PC to be slightly more accurate than SRC. Finally, although PC has been generally recognized as the best similarity measure, see e.g. [28], a more recent investigation has shown that the performance of such measures depended greatly on the data [33].

Table 4.3: The rating prediction accuracy (MAE) obtained using the Mean Squared Difference (MSD), the Spearman Rank Correlation and the Pearson Correaltion (PC) similarity. Results are shown for predictions using an increasing number of neighbors k.

k	MSD	SRC	PC
5	0.7898	0.7855	0.7829
10	0.7718	0.7636	0.7618
20	0.7634	0.7558	0.7545
60	0.7602	0.7529	0.7518
80	0.7605	0.7531	0.7523
100	0.7610	0.7533	0.7528

4.3.2.3 Accounting for significance

Because the rating data is frequently sparse in comparison to the number of users and items of a system, similarity weights are often computed using only a few ratings given to common items or made by the same users. For example, if the system has $10,000$ ratings made by $1,000$ users on 100 items (assuming a uniform distribution of ratings), Table 4.1 shows us that the similarity between two users is computed, on average, by comparing the ratings given by these users to a *single* item. If these few ratings are equal, then the users will be considered as "fully similar" and will likely play an important role in each other's recommendations. However, if the users' preferences are in fact different, this may lead to poor recommendations.

Several strategies have been proposed to take into account the *significance* of a similarity weight. The principle of these strategies is essentially the same: reduce the magnitude of a similarity weight when this weight is computed using only a few ratings. For instance, in *Significance Weighting* [29, 49], a user similarity weight

[2] `http://www.grouplens.org/`

w_{uv} is penalized by a factor proportional to the number of commonly rated items, if this number is less than a given parameter $\gamma > 0$:

$$w'_{uv} = \frac{\min\{|\mathcal{I}_{uv}|, \gamma\}}{\gamma} \times w_{uv}. \tag{4.24}$$

Likewise, an item similarity w_{ij}, obtained from a few ratings, can be adjusted as

$$w'_{ij} = \frac{\min\{|\mathcal{U}_{ij}|, \gamma\}}{\gamma} \times w_{ij}. \tag{4.25}$$

In [29, 28], it was found that using $\gamma \geq 25$ could significantly improve the accuracy of the predicted ratings, and that a value of 50 for γ gave the best results. However, the optimal value for this parameter is data dependent and should be determined using a cross-validation approach.

A characteristic of significance weighting is its use of a threshold γ determining when a weight should be adjusted. A more continuous approach, described in [4], is based on the concept of *shrinkage* where a weak or biased estimator can be improved if it is "shrunk" toward a null-value. This approach can be justified using a Bayesian perspective, where the best estimator of a parameter is the posterior mean, corresponding to a linear combination of the prior mean of the parameter (null-value) and an empirical estimator based fully on the data. In this case, the parameters to estimate are the similarity weights and the null value is zero. Thus, a user similarity w_{uv} estimated on a few ratings is shrunk as

$$w'_{uv} = \frac{|\mathcal{I}_{uv}|}{|\mathcal{I}_{uv}| + \beta} \times w_{uv}, \tag{4.26}$$

where $\beta > 0$ is a parameter whose value should also be selected using cross-validation. In this approach, w_{uv} is shrunk proportionally to $\beta/|I_{uv}|$, such that almost no adjustment is made when $|\mathcal{I}_{uv}| \gg \beta$. Item similarities can be shrunk in the same way:

$$w'_{ij} = \frac{|\mathcal{U}_{ij}|}{|\mathcal{U}_{ij}| + \beta} \times w_{ij}, \tag{4.27}$$

As reported in [4], a typical value for β is 100.

4.3.2.4 Accounting for variance

Ratings made by two users on universally liked/disliked items may not be as informative as those made for items with a greater rating variance. For instance, most people like classic movies such as "The Godfather", so basing the weight computation on such movies would produce artificially high values. Likewise, a user that always rates items in the same way may provide less predictive information than one whose preferences vary from one item to another.

A recommendation approach that addresses this problem is the *Inverse User Frequency* [10]. Based on the information retrieval notion of *Inverse Document Frequency* (IDF), a weight λ_i is given to each item i, in proportion to the log-ratio of users that have rated i:

$$\lambda_i = \log \frac{|\mathcal{U}|}{|\mathcal{U}_i|}.$$

While computing the *Frequency-Weighted Pearson Correlation (FWPC)* between users u and v, the correlation between the ratings given to an item i is weighted by λ_i:

$$\mathrm{FWPC}(u,v) = \frac{\sum\limits_{i \in \mathcal{I}_{uv}} \lambda_i (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum\limits_{i \in \mathcal{I}_{uv}} \lambda_i (r_{ui} - \bar{r}_u)^2 \sum\limits_{i \in \mathcal{I}_{uv}} \lambda_i (r_{vi} - \bar{r}_v)^2}}. \tag{4.28}$$

This approach, which was found to improve the prediction accuracy of a user-based recommendation method [10], could also be adapted to the computation of item similarities.

More advanced strategies have also been proposed to consider rating variance. One of these strategies, described in [35], computes the factors λ_i by maximizing the average similarity between users. In this approach, the similarity between two users u and v, given an item weight vector $\lambda = (\lambda_1, \ldots, \lambda_{|\mathcal{I}|})$, is evaluated as the likelihood of u to have the same rating behavior as user v:

$$\Pr(u|v,\lambda) = \frac{1}{Z_v} \exp\left(\sum_{i \in \mathcal{I}_{uv}} \lambda_i \, r_{ui} \, r_{vi} \right),$$

where Z_v is a normalization constant. The optimal item weight vector is the one maximizing the average similarity between users.

4.3.3 Neighborhood Selection

The number of nearest-neighbors to select and the criteria used for this selection can also have a serious impact on the quality of the recommender system. The selection of the neighbors used in the recommendation of items is normally done in two steps: 1) a global filtering step where only the most likely candidates are kept, and 2) a per prediction step which chooses the best candidates for this prediction.

4.3.3.1 Pre-filtering of neighbors

In large recommender systems that can have millions of users and items, it is usually not possible to store the (non-zero) similarities between each pair of users or items, due to memory limitations. Moreover, doing so would be extremely wasteful as only the most significant of these values are used in the predictions. The pre-filtering of

neighbors is an essential step that makes neighborhood-based approaches practicable by reducing the amount of similarity weights to store, and limiting the number of candidate neighbors to consider in the predictions. There are several ways in which this can be accomplished:

- **Top-N filtering:** For each user or item, only a list of the N nearest-neighbors and their respective similarity weight is kept. To avoid problems with efficiency or accuracy, N should be chosen carefully. Thus, if N is too large, an excessive amount of memory will be required to store the neighborhood lists and predicting ratings will be slow. On the other hand, selecting a too small value for N may reduce the coverage of the recommendation method, which causes some items to be never recommended.
- **Threshold filtering:** Instead of keeping a fixed number of nearest-neighbors, this approach keeps all the neighbors whose similarity weight has a magnitude greater than a given threshold $w_{\min}$. While this is more flexible than the previous filtering technique, as only the most significant neighbors are kept, the right value of $w_{\min}$ may be difficult to determine.
- **Negative filtering:** In general, negative rating correlations are less reliable than positive ones. Intuitively, this is because strong positive correlation between two users is a good indicator of their belonging to a common group (e.g., teenagers, science-fiction fans, etc.). However, although negative correlation may indicate membership to different groups, it does not tell how different these groups are, or whether these groups are compatible for other categories of items. While experimental investigations [29, 25] have found negative correlations to provide no significant improvement in the prediction accuracy, whether such correlations can be discarded depends on the data.

Note that these three filtering approaches are not mutually exclusive and can be combined to fit the needs of the recommender system. For instance, one could discard all negative similarities *as well as* those with a magnitude lower than a given threshold.

4.3.3.2 Neighbors in the predictions

Once a list of candidate neighbors has been computed for each user or item, the prediction of new ratings is normally made with the k-nearest-neighbors, that is, the k neighbors whose similarity weight has the greatest magnitude. The important question is which value to use for k.

As shown in Table 4.3, the prediction accuracy observed for increasing values of k typically follows a *concave* function. Thus, when the number of neighbors is restricted by using a small k (e.g., $k < 20$), the prediction accuracy is normally low. As k increases, more neighbors contribute to the prediction and the variance introduced by individual neighbors is averaged out. As a result, the prediction accuracy improves. Finally, the accuracy usually drops when too many neighbors are used in

the prediction (e.g., $k > 50$), due to the fact that the few strong local relations are "diluted" by the many weak ones. Although a number of neighbors between 20 to 50 is most often described in the literature, see e.g. [28, 25], the optimal value of k should be determined by cross-validation.

On a final note, more serendipitous recommendations may be obtained at the cost of a decrease in accuracy, by basing these recommendations on a few very similar users. For example, the system could find the user most similar to the active one and recommend the new item that has received the highest rated from this user.

4.4 Advanced Techniques

The neighborhood approaches based on rating correlation, such as the ones presented in the previous sections, have two important flaws:

- **Limited coverage:** Because rating correlation measures the similarity between two users by comparing their ratings for the same items, users can be neighbors *only if* they have rated common items. This assumption is very limiting, as users having rated a few or no common items may still have similar preferences. Moreover, since only items rated by neighbors can be recommended, the coverage of such methods can also be limited.
- **Sensitivity to sparse data:** Another consequence of rating correlation, addressed briefly in Section 4.2.5, is the fact that the accuracy of neighborhood-based recommendation methods suffers from the lack of available ratings. Sparsity is a problem common to most recommender systems due to the fact that users typically rate only a small proportion of the available items [7, 25, 68, 67]. This is aggravated by the fact that users or items newly added to the system may have no ratings at all, a problem known as *cold-start* [69]. When the rating data is sparse, two users or items are unlikely to have common ratings, and consequently, neighborhood-based approaches will predict ratings using a very limited number of neighbors. Moreover, similarity weights may be computed using only a small number of ratings, resulting in biased recommendations (see Section 4.3.2.3 for this problem).

A common solution for these problems is to fill the missing ratings with default values [10, 18], such as the middle value of the rating range, and the average user or item rating. A more reliable approach is to use content information to fill out the missing ratings [16, 25, 41, 50]. For instance, the missing ratings can be provided by autonomous agents called *filterbots* [25, 41], that act as ordinary users of the system and rate items based on some specific characteristics of their content. The missing ratings can instead be predicted by a content-based approach [50], such as those described in Section 4.1.2.1. Finally, content similarity can also be used "instead of" or "in addition to" rating correlation similarity to find the nearest-neighbors employed in the predictions [7, 46, 59, 72].

These solutions, however, also have their own drawbacks. For instance, giving a default value to missing ratings may induce bias in the recommendations. Also, as discussed in Section 4.1.2.1, item content may not be available to compute ratings or similarities. This section presents two approaches proposed for the problems of limited coverage and sparsity: *dimensionality reduction* and *graph-based* methods.

4.4.1 Dimensionality Reduction Methods

Dimensionality reduction methods [4, 7, 23, 42, 67, 74, 75] address the problems of limited coverage and sparsity by projecting users and items into a reduced latent space that captures their most salient features. Because users and items are compared in this dense subspace of high-level features, instead of the "rating space", more meaningful relations can be discovered. In particular, a relation between two users can be found, even though these users have rated different items. As a result, such methods are generally less sensitive to sparse data [4, 7, 67].

There are essentially two ways in which dimensionality reduction can be used to improve recommender systems: 1) decomposition of a user-item *rating* matrix, and 2) decomposition of a sparse *similarity* matrix.

4.4.1.1 Decomposing the rating matrix

A popular dimensionality reduction approach to item recommendation is *Latent Semantic Indexing* (LSI) [15]. In this approach, the $|\mathcal{U}|\times|\mathcal{I}|$ user-item rating matrix R of rank n is approximated by a matrix $\hat{R} = PQ^\top$ of rank $k < n$, where P is a $|\mathcal{U}|\times k$ matrix of *users* factors and Q a $|\mathcal{I}|\times k$ matrix of *item* factors. Intuitively, the u-th row of P, $\mathbf{p}_u \in \mathbb{R}^k$, represents the coordinates of user u projected in the k-dimensional latent space. Likewise, the i-th row of Q, $\mathbf{q}_i \in \mathbb{R}^k$, can be seen as the coordinates of item i in this latent space. Matrices P and Q are normally found by minimizing the reconstruction error defined with the squared Frobenius norm:

$$\begin{aligned}\mathrm{err}(P,Q) &= ||R - PQ^\top||_F^2 \\ &= \sum_{u,i}\left(r_{ui} - \mathbf{p}_u\mathbf{q}_i^\top\right)^2.\end{aligned}$$

Minimizing this error is equivalent to finding the *Singular Value Decomposition* (SVD) of R [24]:

$$R = U\Sigma V^\top,$$

where U is the $|\mathcal{U}|\times n$ matrix of left singular vectors, V is the $|\mathcal{I}|\times n$ matrix of right singular vectors, and Σ is the $n\times n$ diagonal matrix of singular values. Denote by Σ_k, U_k and V_k the matrices obtained by selecting the subset containing the k highest

singular values and their corresponding singular vectors, the user and item factor matrices correspond to $P = U_k \Sigma_k^{1/2}$ and $Q = V_k \Sigma_k^{1/2}$.

Once P and Q have been obtained, the typical *model-based* prediction of a rating r_{ui} is:

$$r_{ui} \;=\; \mathbf{p}_u \mathbf{q}_i^\top .$$

There is, however, a major problem with applying SVD to the rating matrix R: most values r_{ui} of R are undefined, since there may not be a rating given to i by u. Although it is possible to assign a default value to r_{ui}, as mentioned above, this would introduce a bias in the data. More importantly, this would make the large matrix R dense and, consequently, render impractical the SVD decomposition of R. The common solution to this problem is to learn P and Q using only the known ratings [4, 42, 73, 75]:

$$\mathrm{err}(P,Q) \;=\; \sum_{r_{ui} \in \mathcal{R}} (r_{ui} - \mathbf{p}_u \mathbf{q}_i^\top)^2 \;+\; \lambda \left(||\mathbf{p}_u||^2 + ||\mathbf{q}_i||^2 \right), \tag{4.29}$$

where λ is a parameter that controls the level of regularization. A more comprehensive description of this recommendation approach can be found in Chapter 5 of this book.

In neighborhood-based recommendation, the same principle can be used to compute the similarity between users or items in the latent-space [7]. This can be done by solving the following problem:

$$\begin{aligned} \mathrm{err}(P,Q) \;&=\; \sum_{r_{ui} \in \mathcal{R}} \left(z_{ui} - \mathbf{p}_u \mathbf{q}_i^\top \right)^2 \\ &\text{subject to:} \\ &||\mathbf{p}_u|| = 1,\ \forall u \in \mathcal{U}, \quad ||\mathbf{q}_i|| = 1,\ \forall i \in \mathcal{I}, \end{aligned} \tag{4.30}$$

where z_{ui} is the mean-centered rating r_{ui} normalized to the $[-1,1]$ range. For example, if $r_{\min}$ and $r_{\max}$ are the lowest and highest values in the original rating range,

$$z_{ui} \;=\; \frac{r_{ui} - \bar{r}_u}{r_{\max} - r_{\min}}.$$

This problem corresponds to finding, for each user u and item i, coordinates on the surface of the k-dimensional unit sphere such that u will give a high rating to i if their coordinates are close together on the surface. If two users u and v are nearby on the surface, then they will give similar ratings to the same items, and, thus, the similarity between these users can be computed as

$$w_{uv} \;=\; \mathbf{p}_u \mathbf{p}_v^\top .$$

Likewise, the similarity between two items i and j can be obtained as

$$w_{ij} \;=\; \mathbf{q}_i \mathbf{q}_j^\top .$$

4.4.1.2 Decomposing the similarity matrix

The principle of this second dimensionality reduction approach is the same as the previous one: decompose a matrix into its principal factors representing projection of users or items in the latent space. However, instead of decomposing the rating matrix, a sparse similarity matrix is decomposed. Let W be a symmetric matrix of rank n representing either user or item similarities. To simplify the presentation, we will suppose the former case. Once again, we want to approximate W with a matrix $\hat{W} = PP^\top$ of lower rank $k < n$ by minimizing the following objective:

$$\begin{aligned} \mathrm{err}(P) &= ||R - PP^\top||_F^2 \\ &= \sum_{u,v} \left(w_{uv} - \mathbf{p}_u \mathbf{p}_v^\top \right)^2 . \end{aligned}$$

Matrix $\hat{W}$ can be seen as a "compressed" version of W which is less sparse than W. As before, finding the factor matrix P is equivalent to computing the eigenvalue decomposition of W:

$$W = V \Lambda V^\top,$$

where Λ is a diagonal matrix containing the $|\mathcal{U}|$ eigenvalues of W, and V is a $|\mathcal{U}| \times |\mathcal{U}|$ orthogonal matrix containing the corresponding eigenvectors. Let V_k be a matrix formed by the k principal (normalized) eigenvectors of W, which correspond to the axes of the k-dimensional latent subspace. The coordinates $\mathbf{p}_u \in \mathbb{R}^k$ of a user u in this subspace is given by the u-th row of matrix $P = V_k \Lambda_k^{1/2}$. Furthermore, the user similarities computed in this latent subspace are given by matrix

$$\begin{aligned} W' &= PP^\top \\ &= V_k \Lambda_k V_k^\top . \end{aligned} \tag{4.31}$$

This approach was used to recommend jokes in the Eigentaste system [23]. In Eigentaste, a matrix W containing the PC similarities between pairs of items is decomposed to obtain the latent subspace defined by the two principal eigenvectors of W. Denote V_2 the matrix containing these eigenvectors. A user u, represented by the u-th row $\mathbf{r}_u$ of the rating matrix R, is projected in the plane defined by V_2:

$$\mathbf{r}'_u = \mathbf{r}_u V_2 .$$

In an offline step, the users of the system are clustered in the plane using a recursive subdivision technique. Then, the rating of user u for an item i is evaluated as the mean rating for i made by users in the same cluster as u.

4.4.2 Graph-based Methods

In graph-based approaches, the data is represented in the form of a graph where nodes are users, items or both, and edges encode the interactions or similarities between the users and items. For example, in Figure 4.4, the data is modeled as a bipartite graph where the two sets of nodes represent users and items, and an edge connects user u to item i if there is a rating given to i by u in the system. A weight can also be given to this edge, such as the value of its corresponding rating. In another model, the nodes can represent either users or items, and an edge connects two nodes if the ratings corresponding two these nodes are sufficiently correlated. The weight of this edge can be the corresponding correlation value.

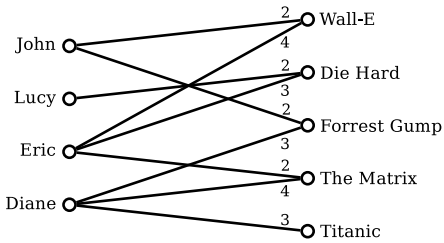

Fig. 4.4: A bipartite graph representation of the ratings of Figure 4.1 (*only ratings with value in* $\{2,3,4\}$ *are shown*).

In these models, standard approaches based on correlation predict the rating of a user u for an item i using only the nodes directly connected to u or i. Graph-based approaches, on the other hand, allow nodes that are not directly connected to influence each other by propagating information along the edges of the graph. The greater the weight of an edge, the more information is allowed to pass through it. Also, the influence of a node on another should be smaller if the two nodes are further away in the graph. These two properties, known as *propagation* and *attenuation* [26, 34], are often observed in graph-based similarity measures.

The transitive associations captured by graph-based methods can be used to recommend items in two different ways. In the first approach, the proximity of a user u to an item i in the graph is used directly to evaluate the rating of u for i [19, 26, 34]. Following this idea, the items recommended to u by the system are those that are the "closest" to u in the graph. On the other hand, the second approach considers the proximity of two users or item nodes in the graph as a measure of similarity, and uses this similarity as the weights w_{uv} or w_{ij} of a neighborhood-based recommendation method [19, 48].

4.4.2.1 Path-based similarity

In path-based similarity, the distance between two nodes of the graph is evaluated as a function of the number of paths connecting the two nodes, as well as the length of these paths.

Shortest path

A recommendation approach that computes the similarity between two users based on their shortest distance in a graph is the one described in [2]. In this method, the data is modeled as a directed graph whose nodes are users, and in which edges are determined based on the notions of *horting* and *predictability*. Horting is an asymmetric relation between two users that is satisfied if these users have rated similar items. Formally, a user u horts another user v provided either $|\mathcal{I}_{uv}| \geq \alpha$ or $|\mathcal{I}_{uv}|/|\mathcal{I}_u| \geq \beta$ is satisfied, where α, β are predetermined thresholds. Predictability, on the other hand, is a stronger property additionally requiring the ratings of u to be similar to those of v, under a mapping representing the difference in the rating scales of u and v. Thus, v predicts u, provided u horts v and there exists a linear transformation $l : \mathcal{S} \rightarrow \mathcal{S}$ such that

$$\frac{1}{|\mathcal{I}_{uv}|} \sum_{i \in \mathcal{I}_{uv}} |r_{ui} - l(r_{vi})| \;\leq\; \gamma,$$

where γ is another given threshold.

The relations of predictability are represented as directed edges in the graph, such that there is a directed edge from u to v if v predicts u. Accordingly, a directed path connecting two users u and v represents the transitive predictability of v for the ratings of u, under a sequence of transformations. Following this idea, the rating of user u for a new item i is predicted using the shortest directed paths from u to other users that have rated i. Let $P = \{u, v_1, v_2, \ldots, v_m\}$ be such a path, where $v_m \in \mathcal{U}_i$. The rating of user v_m for item i is transformed in the rating scale of u using the composition of the linear mappings along the path:

$$\hat{r}_{ui}^{(P)} \;=\; (l_m \circ \ldots \circ l_2 \circ l_1)(r_{vi}).$$

The final prediction of rating r_{ui} is computed as the average of the predictions $\hat{r}_{ui}^{(P)}$ obtained for all shortest paths P.

Number of paths

The number of paths between a user and an item in a bipartite graph can also be used to evaluate their compatibility [34]. Let R be once again the $|U| \times |I|$ rating matrix where r_{ui} equals 1 if user u has rated item i, and 0 otherwise. The adjacency matrix

A of the bipartite graph can be defined from R as

$$A = \begin{pmatrix} 0 & R^\top \\ R & 0 \end{pmatrix}.$$

In this approach, the association between a user u and an item i is defined as the sum of the weights of all distinctive paths connecting u to v (allowing nodes to appear more than once in the path), whose length is no more than a given maximum length K. Note that, since the graph is bipartite, K should be an odd number. In order to attenuate the contribution of longer paths, the weight given to a path of length k is defined as α^k, where $\alpha \in [0,1]$. Using the fact that the number of k length paths between pairs of nodes is given by A^k, the user-item association matrix S_K is

$$\begin{aligned} S_K &= \sum_{k=1}^{K} \alpha^k A^k \\ &= (I - \alpha A)^{-1}(\alpha A - \alpha^K A^K). \end{aligned} \tag{4.32}$$

This method of computing distances between nodes in a graph is known as the *Katz* measure [38]. Note that this measure is closely related to the *Von Neumann Diffusion* kernel [20, 40, 43]

$$\begin{aligned} K_{\mathrm{VND}} &= \sum_{k=0}^{\infty} \alpha^k A^k \\ &= (I - \alpha A)^{-1} \end{aligned} \tag{4.33}$$

and the *Exponential Diffusion* kernel

$$\begin{aligned} K_{\mathrm{ED}} &= \sum_{k=0}^{\infty} \frac{1}{k!} \alpha^k A^k \\ &= \exp(\alpha A), \end{aligned} \tag{4.34}$$

where $A^0 = I$.

In recommender systems that have a large number of users and items, computing these association values may require extensive computational resources. To overcome these limitations, spreading activation techniques [14] have been used in [34]. Essentially, such techniques work by first activating a selected subset of nodes as starting nodes, and then iteratively activating the nodes that can be reached directly from the nodes that are already active, until a convergence criterion is met.

4.4.2.2 Random walk similarity

Transitive associations in graph-based methods can also be defined within a probabilistic framework. In this framework, the similarity or affinity between users or items is evaluated as a probability of reaching these nodes in a random walk. For-

mally, this can be described with a first-order Markov process defined by a set of n states and a $n \times n$ transition probability matrix P such that the probability of jumping from state i to j at any time-step t is

$$p_{ij} \;=\; \Pr\big(s(t+1) = j | s(t) = i\big).$$

Denote $\pi(t)$ the vector containing the state probability distribution of step t, such that $\pi_i(t) = \Pr(s(t) = i)$, the evolution of the Markov chain is characterized by

$$\pi(t+1) \;=\; P^\top \pi(t).$$

Moreover, under the condition that P is row-stochastic, i.e. $\sum_j p_{ij} = 1$ for all i, the process converges to a stable distribution vector $\pi(\infty)$ corresponding to the positive eigenvector of $P^\top$ with an eigenvalue of 1. This process is often described in the form of a weighted graph having a node for each state, and where the probability of jumping from a node to an adjacent node is given by the weight of the edge connecting these nodes.

Itemrank

A recommendation approach, based on the PageRank algorithm for ranking Web pages [11], is ItemRank [26]. This approach ranks the preferences of a user u for new items i as the probability of u to visit i in a random walk of a graph in which nodes correspond to the items of the system, and edges connect items that have been rated by common users. The edge weights are given by the $|\mathcal{I}| \times |\mathcal{I}|$ transition probability matrix P for which $p_{ij} = |\mathcal{U}_{ij}|/|\mathcal{U}_i|$ is the estimated conditional probability of a user to rate an item j if it has rated an item i.

As in PageRank, the random walk can, at any step t, either jump using P to an adjacent node with fixed probability α, or "teleport" to any node with probability $(1-\alpha)$. Let $\mathbf{r}_u$ be the u-th row of the rating matrix R, the probability distribution of user u to teleport to other nodes is given by vector $\mathbf{d}_u = \mathbf{r}_u/||\mathbf{r}_u||$. Following these definitions, the state probability distribution vector of user u at step $t+1$ can be expressed recursively as

$$\pi_u(t+1) \;=\; \alpha P^\top \pi_u(t) \;+\; (1-\alpha)\mathbf{d}_u. \tag{4.35}$$

For practical reasons, $\pi_u(\infty)$ is usually obtained with a procedure that first initializes the distribution as uniform, i.e. $\pi_u(0) = \frac{1}{n}\mathbf{1}_n$, and then iteratively updates π_u, using (4.35), until convergence. Once $\pi_u(\infty)$ has been computed, the system recommends to u the item i for which π_{ui} is the highest.

Average first-passage/commute time

Other distance measures based on random walks have been proposed for the recommendation problem. Among these are the *average first-passage time* and the *average commute time* [19, 20]. The average first-passage time $m(j|i)$ [56] is the average number of steps needed by a random walker to reach a node j for the first time, when starting from a node $i \neq j$. Let P be the $n \times n$ transition probability matrix, $m(j|i)$ can be expressed recursively as

$$m(j|i) = \begin{cases} 0 & , \text{ if } i = j \\ 1 + \sum_{k=1}^{n} p_{ik} m(j|k) & , \text{ otherwise} \end{cases}$$

A problem with the average first-passage time is that it is not symmetric. A related measure that does not have this problem is the average commute time $n(i,j) = m(j|i) + m(i|j)$ [22], corresponding to the average number of steps required by a random walker starting at node $i \neq j$ to reach node j for the first time and go back to i. This measure has several interesting properties. Namely, it is a true distance measure in some Euclidean space [22], and is closely related to the well-known property of resistance in electrical networks and to the pseudo-inverse of the graph Laplacian matrix [19].

In [19], the average commute time is used to compute the distance between the nodes of a bipartite graph representing the interactions of users and items in a recommender system. For each user u there is a directed edge from u to every item $i \in \mathcal{I}_u$, and the weight of this edge is simply $1/|\mathcal{I}_u|$. Likewise, there is a directed edge from each item i to every user $u \in \mathcal{U}_i$, with weight $1/|\mathcal{U}_i|$. Average commute times can be used in two different ways: 1) recommending to u the item i for which $n(u,i)$ is the smallest, or 2) finding the users nearest to u, according to the commute time distance, and then suggest to u the item most liked by these users.

4.5 Conclusion

One of the earliest approaches proposed for the task item recommendation, neighborhood-based recommendation still ranks among the most popular methods for this problem. Although quite simple to describe and implement, this recommendation approach has several important advantages, including its ability to explain a recommendation with the list of the neighbors used, its computational and space efficiency which allows it to scale to large recommender systems, and its marked stability in an online setting where new users and items are constantly added. Another of its strengths is its potential to make serendipitous recommendations that can lead users to the discovery of unexpected, yet very interesting items.

In the implementation of a neighborhood-based approach, one has to make several important decisions. Perhaps the one having the greatest impact on the accuracy

and efficiency of the recommender system is choosing between a user-based and an item-based neighborhood method. In typical commercial recommender systems, where the number of users far exceeds the number of available items, item-based approaches are typically preferred since they provide more accurate recommendations, while being more computationally efficient and requiring less frequent updates. On the other hand, user-based methods usually provide more original recommendations, which may lead users to a more satisfying experience. Moreover, the different components of a neighborhood-based method, which include the normalization of ratings, the computation of the similarity weights and the selection of the nearest-neighbors, can also have a significant influence on the quality of the recommender system. For each of these components, several different alternatives are available. Although the merit of each of these has been described in this document and in the literature, it is important to remember that the "best" approach may differ from one recommendation setting to the next. Thus, it is important to evaluate them on data collected from the actual system, and in light of the particular needs of the application.

Finally, when the performance of a neighborhood-based approach suffers from the problems of limited coverage and sparsity, one may explore techniques based on dimensionality reduction or graphs. Dimensionality reduction provides a compact representation of users and items that captures their most significant features. An advantage of such an approach is that it can obtain meaningful relations between pairs of users or items, even though these users have rated different items, or these items were rated by different users. On the other hand, graph-based techniques exploit the transitive relations in the data. These techniques also avoid the problems of sparsity and limited coverage by evaluating the relationship between users or items that are not "directly connected". However, unlike dimensionality reduction, graph-based methods also preserve some of the "local" relations in the data, which are useful in making serendipitous recommendations.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering **17**(6), 734–749 (2005)
2. Aggarwal, C.C., Wolf, J.L., Wu, K.L., Yu, P.S.: Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In: KDD '99: Proc. of the 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp. 201–212. ACM, New York, NY, USA (1999)
3. Balabanović, M., Shoham, Y.: Fab: Content-based, collaborative recommendation. Communications of the ACM **40**(3), 66–72 (1997)
4. Bell, R., Koren, Y., Volinsky, C.: Modeling relationships at multiple scales to improve accuracy of large recommender systems. In: KDD '07: Proc. of the 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp. 95–104. ACM, New York, NY, USA (2007)
5. Bell, R.M., Koren, Y.: Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In: ICDM '07: Proc. of the 2007 Seventh IEEE Int. Conf. on Data Mining, pp. 43–52. IEEE Computer Society, Washington, DC, USA (2007)

6. Billsus, D., Brunk, C.A., Evans, C., Gladish, B., Pazzani, M.: Adaptive interfaces for ubiquitous web access. Communications of the ACM **45**(5), 34–38 (2002)
7. Billsus, D., Pazzani, M.J.: Learning collaborative information filters. In: ICML '98: Proc. of the 15th Int. Conf. on Machine Learning, pp. 46–54. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)
8. Billsus, D., Pazzani, M.J.: User modeling for adaptive news access. User Modeling and User-Adapted Interaction **10**(2-3), 147–180 (2000)
9. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. Journal of Machine Learning Research **3**, 993–1022 (2003)
10. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: Proc. of the 14th Annual Conf. on Uncertainty in Artificial Intelligence, pp. 43–52. Morgan Kaufmann (1998)
11. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems **30**(1-7), 107–117 (1998)
12. Buckley, C., Salton, G.: Optimization of relevance feedback weights. In: SIGIR '95: Proc. of the 18th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 351–357. ACM, New York, NY, USA (1995)
13. Cohen, W.W., Schapire, R.E., Singer, Y.: Learning to order things. In: NIPS '97: Proc. of the 1997 Conf. on Advances in Neural Information Processing Systems, pp. 451–457. MIT Press, Cambridge, MA, USA (1998)
14. Crestani, F., Lee, P.L.: Searching the Web by constrained spreading activation. Information Processing and Management **36**(4), 585–605 (2000)
15. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. Journal of the American Society for Information Science **41**, 391–407 (1990)
16. Degemmis, M., Lops, P., Semeraro, G.: A content-collaborative recommender that exploits wordnet-based user profiles for neighborhood formation. User Modeling and User-Adapted Interaction **17**(3), 217–255 (2007)
17. Delgado, J., Ishii, N.: Memory-based weighted majority prediction for recommender systems. In: Proc. of the ACM SIGIR'99 Workshop on Recommender Systems (1999)
18. Deshpande, M., Karypis, G.: Item-based top-N recommendation algorithms. ACM Transaction on Information Systems **22**(1), 143–177 (2004)
19. Fouss, F., Renders, J.M., Pirotte, A., Saerens, M.: Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. IEEE Transactions on Knowledge and Data Engineering **19**(3), 355–369 (2007)
20. Fouss, F., Yen, L., Pirotte, A., Saerens, M.: An experimental investigation of graph kernels on a collaborative recommendation task. In: ICDM '06: Proc. of the 6th Int. Conf. on Data Mining, pp. 863–868. IEEE Computer Society, Washington, DC, USA (2006)
21. Freund, Y., Iyer, R.D., Schapire, R.E., Singer, Y.: An efficient boosting algorithm for combining preferences. In: ICML '98: Proc. of the 15th Int. Conf. on Machine Learning, pp. 170–178. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)
22. Gobel, F., Jagers, A.: Random walks on graphs. Stochastic Processes and Their Applications **2**, 311–336 (1974)
23. Goldberg, K., Roeder, T., Gupta, D., Perkins, C.: Eigentaste: A constant time collaborative filtering algorithm. Information Retrieval **4**(2), 133–151 (2001)
24. Golub, G.H., Van Loan, C.F.: Matrix computations (3rd ed.). Johns Hopkins University Press (1996)
25. Good, N., Schafer, J.B., Konstan, J.A., Borchers, A., Sarwar, B., Herlocker, J., Riedl, J.: Combining collaborative filtering with personal agents for better recommendations. In: AAAI '99/IAAI '99: Proc. of the 16th National Conf. on Artificial Intelligence, pp. 439–446. American Association for Artificial Intelligence, Menlo Park, CA, USA (1999)
26. Gori, M., Pucci, A.: Itemrank: a random-walk based scoring algorithm for recommender engines. In: Proc. of the 2007 IJCAI Conf., pp. 2766–2771 (2007)

27. Grcar, M., Fortuna, B., Mladenic, D., Grobelnik, M.: k-NN versus SVM in the collaborative filtering framework. Data Science and Classification pp. 251–260 (2006). URL `http://db.cs.ualberta.ca/webkdd05/proc/paper25-mladenic.pdf`
28. Herlocker, J., Konstan, J.A., Riedl, J.: An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. Inf. Retr. **5**(4), 287–310 (2002)
29. Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In: SIGIR '99: Proc. of the 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 230–237. ACM, New York, NY, USA (1999)
30. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. **22**(1), 5–53 (2004)
31. Hill, W., Stead, L., Rosenstein, M., Furnas, G.: Recommending and evaluating choices in a virtual community of use. In: CHI '95: Proc. of the SIGCHI Conf. on Human Factors in Computing Systems, pp. 194–201. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1995)
32. Hofmann, T.: Collaborative filtering via Gaussian probabilistic latent semantic analysis. In: SIGIR '03: Proc. of the 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 259–266. ACM, New York, NY, USA (2003)
33. Howe, A.E., Forbes, R.D.: Re-considering neighborhood-based collaborative filtering parameters in the context of new data. In: CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management, pp. 1481–1482. ACM, New York, NY, USA (2008)
34. Huang, Z., Chen, H., Zeng, D.: Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. ACM Transactions on Information Systems **22**(1), 116–142 (2004)
35. Jin, R., Chai, J.Y., Si, L.: An automatic weighting scheme for collaborative filtering. In: SIGIR '04: Proc. of the 27th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 337–344. ACM, New York, NY, USA (2004)
36. Jin, R., Si, L., Zhai, C.: Preference-based graphic models for collaborative filtering. In: Proc. of the 19th Annual Conf. on Uncertainty in Artificial Intelligence (UAI-03), pp. 329–33. Morgan Kaufmann, San Francisco, CA (2003)
37. Jin, R., Si, L., Zhai, C., Callan, J.: Collaborative filtering with decoupled models for preferences and ratings. In: CIKM '03: Proc. of the 12th Int. Conf. on Information and Knowledge Management, pp. 309–316. ACM, New York, NY, USA (2003)
38. Katz, L.: A new status index derived from sociometric analysis. Psychometrika **18**(1), 39–43 (1953)
39. Kendall, M., Gibbons, J.D.: Rank Correlation Methods, 5 edn. Charles Griffin (1990)
40. Kondor, R.I., Lafferty, J.D.: Diffusion kernels on graphs and other discrete input spaces. In: ICML '02: Proc. of the Nineteenth Int. Conf. on Machine Learning, pp. 315–322. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002)
41. Konstan, J.A., Miller, B.N., Maltz, D., Herlocker, J.L., Gordon, L.R., Riedl, J.: GroupLens: applying collaborative filtering to usenet news. Communications of the ACM **40**(3), 77–87 (1997)
42. Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: KDD'08: Proceeding of the 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp. 426–434. ACM, New York, NY, USA (2008)
43. Kunegis, J., Lommatzsch, A., Bauckhage, C.: Alternative similarity functions for graph kernels. In: Proc. of the Int. Conf. on Pattern Recognition (2008)
44. Lang, K.: News Weeder: Learning to filter netnews. In: Proc. of the 12th Int. Conf. on Machine Learning, pp. 331–339. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA (1995)
45. Last.fm: Music recommendation service (2009). `http://www.last.fm`
46. Li, J., Zaiane, O.R.: Combining usage, content, and structure data to improve Web site recommendation. In: Proc. of the 5th Int. Conf. on Electronic Commerce and Web Technologies (EC-Web) (2004)
47. Linden, G., Smith, B., York, J.: Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing **7**(1), 76–80 (2003)

48. Luo, H., Niu, C., Shen, R., Ullrich, C.: A collaborative filtering framework based on both local user similarity and global user similarity. Machine Learning **72**(3), 231–245 (2008)
49. Ma, H., King, I., Lyu, M.R.: Effective missing data prediction for collaborative filtering. In: SIGIR '07: Proc. of the 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 39–46. ACM, New York, NY, USA (2007)
50. Melville, P., Mooney, R.J., Nagarajan, R.: Content-boosted collaborative filtering for improved recommendations. In: 18th National Conf. on Artificial intelligence, pp. 187–192. American Association for Artificial Intelligence, Menlo Park, CA, USA (2002)
51. Miller, B.N., Albert, I., Lam, S.K., Konstan, J.A., Riedl, J.: Movielens unplugged: experiences with an occasionally connected recommender system. In: IUI '03: Proc. of the 8th Int. Conf. on Intelligent User Interfaces, pp. 263–266. ACM, New York, NY, USA (2003)
52. Mobasher, B., Dai, H., Luo, T., Nakagawa, M.: Discovery and evaluation of aggregate usage profiles for Web personalization. Data Mining and Knowledge Discovery **6**(1), 61–82 (2002)
53. Mooney, R.J.: Content-based book recommending using learning for text categorization. In: Proc. of the Fifth ACM Conf. on Digital Libraries, pp. 195–204. ACM Press (2000)
54. Nakamura, A., Abe, N.: Collaborative filtering using weighted majority prediction algorithms. In: ICML '98: Proc. of the 15th Int. Conf. on Machine Learning, pp. 395–403. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)
55. Netflix: Online movie rental service (2009). `http://www.netflix.com`
56. Norris, J.R.: Markov Chains, 1 edn. Cambridge University Press, Cambridge (1999)
57. Paterek, A.: Improving regularized singular value decomposition for collaborative filtering. In: Proceedings of the KDD Cup and Workshop (2007)
58. Pazzani, M., Billsus, D.: Learning and revising user profiles: The identification of interesting Web sites. Machine Learning **27**(3), 313–331 (1997)
59. Pazzani, M.J.: A framework for collaborative, content-based and demographic filtering. Artificial Intelligence Review **13**(5-6), 393–408 (1999)
60. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: GroupLens: An open architecture for collaborative filtering of netnews. In: CSCW '94: Proc. of the 1994 ACM Conf. on Computer Supported Cooperative Work, pp. 175–186. ACM, New York, NY, USA (1994)
61. Rich, E.: User modeling via stereotypes. Cognitive Science **3**(4), 329–354 (1979)
62. Rissanen, J.: Modeling by shortest data description. Automatica **14**, 465–471 (1978)
63. Rocchio, J.: Relevance Feedback in Information Retrieval. Prentice Hall, Englewood, Cliffs, New Jersey (1971)
64. Salakhutdinov, R., Mnih, A., Hinton, G.: Restricted Boltzmann machines for collaborative filtering. In: ICML '07: Proceedings of the 24th international conference on Machine learning, pp. 791–798. ACM, New York, NY, USA (2007)
65. Salton, G. (ed.): Automatic text processing. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1988)
66. Sarwar, B., Karypis, G., Konstan, J., Reidl, J.: Item-based collaborative filtering recommendation algorithms. In: WWW '01: Proc. of the 10th Int. Conf. on World Wide Web, pp. 285–295. ACM, New York, NY, USA (2001)
67. Sarwar, B.M., Karypis, G., Konstan, J.A., Riedl, J.T.: Application of dimensionality reduction in recommender systems A case study. In: ACM WebKDD Workshop (2000)
68. Sarwar, B.M., Konstan, J.A., Borchers, A., Herlocker, J., Miller, B., Riedl, J.: Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In: CSCW '98: Proc. of the 1998 ACM Conf. on Computer Supported Cooperative Work, pp. 345–354. ACM, New York, NY, USA (1998)
69. Schein, A.I., Popescul, A., Ungar, L.H., Pennock, D.M.: Methods and metrics for cold-start recommendations. In: SIGIR '02: Proc. of the 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 253–260. ACM, New York, NY, USA (2002)
70. Shardanand, U., Maes, P.: Social information filtering: Algorithms for automating "word of mouth". In: CHI '95: Proc. of the SIGCHI Conf. on Human factors in Computing Systems, pp. 210–217. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1995)

71. Sheth, B., Maes, P.: Evolving agents for personalized information filtering. In: Proc. of the 9th Conf. on Artificial Intelligence for Applications, pp. 345–352 (1993)
72. Soboroff, I.M., Nicholas, C.K.: Combining content and collaboration in text filtering. In: Proc. of the IJCAI'99 Workshop on Machine Learning for Information Filtering, pp. 86–91 (1999)
73. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Major components of the gravity recommendation system. SIGKDD Exploration Newsletter **9**(2), 80–83 (2007)
74. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Investigation of various matrix factorization methods for large recommender systems. In: Proc. of the 2nd KDD Workshop on Large Scale Recommender Systems and the Netflix Prize Competition (2008)
75. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Scalable collaborative filtering approaches for large recommender systems. Journal of Machine Learning Research (Special Topic on Mining and Learning with Graphs and Relations) **10**, 623–656 (2009)
76. Terveen, L., Hill, W., Amento, B., McDonald, D., Creter, J.: PHOAKS: a system for sharing recommendations. Communications of the ACM **40**(3), 59–62 (1997)
77. Zhang, Y., Callan, J., Minka, T.: Novelty and redundancy detection in adaptive filtering. In: SIGIR '02: Proc. of the 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 81–88. ACM, New York, NY, USA (2002)
78. Zitnick, C.L., Kanade, T.: Maximum entropy for collaborative filtering. In: AUAI '04: Proc. of the 20th Conf. on Uncertainty in Artificial Intelligence, pp. 636–643. AUAI Press, Arlington, Virginia, United States (2004)