Chapter 11

Social and Trust-Centric Recommender Systems

"*Society is like a large piece of frozen water; and skating well is the great art of social life.*"– Letitia Elizabeth Landon

11.1 Introduction

With increasing access to social information about users, merchants can directly incorporate social context in collaborative filtering algorithms. Although some of these methods are discussed in Chapter 10, the focus of this chapter is primarily on recommending nodes and links in network settings. Social context is a much broader concept, not only including social (network) links, but also various types of side information, such as tags or folksonomies. Furthermore, the social context can also be understood in a network-agnostic way, as a special case of context-sensitive recommender systems (cf. Chapter 8). The social setting results in a number of human-centric factors, such as *trust*. When users are aware of the identity of the actors who participate in the feedback process, the trust factor plays an important role. Therefore, the material in this chapter is closely related to that in Chapter 10, but nevertheless it is distinct enough a merit a separate chapter in its own right. In particular, we will study the following aspects of social context in recommender systems:

1. *Social context as a special case of context-aware recommender systems:* Context-aware recommender systems are discussed in Chapter 8. An important framework for contextual recommendations is that of the multidimensional model [6]. One of the possible forms of context is the *social context* in which the social information is used as side information to improve the effectiveness of the recommendation process. For example, the choice of movie that a user might watch depends on the companion with whom she chooses

C.C. Aggarwal, *Recommender Systems: The Textbook*,
DOI 10.1007/978-3-319-29659-3_11

to watch the movie. In other words, she would often choose a different film depending on whether she was watching it with her friends, parents, or significant other. Such recommendations can be handled directly with the multidimensional model, without the need to use the structure of the social network in the recommendation process.

2. *Social context from a network-centric and trust-centric perspective:* In these cases, it is assumed that the merchant is aware of the social structure of the user. Users are often likely to ask their friends for suggestions about movies, restaurants, or other items. Therefore, the social structure of a user can be viewed as a *social trust network*, which is useful for the recommendation process. For example, if a user is friends with many people who have watched a particular movie, then she is more likely to watch that movie. Furthermore, if the user is densely connected with a community of users interested in the movie, this provides further evidence of user interest. Therefore, the network structure and interests in the vicinity of the user play a key role in the recommendation process.

   In some networks, such as Epinions.com [705], trust networks are created between users, which provide feedback on how much users can rely on each other's opinions in the recommendation process. The trust factor is particularly important because a user's personalized interests can be better predicted from the rating patterns of other users whom she has trusted in the past. It has now been conclusively shown in the literature that the incorporation of trust has a significant and positive effect on the recommendation process. Such methods are closely related to some of the network-centric methods discussed in Chapter 10. Here, we discuss these methods in further detail, especially in the context of trust-centric systems.

3. *User interaction perspective:* The user interactions with social networks create many forms of feedback, such as comments or tags. These tags can be viewed as *folksonomies*, which collaboratively annotate and classify content. Such folksonomies are very informative and can be used to improve the recommendation process. These methods are closely related to content-centric recommendations except that a combination of a collaborative and content-centric approach is used. This is particularly natural because sufficient data is available in such cases to leverage both collaborative and content-centric factors.

It is noteworthy that these methods apply to completely different recommendation settings and input data. Furthermore, the social information is used in a completely different way in each of these settings. Therefore, social recommender systems can be understood from many different perspectives, depending on whether social participants serve as the context, as the peer recommenders or as the providers of interaction data.

In this chapter, we will discuss all the aforementioned scenarios for social network recommendations. We will discuss the key settings in which each of these methods apply, and also the settings in which they work most effectively. We will also discuss how many of these techniques relate to the methods discussed in earlier chapters. The use of multidimensional context to address the social setting is closely related to the techniques introduced in Chapter 8. On the other hand, the use of network-centric methods is closely related to the techniques introduced in Chapter 10. The discussion in this chapter expands on these themes as they relate to the social context.

This chapter is organized as follows. In section 11.2, we will discuss the use of social context as a special case of social recommender systems. In other words, we will discuss the use of the multidimensional model [6] to address social context. Network-centric methods for social recommendations are discussed in section 11.3. The utilization of user interaction in social recommendations is discussed in section 11.4. A summary is given in section 11.5.

11.2 Multidimensional Models for Social Context

The multidimensional model of Chapter 8 is the simplest method for incorporating social information within the recommendation process. This approach has the merit that we can reuse traditional collaborative filtering models by using the reduction-based approach of Chapter 8. The use of ratings associated with a social context is one instance where this approach is applicable. Data about social context may either be directly collected or inferred from other information sources. Some typical modes of collecting data about social context are as follows:

1. *Explicit feedback:* While rating items, such as a movie, the system can be designed to capture various types of information, such as details about who the movie was watched with. Similarly, the destination of a tourist might depend on their travel companion. For example, a tourist is far more likely to travel to Disneyland than to Las Vegas when her children accompany her. The main challenge with this approach is that users are generally not very willing to spend too much effort in specifying such contextual details while providing ratings. Therefore, it becomes more difficult to collect sufficient data. Nevertheless, when it is possible to collect such data through explicit feedback, it is generally of high quality. Therefore, it should be considered the first choice where possible.

2. *Implicit feedback:* The social context of a user can often be inferred from where, when, and how the item was bought, or her other social activities. For example, if a tourist uses the same credit card to book a set of tickets for herself and her travel companions, this provides useful contextual information to the tour operator for future recommendations. In some cases, the collection of contextual data might require the use of machine learning techniques. With the increasing availability of mobile phones and the ability to perform online user activity analysis, it has become increasingly easy to collect such information in an automated way.

Let U be the set of users, I be the set of items, and C be the set of alternatives representing the social context. The ratings can then be viewed as a mapping g_R on a 3-dimensional ratings cube R. The domain of the mapping is defined by $U \times I \times C$, and the range corresponds to the values of the ratings. This mapping may be written as follows:

$$g_R : U \times I \times C \rightarrow \textit{rating}$$

For example, consider a travel recommendation application in which the context is the travel companion. Figure 11.1 illustrates an example of a 3-dimensional ratings matrix with social context. Here, the items correspond to the tourist locations and the contexts correspond to the travel companions. Each entry in the cube corresponds to the rating of a specific travel location of a user in a particular context. It is noteworthy that this example is a simple adaptation of Figure 8.3 in Chapter 8 to fit the social context. It is also possible to have multiple social contexts. In such a case, the dimensionality of the underlying cube will increase accordingly and one can work with a w-dimensional cube of ratings. It is evident

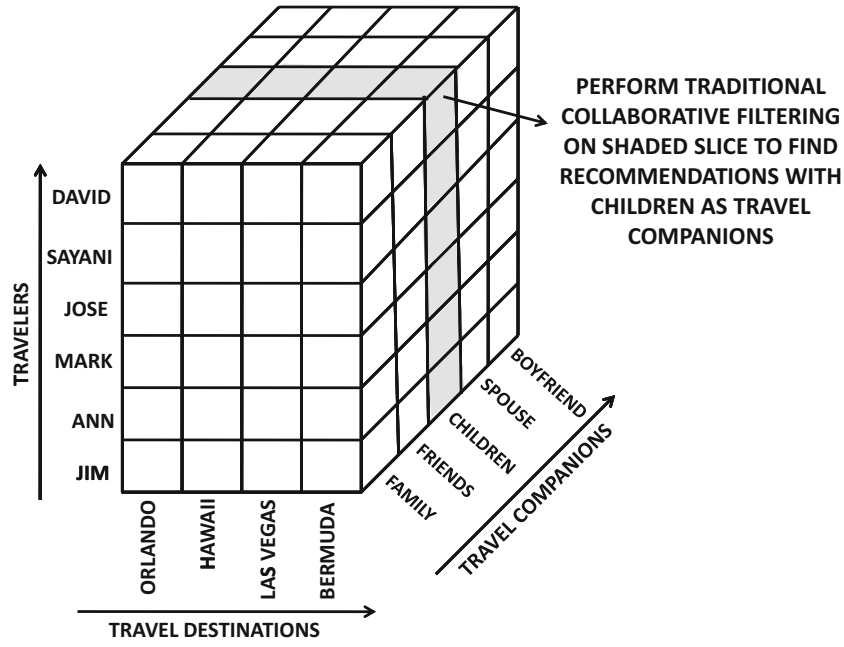


Figure 11.1: Travel recommendations with varying social context (Adaptation of Figure 8.3 in Chapter 8)

that the multidimensional model in the case of social context is not very different from that in the case of other types of context. Therefore, the algorithms discussed in Chapter 8 can be generalized to this scenario in a relatively painless way.

Queries can be posed in a way that is similar to multidimensional context by partitioning the dimensions into "what" dimensions and "for whom" dimensions. A typical query is of the following form:

Determine the top-k possibilities in the "what" dimensions for a particular set of specified values in the "for whom" dimensions.

In the aforementioned example, some possibilities for the various queries include:

1. Determine the top-k destinations for a particular user.
2. Determine the top-k destination-companion pairs for a particular user.
3. Determine the top-k destinations for a particular user-companion pair.
4. Determine the top-k companions for a particular user-destination pair.

The reduction approach of section 8.2 in Chapter 8 can be used to provide responses to such queries. Let $f_{R'} : U \times I \rightarrow \textit{rating}$ be a conventional collaborative filtering algorithm on a 2-dimensional ratings matrix R'. Then, each of the aforementioned queries can be reduced to a standard collaborative filtering problem. For example, in order to determine the best destinations to visit with children, one can extract the corresponding 2-dimensional slice $R'(\textit{children})$ from the original 3-dimensional ratings matrix R. This slice is shaded in

Figure 11.1. Then, a standard collaborative filtering algorithm can be applied to this 2-dimensional matrix. In the event that there are multiple travel companions, this results in context set V. The data cube slices for each of the contexts in V can be extracted and the ratings can be averaged over the different contextual values for a particular user-item combination. This process is similar to that of Equation 8.2 in Chapter 8:

$$g_R(User, Item, V) = \text{AVERAGE}_{[y \in V]}\, g_R(User, Item, y) \tag{11.1}$$

Therefore, the problem can again be reduced to the 2-dimensional case with the use of the averaged slice over the social context set V. A similar approach can be used for the problem of determining the top-k destinations of a particular user without any specific context in mind. In such a case, the ratings can be averaged over all the different contexts rather than a specific social context set V. This approach is referred to as prefiltering. However, other methods, such as postfiltering, latent factor models, or other machine learning models, are also discussed in Chapter 11. All these methods can be easily generalized for providing recommendations in these settings.

11.3 Network-Centric and Trust-Centric Methods

The basic idea in network-centric methods is that the friendship structure of a user has a profound influence on her tastes, choices, or consumption patterns. Users often ask their friends for suggestions regarding movies, travel, or other items. Furthermore, social connections exhibit the well-known principle of *homophily*, in which connected users often have similar interests and tastes. This similarity in tastes often leads a user to trust the recommendations from their connected users more than others. Numerous methods can be designed to incorporate such links into the recommendation process. Although such links may have varying effectiveness depending on the application domain, they are usually particularly helpful in cases of cold-start, when there is little information available about a particular user's ratings. In such cases, the knowledge embedded in the user's social connections can be particularly helpful in identifying her most relevant peers. In the following, a number of key methods will be discussed for incorporating social knowledge into the recommendation process. First, we will discuss the two important concepts of trust and homophily, which are related but not quite the same.

11.3.1 Collecting Data for Building Trust Networks

Trust and homophily both play an important role in the social recommendation process. These concepts are related but they are not quite the same. *Homophily* refers to the fact that linked users in social networks are likely to be similar to one another in terms of their tastes and interests. *Trust* refers to the fact that users are more likely to trust the tastes and recommendations of their friends. In some cases, trust is the consequence of homophily. As linked users tend to be similar to one another, they tend to trust each other's tastes and recommendations. The strong correlation between trust and homophily has been shown in [224, 681].

Either homophily or trust or both may be relevant in a given network. In some social networks, such as Facebook, both homophily and trust are relevant because links typically represent friendship relationships. In fact, trust relationships can often be computationally inferred from such Web-based social networks [226]. Many characteristics, such as feature

similarity and email exchanges can be used to infer the trust links. For example, one might use the following user-to-user similarity [588] to determine the trust t_{ij} between user i and j.

$$t_{ij} = \begin{cases} \text{Cosine}(i, j) & \text{if } i \text{ and } j \text{ are connected} \\ \text{Undefined} & \text{otherwise} \end{cases} \tag{11.2}$$

The cosine similarity is calculated on the ratings of users i and j. It is noteworthy that when i and j are not connected, the trust between them is undefined. As we will see later, this undefined value can also be inferred with the use of trust propagation methods. Therefore, such methods infer the trust values between connected users in a different way from how they infer trust values between unconnected users.

The aforementioned methodology can be viewed as an implicit way of inferring the trust. In some networks such as Epinions [705], the trust links are explicitly specified by the users. Some examples of such networks are as follows:

1. In Golbeck's Filmtrust system [225], users are asked to evaluate their trust in their acquaintances' ratings in addition to providing ratings. This data is then used to make recommendations.

2. In the Epinions site [705], users are explicitly asked to specify the other users that they trust or distrust.

3. In the Moleskiing site [461], the inter-user trust information is obtained via explicit feedback. Users are allowed to rate how useful they found the comments by other users. This can help in inferring trust links between users. When a user frequently expresses a positive opinion about the comments of another user, a directed edge can be added from the former to the latter. A modeling approach may be used to relate this frequency to an explicit trust value. An example of such a modeling approach is provided in [591], although this work is focused on distrust relations rather than trust relations. The ability to leave feedback about reviews is also available on sites such as Amazon.com.

4. Trust and distrust relationships are also available in the Slashdot network [706], which is a technology blog. In this case, the relevant trust relationships are directly specified by the users.

In all cases, whether the trust relationships are implicitly inferred or explicitly specified by the users, a *trust network* can be created. This trust network is also referred to as the *Web of trust*. For the purpose of this chapter, we will assume that trust is specified as an $m \times m$ user-user matrix $T = [t_{ij}]$, in which each value of t_{ij} is drawn from the range $(0, 1)$. Large values of t_{ij} indicate user i trusts user j to a greater degree. The case where $t_{ij} \in (0, 1)$ represents the *probabilistic* model of trust representation. This representation provides a way of modeling trust but not distrust. In general, the value of t_{ij} may not be the same as that of t_{ji}, although some implicit inference models may make this assumption.

In some cases, distrust relationships are also available. For example, Epinions provides users the option to specify *block lists* corresponding to distrusted users. Ideally, distrust relations should be negative values and one can extend the model to use values in $[-1, +1]$. However, it is often challenging to generalize inference algorithms to trust networks with both trust and distrust relationships. Most of the work in the literature has focused only on using the trust relationships while ignoring the distrust relations. Therefore, most of

the discussion in this section will be based on positive trust relations between nodes. The bibliographic notes contain more details about methods that use distrust relations.

Trust-aware recommender systems are able to use the knowledge in the Web of trust in order to make personalized and accurate recommendations. Such recommender systems are also referred to as *trust-enhanced recommender systems*. Many of these methods use specialized operators, referred to as *trust aggregation* and *trust propagation*. These are mechanisms to estimate the unknown trust level between two users with the use of *transitivity* in the trust network. In other words, once it is known how much A trusts B and how much B trusts C, it can also be estimated how much A trusts C. *Trust metrics* estimate how much one user should trust another based on existing trust relationships in the network [682].

Trust networks are directed, especially when they are explicitly specified by users. This is because trust relations are asymmetric. The level of trust of A for B may be different from that of B for A. Most trust-based algorithms take the edge directions into account during the computation. However, in some cases, the simplifying assumption of undirected networks is made, especially when the trust relations are implicitly inferred from Web-based social networks. For example, the trust relation of Equation 11.2 is symmetric.

11.3.2 Trust Propagation and Aggregation

Trust propagation and aggregation play an important role in the design of social recommender systems. These operators are motivated by the fact that trust networks are sparsely specified, in which all pairs of users do not necessarily have trust relationships between them. Therefore, the transitivity in trust relations needs to be used to infer the missing trust relations with the use of operators like propagation and aggregation.

What does transitivity mean? For example, if Alice trusts John and John trusts Bob, then one can infer the fact that Alice might trust Bob. This fact is, in turn, useful for Alice to make recommendations based on the items that Bob might have liked. In other words, one needs to determine *paths* in the trust network in order to make such inferences. The determination of the unknown value of the trust between two nodes at the end points of a path is referred to as trust propagation. However, there are typically multiple paths in a trust network between a pair of users. For example, in the simple trust network of Figure 11.2, the trust values on the edges are assumed to be drawn from $(0, 1)$. The value on the directed edge from any user A to any other user B indicates how much A trusts B. There are two paths between Alice and Bob, and the (propagated) trust values between Alice and Bob need to be aggregated over these two paths. When quantifying Alice's trust in Bob, Alice is the *source* and Bob is the *sink*. The trust propagation and trust aggregation operators are computed as follows:

1. *Trust propagation along a single path:* A multiplicative approach is commonly used for trust propagation [241, 509]. In this case, the trust values on the edges are multiplied in order to obtain the trust between the two endpoints. For example, consider the path Alice $\rightarrow$ John $\rightarrow$ Bob in Figure 11.2. In this case, multiplying the trust values on the path yields a propagated trust value of $0.7 \times 0.6 = 0.42$. Similarly, for the path Alice $\rightarrow$ Mary $\rightarrow$ Tim $\rightarrow$ Bob, the multiplicatively propagated trust value is $0.3 \times 0.4 \times 1 = 0.12$. Many methods also use trust decay to deemphasize long paths, or simply use shortest paths. For example, a user-defined decay factor $\beta < 1$ is used to multiply the computed trust value with β^q, where q is the path length of the propagation. In Figure 11.2, the decay approach would multiply the propagated results with β^2 for the upper path and β^3 for the lower path. The resulting computed values

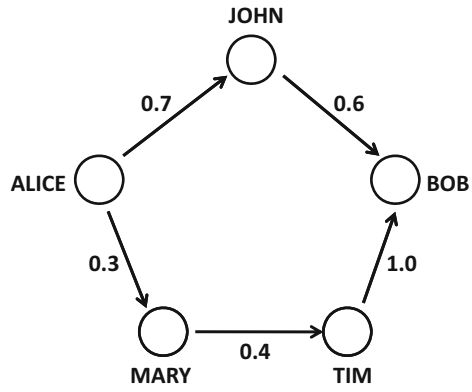


Figure 11.2: A simple trust network

are $0.42 \times \beta^2$ and $0.12 \times \beta^3$, respectively. More sophisticated methods for incorporating decay include the *Appleseed* algorithm [682], in which a spreading activation model is used.

These types of multiplicative propagation algorithms are only designed for non-negative trust values in the range $(0, 1)$. Distrust relations pose significant challenges because a sequence of two distrusts does not always imply a trust relationship [241, 590, 591]. Therefore, a multiplicative approach cannot be directly used in case of negative trust values. Refer to the bibliographic notes for pointers on propagation methods that are designed for distrust relations.

2. *Trust aggregation across multiple paths:* In trust aggregation, the propagated values over various paths are aggregated into a single value. Common aggregation operators include the use of the minimum, maximum, average, weighted average, or weighted sum. In weighted averages, some propagation paths are considered more important than others. For example, shorter paths or recommendations from closer friends might be considered more important. Such weighting can also be handled within the trust propagation operator with the use of decay function.

   Consider the example of Figure 11.2. The use of the average operator in the aforementioned example leads to an estimated trust value between Alice and Bob of $(0.42 + 0.12)/2 = 0.27$, whereas the summation operator leads to an estimated value of $(0.42 + 0.12) = 0.54$. The bibliographic notes contain pointers related to various trust aggregation methods.

Trust propagation and aggregation are unsupervised methods for performing recommendations in trust-centric systems because they use fixed heuristics, irrespective of the underlying data. Supervised methods use low rank representations, such as matrix factorization, to learn these dependencies. In a later section, we will also discuss supervised methods, in which the algorithms learns the importance of different paths. It is noteworthy that some of the decay-based propagation algorithms with summation-based aggregation are very similar to the unsupervised Katz measure used in link prediction. The use of the Katz measure for link prediction is discussed in Chapter 10. As we will see later in section 11.3.7, the problem of trust-aware recommendations can be directly transformed to an instance of the link prediction problem.

11.3.3 Simple Recommender with No Trust Propagation

Consider a scenario in which a trust network is available, but only directly observed trust values (e.g., by user feedback in Epinions) are used. Furthermore, propagation and aggregation are not used to infer trust values between indirectly connected users. In other words, if user i has not *directly* provided feedback about user j, then no trust value between i and j is available. We have an $m \times n$ ratings matrix $R = [r_{ij}]$ for m users and n items, and an $m \times m$ trust matrix $T = [t_{ij}]$ representing the trust relationships. In other words, t_{ij} represents the degree to which user i trusts user j.

A simple approach to predict the rating $\hat{r}_{ij}$ of user i for item j is to define the peer group of user i as all users $N(i, \theta)$ who have rated the item j and are trusted by user i above a given threshold θ. Then, we can use the formula that is commonly used in neighborhood-based methods:

$$\hat{r}_{ij} = \frac{\sum_{k \in N(i,\theta)} t_{ik} r_{kj}}{\sum_{k \in N(i,\theta)} t_{ik}} \tag{11.3}$$

This approach can be viewed as a user-based version of neighborhood methods, in which trust values are used instead of the Pearson correlation coefficient. The formula is also referred to as the *trust weighted mean*. An alternative is to use the mean rating μ_k of each user k for centering the ratings, as in traditional collaborative filtering:

$$\hat{r}_{ij} = \mu_i + \frac{\sum_{k \in N(i,\theta)} t_{ik}(r_{kj} - \mu_k)}{\sum_{k \in N(i,\theta)} t_{ik}} \tag{11.4}$$

This approach can lead to predictions that do not lie within the specified rating scale. In such cases, one can adjust the rating to the nearest rating within the specified scale.

11.3.4 TidalTrust Algorithm

The *TidalTrust* algorithm is based on the observation that shorter paths are more reliable for propagation. Therefore, one should use the shortest path between a source-sink pair for the trust computation. For the purpose of further discussion, assume that the trust needs to be computed from source i to sink j. The algorithm derives its name from the fact that it first has a forward phase in which nodes are explored from source i to sink j in breadth-first order in order to discover all the shortest paths from i to j and also set a trust threshold $\beta(i, j)$. Then, the algorithm uses a backward phase in which recursive trust computations are made in reverse order from that in which nodes were explored in the forward phase (i.e., from sink to source). Only edges lying on the shortest paths (discovered in the forward phase) with trust at least equal to $\beta(i, j)$ are used in the backward phase. Therefore, the algorithm may be summarized as follows:

1. *Forward phase:* The goal of the forward phase is to determine a minimum threshold $\beta(i, j)$ on the trust values for them to be considered relevant in the trust computation between source i and sink j. The approach for computing $\beta(i, j)$ will be discussed later. Furthermore, all shortest paths from source to sink are determined during this phase with breadth-first search. Note that the subgraph $\mathcal{G}(i, j)$ of (all) shortest paths from source i to sink j is always a directed acyclic graph with no cycles. The *children* $C(q)$ of each node q are defined as all nodes to which node q points in this directed acyclic graph $\mathcal{G}(i, j)$ of shortest paths. Only the edges in this subgraph are relevant for the backward phase. The forward phase will be described in more detail later.

2. *Backward phase:* In the backward phase, starting from the sink node j, nodes are processed in reverse order of their distance to source node s using the edges of $\mathcal{G}(i,j)$. In other words, nodes closest to the sink are processed first. Let the currently processed node be denoted by q. If the edge (q,j) is already present in the trust network, then we can trivially set the predicted trust value $\hat{t}_{qj}$ to the observed trust value t_{qj}. Otherwise, if the edge (q,j) is not present in the trust network, then the predicted trust value $\hat{t}_{qj}$ between user node q and sink node j is recursively computed using only the edges in $\mathcal{G}(i,j)$ with observed trust values of at least $\beta(i,j)$:

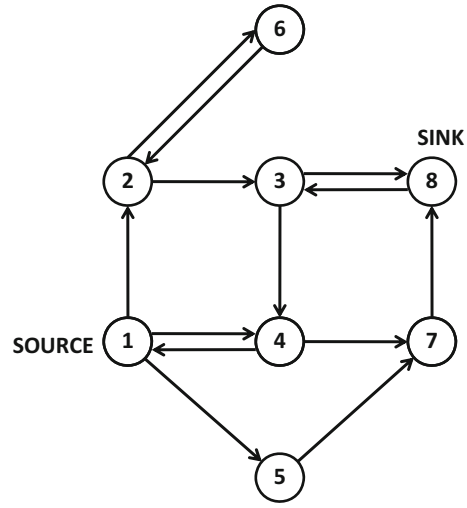
$$\hat{t}_{qj} = \frac{\sum_{k \in C(q), t_{qk} \geq \beta(i,j)} t_{qk} \hat{t}_{kj}}{\sum_{k \in C(q), t_{qk} \geq \beta(i,j)} t_{qk}} \tag{11.5}$$

It is noteworthy that computing the trust value $\hat{t}_{qj}$ according to Equation 11.5 always requires the computed values $\hat{t}_{kj}$ for all children $k \in C(q)$. The value of $\hat{t}_{kj}$ is always available at the time of computing the trust value $\hat{t}_{qj}$ because k is a child of q, and all computations are performed in the backward direction. Even though the approach computes many intermediate values $\hat{t}_{kj}$, the computed source-sink value $\hat{t}_{ij}$ is the only relevant one for a particular source-sink pair (i,j) and the other intermediate values are discarded. The approach, therefore, needs to be repeated over various source-sink pairs.

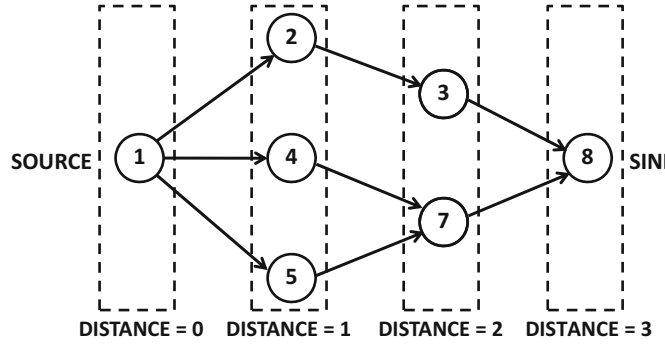
It remains to describe the forward phase in more detail. In the forward phase, a modified version of breadth-first search is used starting from node i in order to compute the directed acyclic graph of shortest paths $\mathcal{G}(i,j)$. Standard breadth-first search only discovers the first shortest path between i and j (depending on node exploration order), whereas we would like to find all of them. The main difference from standard breadth-first search is that previously visited neighbors of a node are also checked to see if they might be children of a given node. The source i is labeled with a distance value $d(i)$ of 0. All other distances are labeled as ∞. All outgoing neighbors of i are then labeled with a distance value of 1 and added to a list L. In each iteration, the node q with the smallest distance label $d(q)$ from L is selected. The labels of each its neighbors k on outgoing edges are modified as follows:

$$d(k) = \min\{d(k), d(q) + 1\} \tag{11.6}$$

The node k is added to the children $C(q)$ of q if and only if $d(k) = d(q)+1$ after the update. The node q is deleted from the list L after updating the labels of all its neighbors (including those that were visited earlier). The algorithm terminates when the node with the smallest distance label in L is the sink j. At this point, all nodes in the graph with distance labels greater than or equal to that of the sink node j are deleted from the network. Furthermore, any edge (q,k) not satisfying the condition $d(k) = d(q) + 1$ is deleted. The remaining subgraph $\mathcal{G}(i,j)$ contains all the shortest paths from node i to node j. For example, the shortest path subgraph $\mathcal{G}(i,j)$ for the trust network in Figure 11.3(a) is shown in Figure 11.3(b). Note that node 6 is missing in Figure 11.3(b) because it is irrelevant to any of the paths between the source node 1 and sink node 8. Several edges have also been dropped from the original graph because they do not lie on any shortest path. For each source to sink path in $\mathcal{G}(i,j)$, the minimum weight edge is determined. The value of $\beta(i,j)$ is set to the maximum of these various minima. A dynamic programming approach can also be used to efficiently compute $\beta(i,j)$ by keeping track of intermediate values of $\beta(i,k)$ during the forward phase. We initialize $\beta(i,i) = \infty$ and $\beta(i,k) = 0$ for each $k \neq i$. Whenever, the label



(a) A trust network



(b) Subgraph of shortest paths

Figure 11.3: The subgraph of shortest paths found by *TidalTrust* for a trust network

of node k strictly reduces because of incoming edge (q, k) (based on Equation 11.6), the following update is also executed:

$$\beta(i,k) = \max\{\beta(i,k), \min\{t_{qk}, \beta(i,q)\}\} \tag{11.7}$$

As a result, the end of the forward phase also yields the value of $\beta(i, j)$.

So far, we have only discussed user-to-user trust computation in *TidalTrust*. How can this computation help in recommendation of *items*? The final rating of an item is computed using the trust-weighted mean, in a manner similar to Equation 11.3. The main difference is that predicted trust values $\hat{t}_{ik}$ can also be used on the right-hand side of Equation 11.3, rather than only the observed trust values of the neighbors of node i. Let I_i be the indices of the items rated by users i. Therefore, Equation 11.3 is modified as follows:

$$\hat{r}_{ij} = \frac{\sum_{k:k\in I_i \hat{t}_{ik}\geq\theta} \hat{t}_{ik} r_{kj}}{\sum_{k:k\in I_i \hat{t}_{ik}\geq\theta} \hat{t}_{ik}} \tag{11.8}$$

As before, θ is a user-defined threshold on the inferred trust value, for it to be used in the computation. These methods have a particularly beneficial effect on recommendations to controversial users whose ratings on items differ significantly from other users [223].

11.3.5 MoleTrust Algorithm

The *MoleTrust* algorithm shares a number of conceptual similarities with the *TidalTrust* algorithm, but is quite different in terms of how it is implemented. The *TidalTrust* algorithm uses a forward phase followed by a backward phase for each source-sink pair, whereas the *MoleTrust* algorithm uses two forward phases for each source node. Note that a single application of the forward and backward phase in the *TidalTrust* algorithm is able to compute the trust from a particular source to a particular sink, whereas the *MoleTrust* algorithm is able to compute the trust from the source i to *all* other nodes within a maximum distance threshold in two forward phases. As a sink is not specified in *MoleTrust*, a different criterion (in terms of maximum path length δ) is used to terminate the shortest path computation. Furthermore, a user-defined trust threshold α is used across all source-sink pairs, rather than one that is computed for each source-sink pair. Therefore, the two phases are as follows:

1. *Forward phase 1:* Determine all shortest paths starting from source node i with length at most δ. As in *TidalTrust*, the modified breadth-first approach is used, except that the termination criterion is based on maximum path length rather than on reaching the sink node. We determine the directed acyclic graph $\mathcal{G}(i, \delta)$ in which all edges lie on one of these shortest paths. The *predecessors* $P(q)$ of each node are the nodes that point to q in the graph $\mathcal{G}(i, \delta)$. Note that the notion of predecessor in *MoleTrust* is exactly the converse of the notion of children in *TidalTrust*.

2. *Forward phase 2:* The algorithm starts by setting $\hat{t}_{ik} = t_{ik}$ for all nodes k such that the edge (i, k) is present in the graph $\mathcal{G}(i, \delta)$. These represent nodes at distance 1 from source node i. Then, the trust value between the source and the nodes at higher distances are computed. For any node q at a distance 2 or more from the source node i in $\mathcal{G}(i, \delta)$, the trust $\hat{t}_{iq}$ is computed as follows:

$$\hat{t}_{iq} = \frac{\sum_{k \in P(q), t_{kq} \geq \alpha} \hat{t}_{ik} \cdot t_{kq}}{\sum_{k \in P(q), t_{kq} \geq \alpha} t_{kq}} \tag{11.9}$$

   Note the similarity with the *TidalTrust* computation. The main difference is that this computation is in the forward direction and the threshold α is user-defined. and it is invariant across all source-sink pairs. Unlike *TidalTrust*, in which a *source-sink specific* threshold $\beta(\cdot, \cdot)$ (computed during the forward phase) is used, the threshold α of the *MoleTrust* algorithm is invariant across all source-sink pairs.

The final approach for item recommendation is similar to that of *TidalTrust*. After all the trust values have been computed, one can use Equation 11.8 to make ratings predictions.

The directed acyclic subgraph for a maximum horizon of length 2, for the graph of Figure 11.3(a) is shown in Figure 11.4. As in the case of Figure 11.3, node 1 is used as the source node. Note that unlike Figure 11.3(b), node 6 is present in Figure 11.4, but node 8 is absent. In the *TidalTrust* algorithm, the trust values of nodes that are outside the distance horizon of the source node cannot be the computed. Therefore, $\hat{t}_{18}$ cannot be computed by *MoleTrust*. The assumption is that the trust computation $\hat{t}_{18}$ is too unreliable to be used in the recommendation process. Therefore, such trust values are implicitly set to 0. *MoleTrust* is more efficient than *TidalTrust* because it needs only be applied once for each source node, rather than for each source-sink pair.

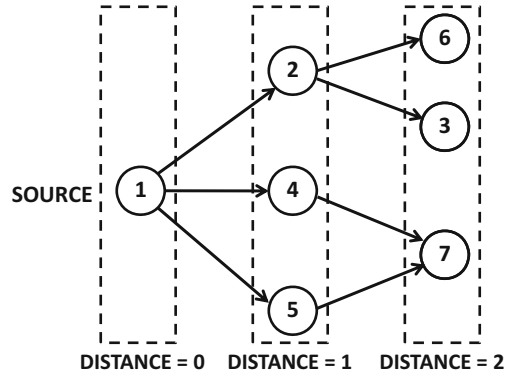


Figure 11.4: The subgraph of shortest paths found by *MoleTrust* at a maximum horizon of 2 for the trust network in Figure 11.3(a)

11.3.6 TrustWalker Algorithm

The TrustWalker algorithm [269] is based on the observation that social network links provide an independent source of information from ratings [172]. Therefore, a random-walk approach is used to discover the similar users. However, a major dilemma is that if one goes too far in the random walk, then irrelevant users might be used. An important observation in this context is that the ratings of strongly trusted friends on similar items are better predictors than the ratings of weakly trusted friends on the same item. Therefore, the *TrustWalker* approach combines trust-based user similarity and item-based collaborative filtering models in a unified random-walk framework.

The *TrustWalker* algorithm uses a random-walk approach on the social network of users. The algorithm starts with the source user i in order to determine the rating $\hat{r}_{ij}$ for item j. At each step of the random walk, it checks whether the visited user k in the random walk has rated item j. If this is indeed the case, then the observed rating r_{kj} is returned. Otherwise, the algorithm has two choices, which can be viewed as a modified version of the restart method in random walks:

1. At step l of the random walk, the algorithm can terminate at node k with probability ϕ_{kjl}. In such a case, the rating of the user k on a random item similar to j is returned. Among all items rated by user k, this random item is chosen with a probability proportional to its item-item similarity with target item j. Note that the returned rating can be viewed as a randomized and trust-based version of item-based collaborative filtering algorithms.

2. With probability $(1 - \phi_{kjl})$, the random walk is continued to the neighbors of k.

The random walk is repeated multiple times, and the ratings are averaged in a probabilistic way over the various walks. This weighting is based on the probability of termination at various random walks and the probability of selecting the specific item used for making the prediction. Refer to [269] for details.

It is noteworthy that the restart probability ϕ_{kjl} is dependent on the currently visited user k, item j, and number of steps l. The intuition for deciding this value is as follows. The value of the termination probability ϕ_{kjl} increases with the number of steps l to avoid the use of weakly trusted users that are far away from the source user. This is consistent with

all trust-based algorithms that avoid using longer paths for trust propagation. Furthermore, the probability of termination should also be high, if we are confident that similar items rated by k will provide a reliable prediction. This is achieved by increasing the probability of termination when the similarity value of target item j to the closest item rated by k is high. Let this maximum similarity value be $\Delta_{kj} \in (0, 1)$. Therefore, the overall probability of termination is set as follows:

$$\phi_{kjl} = \frac{\Delta_{kj}}{1 + \exp(-l/2)} \tag{11.10}$$

The aforementioned computation requires the determination of item-item similarity. To compute the similarities between two items, a discounted version of the Pearson correlation coefficient is used. First, only items with positive correlation are considered. Second, the discount factor is set in such a way that the similarity value is reduced when the number of common users rating the items is small. Therefore, for two items j and s with N_{js} raters in common, we have:

$$\text{Sim}(j, s) = \frac{\text{Pearson}(j, s)}{1 + \exp(-N_{js}/2)} \tag{11.11}$$

Therefore, the *TrustWalker* algorithm is able to combine the notions of user trust and item-item similarity in a seamless way, within a single random-walk framework.

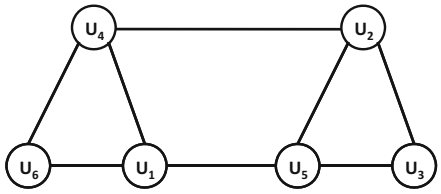
11.3.7 Link Prediction Methods

Most of the aforementioned methods are designed to work with trust propagation and aggregation *heuristics*. The effectiveness of a particular heuristic might depend on the data set at hand. This is because such methods are *unsupervised*, and they do not always adapt well to the particular structure of the network at hand. A natural question arises whether one can *directly* learn the relevance of different parts of the trust network in a data-driven manner while performing the propagation and aggregation. Link prediction methods are useful if only a ranked list of recommended items is required rather than the prediction of exact values of the ratings. This caveat is primarily because most link prediction methods are good at recommending ranked lists of edges, but do not work very well for predicting weights on the edges exactly.

As discussed in section 10.4.6 of Chapter 10, traditional collaborative filtering problems can be posed as link prediction problems on user-item graphs. Refer to sections 10.2.3.3 and 10.4.6 for a detailed discussion of how user-item graphs can be used for traditional collaborative filtering. A detailed discussion of the process of user-item graph construction is also provided in section 2.7 of Chapter 2. In this case, the user-item graphs need to be augmented with social links corresponding to the links between various users. The augmentation of user-item graphs with social links allows the use of social information in the collaborative filtering process.

Consider an $m \times n$ ratings matrix with m users and n items. It is assumed that the users are arranged in the form of a social network $G_s = (N_u, A_u)$. Here, N_u denotes the set of nodes representing users, and A_u denotes the set of social links between the users. A one-to-one correspondence exists between users and nodes in N_u. Since the number of users is m, we have $|N_u| = m$. An example of a toy social network between a set of users is illustrated in Figure 11.5(a).

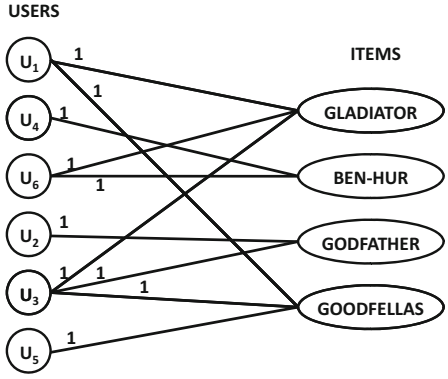
The user-item graph can be viewed as an augmentation of the social network graph with item nodes. Let N_i be the set of nodes representing items. As in the case of user nodes, a



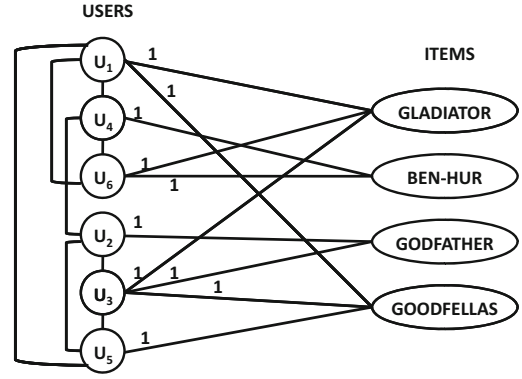
(a) A toy social network

	GLADIATOR	GODFATHER	BEN-HUR	GOODFELLAS
U_1	1			1
U_2		1		
U_3	1	1		1
U_4			1	
U_5				1
U_6	1		1	

(b) Unary ratings matrix



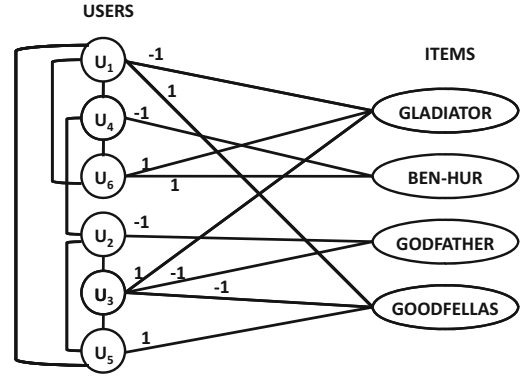
(c) User-Item graph (no social links)



(d) User-item graph (with social links)

	GLADIATOR	GODFATHER	BEN-HUR	GOODFELLAS
U_1	-1			1
U_2		-1		
U_3	1	-1		-1
U_4			-1	
U_5				1
U_6	1		1	

(e) Binary ratings matrix



(f) Signed user-item graph with social links

Figure 11.5: A toy example showing the amalgamation of social links and user-item graphs

one-to-one correspondence exists between items and nodes in N_i. Since we have n items, we have $|N_i| = n$. We construct the graph $G = (N_u \cup N_i, A_u \cup A)$. Here, A is a set of edges that exist between user nodes in N_u and item nodes in N_i. Note that the nodes and edges of this graph are supersets of those in the original social network G_s. The edges in A correspond to the relationships in the user-item graph (cf. section 2.7 of Chapter 2). Specifically, an edge exists between a user node in N_u and an item node in N_i, if the user has rated that item. The weight on that edge is equal to the mean-centered rating of the user for that item. This will often result in negative weights on the edges. In the case of implicit feedback data sets, the feedback is not mean-centered but the corresponding weight (e.g., a 0-1 value or the number of items bought) is used. The reason for mean-centering in the former case is that ratings are supposed to indicate both likes and dislikes, whereas implicit feedback provides a form of unary rating with no explicit mechanism to specify a dislike. In the case of implicit feedback, the resulting network is a conventional network with only non-negative weights on the links. In the case of explicit feedback, the resulting network is a *signed* network with positive and negative edge weights. It is noteworthy that the resulting network can be viewed as a union of the nodes and edges in the original social network, and the user-item graphs discussed in section 2.7 of Chapter 2.

In order to illustrate this point, we show an example of a unary ratings matrix in Figure 11.5(b). This matrix is the same as that shown in Figure 10.11(a) of Chapter 10. The corresponding user-item graph (without social connections) is illustrated in Figure 11.5(c). This graph is identical to that in Figure 10.11(b) of Chapter 10. The user-item graph with social connections is illustrated in Figure 11.5(d). Note that the graph in Figure 11.5(d) is a union of the graphs in Figures 11.5(a) and (c). Furthermore, social links can also have weights depending on the strength of the social ties or the level of trust between the corresponding social actors. As discussed in section 10.4.6, link prediction methods can be used to determine user affinity for items. Most link prediction methods also return a quantification of the strength of the predicted link. The strengths of predicted links of users for items can be ranked in order to create a ranked item list for the user. Link prediction methods are discussed in section 10.4 of Chapter 10. In the case of implicit ratings, conventional link prediction methods may be used because all link weights are non-negative. The only difference from the approach in section 10.4.6 is that the user-item graphs are enhanced with social connections. One challenging issue with the use of this approach is that the social links and the user-item links may not be equally important for the particular application at hand. In order to address this issue, the weights of all the social links are multiplied with the parameter λ. The value of λ regulates the relative importance between social (trust) links and user-item links. The optimal value of λ is chosen using cross-validation in order to maximize the prediction accuracy.

For explicit feedback, the ratings need to be mean-centered, which will result in edges with signed weights. In the special case of binary ratings, the values of -1 and $+1$ are used to retain simplicity. A value of $+1$ indicates a "like," whereas a value of -1 indicates a "dislike." An example of a binary ratings matrix is shown in Figure 11.5(e), and its socially augmented user-item graph is shown in Figure 11.5(f). For such problems, *signed* link prediction methods [346, 591] can be used to predict both like and dislike ratings. Furthermore, it is also possible to include distrust relationships in these predictions by using negative social links.

One of the nice aspects of link prediction methods is that they do not require the explicit use of trust propagation and aggregation heuristics because the transitivity of user trust and corresponding preferences is already learned in a data-driven manner with the use of machine learning algorithms. In fact, one can even use link prediction methods to infer the trust

values between pairs of users in the social network, rather than directly inferring user-item affinities. In other words, the machine learning techniques of link prediction can automatically propagate and aggregate trust in a data-driven manner. It is particularly helpful to use *supervised* methods (cf. section 10.4.4 of Chapter 10) for link prediction because such methods can learn the importance of the trust network in a data-driven manner. In fact, many of the trust propagation methods can be viewed as *unsupervised* heuristics, whereas link prediction provides a route for incorporating supervision into the computation. Indeed, unsupervised measures for link prediction, such as the Katz measure (cf. section 10.4.2 of Chapter 10), are very similar in principle to some of the decay-based trust propagation heuristics. It is well known [355] that supervised methods for link prediction generally outperform unsupervised methods.

Many link prediction methods are designed for undirected networks. While we have assumed an undirected trust network in the aforementioned example for simplicity, it could very easily have been used in a directed manner. In the aforementioned user-item graph model, the user-user links can be assumed to be asymmetric and directed according to the specified trust relationships, whereas the user-item links are always directed from users to items. Therefore, directed paths from users to items imply trust-based affinities from users to items. Either supervised methods or matrix factorization methods [432] can be used to perform directed link prediction. Therefore, link prediction methods provide a very general framework that can be used in a variety of scenarios. The bibliographic notes contain pointers to some recent methods that have use link prediction for recommendations.

11.3.8 Matrix Factorization Methods

Matrix factorization methods are closely related to link prediction [432]. Although one can use matrix factorization methods within the link prediction framework of the previous section by using the approach in [432] as the base algorithm, it is more fruitful to design and optimize matrix factorization methods directly for trust networks.

Let R be an $m \times n$ ratings matrix with m users and n items. Let us assume that the social trust matrix is given by an $m \times m$ matrix $T = [t_{ip}]$. Note that both R and T are incomplete matrices that are highly sparse. Let S_R and S_T be the observed indices in these matrices:

$$S_R = \{(i, j) : r_{ij} \text{ is observed}\}$$
$$S_T = \{(i, p) : t_{ip} \text{ is observed}\}$$

In cases where all observed values of t_{ip} are strictly positive, it is helpful to set a sample of the unobserved values of t_{ip} to 0, and include the corresponding indices within S_T. Such an approach can help in avoiding overfitting because it compensates for the lack of negative feedback (cf. section 3.6.6.2 of Chapter 3).

We start by introducing the *SoRec* algorithm. The *SoRec* algorithm [381] can be viewed as an extension of the matrix factorization methods in Chapter 3 to include social information. We emphasize that the presentation here is a simplified version of the *SoRec* algorithm, which is originally presented as a probabilistic factorization algorithm. The simplified presentation helps in understanding the key ideas behind the algorithm, by abstracting out the less important but complex details. Readers are referred to [381] for the exact description.

In Chapter 3, a matrix factorization model of rank-k is proposed to create an $m \times k$ user-factor matrix $U = [u_{ij}]$ and an $n \times k$ item-factor matrix $V = [v_{ij}]$, so that the following condition is satisfied as closely as possible over the observed entries:

$$R \approx UV^T \tag{11.12}$$

In order to incorporate social information, we introduce a second $m \times k$ user factor matrix $Z = [z_{ij}]$, so that the following condition is satisfied as closely as possible over the observed trust values:

$$T \approx UZ^T \tag{11.13}$$

Two user-factor matrices are employed here because matrix U is for the initiator and matrix Z is for the receiver. Furthermore, as T might not be symmetric, U and Z need not be the same. Intuitively, the initiator is the participant who decides whether or trust or not to trust (i.e., source), and the receiver is the participant who is the recipient of this trust/distrust (i.e., sink). Note that the user matrix U, which is the initiator, is shared in both factorizations. The initiator is shared rather than the receiver because the trust opinions of sources for sinks are used to predict ratings in such systems. It is this sharing of U that results in a factorization with the incorporation of the social trust information. Therefore, a joint factorization objective function is set up in which the errors in factorizing each of R and T are added. How much should the errors in each of the two factorizations be weighted? This is achieved with the use of a balance parameter β. Then, the overall objective function may be stated as follows:

$$\text{Minimize } J = \underbrace{||R - UV^T||^2}_{\text{Observed entries in } R} + \underbrace{\beta \cdot ||T - UZ^T||^2}_{\text{Observed entries in } T} + \underbrace{\lambda \left(||U||^2 + ||V||^2 + ||Z||^2\right)}_{\text{Regularizer}}$$

The parameter λ controls the level of regularization. Note that this objective function is computed only over the observed entries and the unspecified entries are ignored in the computation of the Frobenius norm. This is consistent with the approach used in Chapter 3. The resulting objective function is, therefore, a straightforward extension of the matrix factorization methods in Chapter 3 with an additive term for the social contribution. We rewrite the objective function in terms of observed entries in S_R and S_T:

$$\text{Min. } J = \underbrace{\sum_{(i,j) \in S_R} (r_{ij} - \sum_{s=1}^{k} u_{is} v_{js})^2}_{\text{Observed entries in } R} + \beta \underbrace{\sum_{(i,p) \in S_T} (t_{ip} - \sum_{s=1}^{k} u_{is} z_{ps})^2}_{\text{Observed entries in } T} + \underbrace{\lambda \left(||U||^2 + ||V||^2 + ||Z||^2\right)}_{\text{Regularizer}}$$

A gradient-descent approach can be used to determine the factor matrices U, V, and Z. A gradient vector of J with respect to all parameters in U, V, and Z is used to update the current vector of parameters representing all entries in U, V, and Z. The gradient descent steps depend on the errors $e_{ij}^{(r)}$ and $e_{ip}^{(t)}$ in the matrices between the observed and predicted values in the two matrices:

$$e_{ij}^{(r)} = r_{ij} - \hat{r}_{ij} = r_{ij} - \sum_{s=1}^{k} u_{is} v_{js}$$

$$e_{ip}^{(t)} = t_{ip} - \hat{t}_{ip} = t_{ip} - \sum_{s=1}^{k} u_{is} z_{ps}$$

The error matrix on the ratings can be written as $E_r = [e_{ij}^{(r)}]$ in which unobserved entries (i.e., entries not in S_R) are set to 0. The error matrix on the trust entries can be written as $E_t = [e_{ij}^{(t)}]$ in which unobserved entries (i.e., entries not in S_T) are set to 0. Then, the gradient-descent steps can be written in the form of matrix updates as follows:

$$U \Leftarrow U(1 - \alpha \cdot \lambda) + \alpha E_r V + \alpha \cdot \beta E_t Z$$
$$V \Leftarrow V(1 - \alpha \cdot \lambda) + \alpha E_r^T U$$
$$Z \Leftarrow Z(1 - \alpha \cdot \lambda) + \alpha \cdot \beta E_t^T U$$

Here, $\alpha > 0$ represents the step-size. The details of the derivation of the gradient-descent method are left as an exercise for the reader. Note that only the observed entries of E_r and E_t need to be computed in each iteration, and it makes sense to use a sparse data structure to represent these matrices because unobserved entries are set to 0. Although we have used a single regularization parameter λ and update step-size α for all updates, it often makes sense to use different regularization parameters and step-sizes for the different matrices U, V, and Z.

Next, we describe the *stochastic* gradient descent method in which the error is approximated in a randomized way with that over a single entry. This entry is selected in random order and might belong to either the ratings matrix or the trust matrix. Then, the stochastic gradient-descent approach first iterates through each observed entry $(i, j) \in S_R$ in the ratings matrix in random order and makes the following updates:

$$u_{iq} \Leftarrow u_{iq} + \alpha \left(e_{ij}^{(r)} \cdot v_{jq} - \frac{\lambda \cdot u_{iq}}{2 \cdot n_i^{user}} \right) \quad \forall q \in \{1 \ldots k\}$$
$$v_{jq} \Leftarrow v_{jq} + \alpha \left(e_{ij}^{(r)} \cdot u_{iq} - \frac{\lambda \cdot v_{jq}}{n_j^{item}} \right) \quad \forall q \in \{1 \ldots k\}$$

Here, $\alpha > 0$ represents the step-size. Furthermore, n_i^{user} represents the number of observed ratings for user i, and n_j^{item} represents the number of observed ratings for item j. Note that this set of updates is identical to that used in matrix factorization for collaborative filtering without the trust matrix (cf. section 3.6.4.2 of Chapter 3). One difference is that we have respectively normalized[1] the two regularization components with the number of observed ratings for users and items.

Subsequently, the stochastic gradient-descent approach iterates through each observed entry $(i, p) \in S_T$ in the trust matrix in random order and makes the following update steps:

$$u_{iq} \Leftarrow u_{iq} + \alpha \left(\beta \cdot e_{ip}^{(t)} \cdot z_{pq} - \frac{\lambda \cdot u_{iq}}{2 \cdot n_i^{out}} \right) \quad \forall q \in \{1 \ldots k\}$$
$$z_{pq} \Leftarrow z_{pq} + \alpha \left(\beta \cdot e_{ip}^{(t)} \cdot u_{iq} - \frac{\lambda \cdot z_{pq}}{n_p^{in}} \right) \quad \forall q \in \{1 \ldots k\}$$

Here n_i^{out} denotes the number of observed entries in S_T, for which i is the origin of the edge, and n_p^{in} denotes the number of observed entries in S_T for which p is the destination of the edge. We alternately cycle through the observed entries in the ratings matrix and the trust matrix with these updates, until convergence is reached. During a particular

[1]Strictly speaking, such a normalization should also be used in traditional matrix factorization, but it is often omitted on a heuristic basis. In the particular case of trust-centric systems, normalization becomes more important because of the varying sizes of the ratings matrix and trust matrix.

cycle, the entries are processed in random order, corresponding to the "stochastic" nature of this gradient-descent approach. The parameters β and λ can be selected using cross-validation or by simply trying various values of these parameters on a hold-out set. Different regularization parameters can be used for different matrices for better results, although such an approach increases the complexity of parameter tuning.

As in the case of all matrix factorization methods, the ratings matrix can be reconstructed as $\hat{R} = UV^T$. Note that one can also perform full reconstruction of the trust matrix T as $\hat{T} = UZ^T$. In fact, the reconstruction of the trust matrix can be viewed as a data-driven method for trust propagation and aggregation in which the rating information is used in addition to the existing trust relations.

11.3.8.1 Enhancements with Logistic Function

The aforementioned description provides a simplified version of *SoRec* in order to align it more closely with the discussion in Chapter 3. The actual *SoRec* algorithm uses a somewhat more sophisticated objective function. Matrix factorization methods have the drawback of predicting values that are outside the range of item ratings in R or trust values in T. One way of forcing the factorization to produce range-bound ratings is to use a logistic function $g(x) = 1/(1 + \exp(-x))$ within the factorization. The logistic function always maps values to the range $(0, 1)$. Without loss of generality, the ratings in R and trust values in T can be assumed[2] to be drawn from the range $(0, 1)$. In other words, the ratings R and trust matrix T should be reconstructed as $R \approx g(UV^T)$ and $T \approx g(UZ^T)$. The expression $g(UV^T)$ denotes the fact that the function $g(\cdot)$ is applied to each matrix element of UV^T. Then, the aforementioned objective function also needs to be modified as follows:

$$\text{Minimize } J = \underbrace{||R - g(UV^T)||^2}_{\text{Observed entries in } R} + \underbrace{\beta \cdot ||T - g(UZ^T)||^2}_{\text{Observed entries in } T} + \underbrace{\lambda \left(||U||^2 + ||V||^2 + ||Z||^2\right)}_{\text{Regularizer}}$$

Note the use of the logistic function within the objective function. Correspondingly, the gradient descent method will multiplicatively incorporate the derivative of the logistic function in one of its terms. It is noteworthy that the logistic function-based enhancement is really an optimization and can be used within the context of any matrix factorization method in Chapter 3, not just in trust-based methods.

11.3.8.2 Variations in the Social Trust Component

Many variations of the aforementioned matrix factorization method exist, especially in terms of how the social (trust) part of the objective function is formulated.

1. Instead of using an $m \times k$ social factor matrix Z to impose $T \approx UZ^T$, one might use a $k \times k$ matrix H to impose $T \approx UHU^T$. The corresponding social term in the objective function will be modified to $||T - UHU^T||^2$. Intuitively, the matrix H captures the pairwise correlations between the various latent components of the users. The approach is referred to as *LOCALBAL* [594]. It is also possible to use a logistic function within the objective function as in the case of *SoRec*, although the original work does not use this approach.

   Note that this approach has a similar form as *SoRec*, except that it parameterizes $Z = UH^T$. The matrix H has only k^2 variables, whereas Z has $m \cdot k$ variables. Therefore,

[2] Ratings do not always lie in $(0, 1)$. If needed, the ratings matrix can be scaled using its ranges to $(r_{ij} - r_{min})/(r_{max} - r_{min})$, so that all its entries lie in $(0, 1)$.

LOCALBAL makes stronger assumptions about the social correlation structure of the users, as compared to *SoRec*. Fewer variables reduce the likelihood of overfitting, at the expense of incorporating some bias.

2. The *SocialMF* [270] algorithm imposes the constraint $U \approx TU$. Note that TU is not defined because some of the entries in T might be unspecified. Such entries are set to 0 for the purpose of computing TU. The corresponding social term in the objective function is $||U - TU||^2$. It is assumed that each row of T is normalized to sum to 1. The logistic function is used only within the term $||R - g(UV^T)||^2$ involving the ratings matrix. Note that the number of factor variables in this case is even fewer because the matrix Z is missing. In fact, the number of factor variables is exactly the same as in conventional matrix factorization. Reducing the number of factor variables will help in avoiding overfitting, but comes at the expense of greater bias.

   The approach sets the preference vector of each user to her trust-weighted average preference vector over all her neighbors. This is a direct result of the normalization of each row of T to sum to 1. The basic assumption is that the behavior of a user is affected by the behavior of her direct neighbors due to social influence.

3. *Social regularization:* In this approach [382], the user factors are forced to be more similar across links, and the difference in similarity is weighted with the trust values in the objective function. In other words, if $\overline{u_i}$ is the ith row of U, then the social part of the objective function is $\sum_{(i,j):t_{ij}>0} t_{ij}||\overline{u_i} - \overline{u_j}||^2$. This approach can be viewed as an indirect way of forcing homophily, and works best with implicitly inferred trust values t_{ij}. An example of such an implicitly inferred trust value is provided in Equation 11.2. Many variations of this approach, such as average-based regularization, are also discussed in the same work. The average-based regularization approach is somewhat similar to the *SocialMF* algorithm.

The bibliographic notes also provide pointers related to several other variations of the basic objective function.

11.3.9 Merits of Social Recommender Systems

Social recommender systems have a number of merits because they incorporate additional trust information into the recommendation process. This is particularly useful for improving the recommendation quality of items, addressing cold-start issues, and making the approach attack-resistant.

11.3.9.1 Recommendations for Controversial Users and Items

The greatest advantages in incorporating trust lie in the improvement of the recommendation quality for controversial users and items. Controversial users are those that disagree with the other users about the ratings of specific items [223]. Controversial items are those that receive diverse or polarized reviews. In such cases, the use of trust metrics generally enhances the user-specific or item-specific accuracy significantly [223, 406, 617] because the opinions of users are highly personalized in such cases. For example, users who are more similar and trust one another are more likely to provide similar ratings for controversial items.

11.3.9.2 Usefulness for Cold-Start

Social links are particularly useful to handle the cold-start problem for new *users*. Consider the case where a link prediction system is used for recommendation. A new user comes into the system and therefore has no ratings associated with any of the items covered by the recommender system; no user-item links are incident on that user. On the other hand, if social links are incident on that user, the link prediction method can still be used in order to predict the top matching items. This observation is true for other recommendation methods, such as matrix factorization. The main assumption is that the social links for the users are often available even before the user actively starts using the system. This is especially true for implicitly inferred trust networks. In any case, social links do add more data, which is helpful for alleviating the sparsity issue in the recommendations.

11.3.9.3 Attack Resistance

In general, there are significant commercial motivations for merchants to try to "cheat" recommender systems hosted by third parties. For example, the manufacturer of an item might try to post fake reviews for his item on Amazon.com. In many cases, such reviews are posted with the use of fake profiles created by the manufacturer. Trust-based recommender systems are more resistant to such attacks because these algorithms rely on the trustworthy peers of a user for predicting ratings. For example, Equations 11.3 and 11.4 explicitly weight the trust of a user for other users in the prediction process. A user is highly unlikely to specify a trust relation with a fake profile. As a result, such an approach is less likely to use the ratings posted from fake profiles in the prediction process. The topic of attack-resistant recommendation systems will be discussed in more detail in the next chapter.

11.4 User Interaction in Social Recommenders

The next-generation Web, which is also referred to as Web 2.0, has supported the development of a number of open systems in which users can actively participate and leave feedback. In particular, the development of *social tagging systems* allows the user to create and share meta-data about media objects. Such meta-data are also referred to as *tags*. Users may tag any form of object supported by the social network, such as an image, a document, music, or video. Virtually all social media sites allow some form of tagging. Some examples of such tagging systems are as follows:

- Flickr [700] allows users to tag images with keywords. For example, a keyword might describe the scenery or object in a specific image.
- The site last.fm [692] hosts music and allows users to tag music.
- Delicious [702] promotes the sharing of bookmarks and online links.
- The Bibsonomy [256, 708] system allows the sharing and tagging of publications.
- For a while, Amazon.com allowed its customers to tag products [709].

It is instructive to examine the nature of the tags created by a social tagging site such as last.fm. For the well-known musical album *Thriller* by Michael Jackson, the top tags at last.fm are as follows:

1001 albums you must hear before you die, 1980s, 1982, 1983, 80s pop, albums, albums I own, albums I own on vinyl, beat it, classic, classic pop, classic rock, crates of vinyl, dance-pop, disco, epic, thirller ...

These tags are quite informal because they are created by users in an open and participatory environment, rather than by specialists. Note that the tag "thirller" is a misspelling, which is quite common in such settings. Furthermore, all songs that were tagged by various users with a particular tag are indexed by it. For example, by clicking on the tag "classic rock," one can access various resources (artists, albums, or events) related to this tag because the corresponding albums and songs were tagged by various users. In other words, the tag "classic rock" serves as a bookmark or index to other relevant resources.

This process, therefore, results in the organization of content and the creation of a knowledge resource referred to as a *folksonomy*. The term "folksonomy" derives its roots from "folk" and "taxonomy," and it therefore intuitively refers to the classification of Web objects by non-specialist, voluntary, participants on the World Wide Web (i.e., common folk). This term was coined by Thomas Vander Wal, who defined it as follows [707]:

> "Folksonomy is the result of personal free tagging of information and objects (anything with a URL) for one's own retrieval. The tagging is done in a social environment (usually shared and open to others). Folksonomy is created from the act of tagging by the person consuming the information.
>
> The value in this external tagging is derived from people using their own vocabulary and adding explicit meaning, which may come from inferred understanding of the information/object. People are not so much categorizing, as providing a means to connect items (placing hooks) to provide their meaning in their own understanding."

Other terms used to describe social tagging include *collaborative tagging*, *social classification*, and *social indexing*. Tags provide an understanding of the topic of the object, and they often use vocabulary that are commonly used and understood by other participants. Therefore, the non-specialist nature of the participants is actually an asset, and it contributes to the collaborative power of such a system. Tags are also referred to as *social indexes* because they serve the dual role of organizing items. For example, by clicking on a tag, a user might be able to browse items related to that tag.

Folksonomies have numerous applications, including recommender systems [237]. In the particular context of recommender systems, folksonomies are valuable because they contribute to the available knowledge about the object at hand. At the very least, each tag can be considered a feature describing an object, although the underlying description may sometimes be noisy and irrelevant. In spite of their noisy nature, it has been observed that such social tagging methods can be used to improve the effectiveness of recommender systems significantly by complementing the knowledge available in ratings and other sources.

11.4.1 Representing Folksonomies

In tagging systems, *users* annotate *items* (or *resources*) with *tags*. The nature of the resource depends on the underlying system at hand. For example, the resource might be an image for Flickr, or a song for last.fm. Therefore, a 3-way relationship exists between users, items, and tags. Correspondingly, it can be represented as a hypergraph, in which each hyper-edge connects three objects. One can also represent it as a 3-dimensional cube (or tensor) containing unary bits with information about whether a user has tagged a specific

resource (e.g., image) with a particular tag (e.g., "landscape"). The action of a user tagging a resource sets the corresponding bit to 1, and it is unspecified otherwise. In many cases, the unspecified values are approximated as 0 for analytical purposes. Figure 11.6 shows a toy example of 6 users with 4 items (images) and 5 tags both in the hypergraph and in the tensor representation. Figure 11.6(a) shows the hypergraph representation, whereas the Figure 11.6(b) shows the hypergraph representation. For example, Ann has tagged item 2 with the tag "flower." This results in a hyper-edge between these three entities in Figure 11.6(a), whereas the corresponding bit is set to 1 in Figure 11.6(b). Formally, we define a folksonomy as follows:

Definition 11.4.1 (Folksonomy) *A folksonomy is defined over m users, n items, and p tags as a 3-dimensional array $F = [f_{ijk}]$ of size $m \times n \times p$. The element f_{ijk} is a unary value indicating whether user i has tagged item j with the kth tag. In other words, the value of f_{ijk} is defined as follows:*

$$f_{ijk} = \begin{cases} 1 & \text{if user } i \text{ has tagged the } j\text{th resource with the } k\text{th tag} \\ \text{unspecified} & \text{otherwise} \end{cases} \tag{11.14}$$

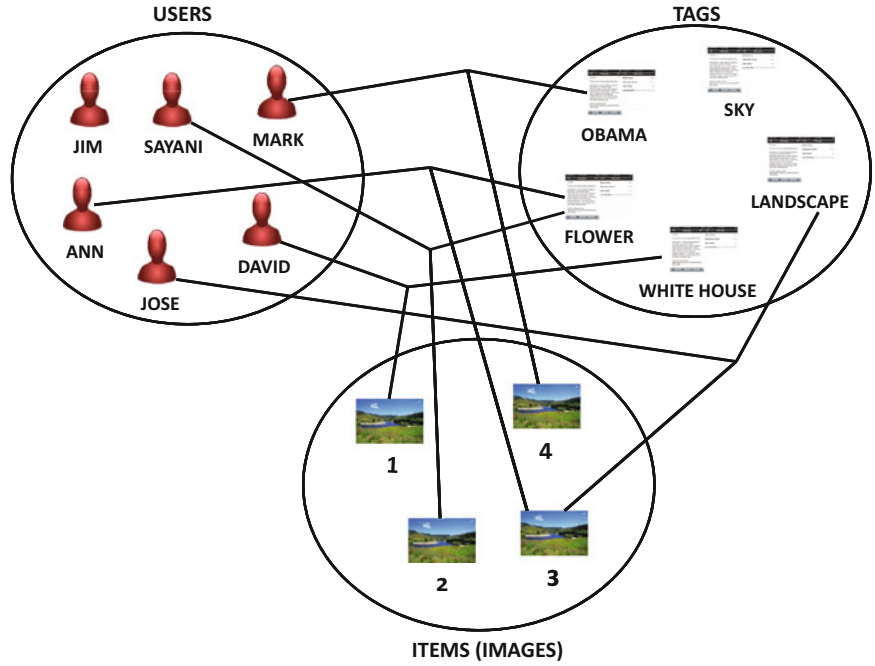
In practical settings, the unspecified values are set to 0 by default, as is common in highly sparse implicit-feedback settings. Henceforth, we will refer to F as the *tag-cube*. It is immediately evident from Figure 11.6 that folksonomies have much in common with the multidimensional representation of context-sensitive recommender systems (see Chapter 8). As we will see later, this similarity is very useful because many of the methods of Chapter 8 can be used to resolve some of the queries.

Although Figure 11.6 illustrates a small toy example, the number of users and items may be on the order of hundreds of millions in a social platform like Flickr, and the number of tags may be on the order of millions. Therefore, such systems face challenges of scalability in a data-rich environment. This is both a challenge and an opportunity for research in the field of social-tagging recommender systems.

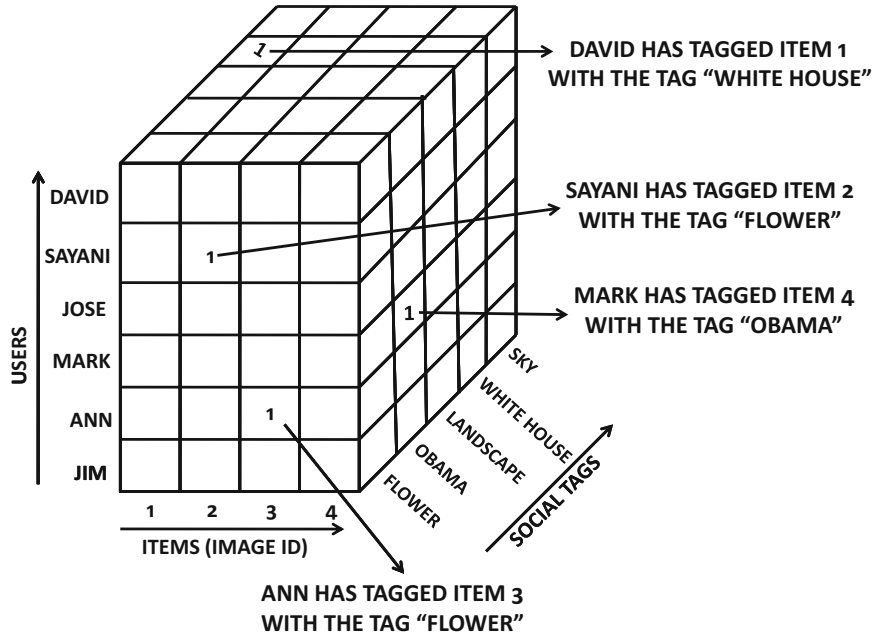
11.4.2 Collaborative Filtering in Social Tagging Systems

The nature of the recommendation formulation depends on the type of the underlying application. In some sites, such as Flickr, tagging information is available but rating information is not. In such cases, one can still develop a social tagging recommender system in which either tags or items are recommended based on the underlying patterns in the tag cube. In other cases, a separate $m \times n$ ratings matrix R is available along with the $m \times n \times p$ tag cube F. The ratings matrix is defined over the same set of users and items as the tag cube. For example, the *MovieLens* site contains both ratings and tagging information. The resulting collaborative filtering system is said to be a tag-*aware* recommender system in which the ratings matrix is the primary data, and the tagging information provides additional side information to improve the accuracy of rating prediction. Note that the ratings matrix could be an implicit-feedback matrix, such as in the case of last.fm, where accesses to resources are logged. In fact, implicit feedback is more common in social tagging sites. From an algorithmic perspective, implicit-feedback matrices are easier to use because they generally do not contain negative preferences, and missing entries can often be treated as 0 as an approximation. In the following, we will assume an explicit ratings matrix, unless otherwise specified.

When the ratings matrix is available, richer varieties of collaborative filtering queries can be formulated than in the case where only tagging information is available. In such cases,



(a) The hypergraph representation



(b) The multidimensional cube representation

Figure 11.6: Representations of a folksonomy

the tag-cube and the ratings matrix share the user and item dimensions in common, but the ratings matrix does not contain the tag dimension. The information in these two sources of information can then be integrated in order to provide recommendations. It is noteworthy that this scenario can be viewed as a generalization of content-based collaborative filtering applications. In content-based collaborative filtering, keywords are associated only with items, whereas in the tag-cube keywords are associated with user-item *combinations*. One can view content-based recommender systems as a special case of the tag-cube, in which the 2-dimensional item-tag slice for each user is identical. Therefore, many of the methods discussed in the following sections can also be used for content-based collaborative filtering applications.

Because of this wide diversity in the underlying applications, the problem of collaborative filtering can be posed in a variety of ways, not all of which have been fully explored in the literature. In fact, much remains to be done to advance the field with respect to collaborative filtering, as it is a relatively recent area of interest. Some examples of possible queries are as follows:

1. **(Tagging data only)** Given an $m \times n \times p$ tagging cube F, recommend
   (a) a ranked list of tags to user i,
   (b) a ranked list of other users with similar interests (tagging patterns) to user i,
   (c) a ranked list of items to user i,
   (d) a ranked list of tags to user i for item j, and
   (e) a ranked list of items to user i for tag context k.
2. **(Tagging data and ratings matrix)** Given a ratings matrix R and an $m \times n \times p$ tagging cube F, recommend
   (a) a ranked list of items to user i, and
   (b) a ranked list of items to user i for tag context k.

The aforementioned queries can be partitioned into two categories. The first set of queries does not use a ratings matrix. In such queries, the recommendation of tags and users is generally more important than the recommendation of items, although one might also use the approach for item recommendation. Because tags serve as bookmarks and indexes for resources (items), finding relevant tags is a way of finding relevant items. The second set of queries is more closely aligned with traditional recommender systems because they are primarily based on the ratings matrix R. The only difference from traditional recommender systems is that the tag-cube is used as side information, and it plays a secondary role because of the amount of noise in it. Such methods are also referred to as *tagommenders* [535] or *tag-informed collaborative filtering* [673]. The primary strength of these systems lies in their ability to integrate the best of both worlds in user ratings and tag activity. There are generally fewer methods of the second type, but an increasing number of methods are able to integrate the knowledge in ratings matrices and the tag-cube. It is important to note that the matrix R might be unary in cases where explicit ratings are not available, and only implicit feedback (e.g., buying behavior) is available. It is, nevertheless, important to understand that the matrix R is an independent source of information from the tag-cube even when it is implicitly derived.

Table 11.1: A list of features [536] used to evaluate tag quality

Feature	Specificity	Criterion by which tags are ordered
num-item-apps	per item-tag	Number of times tag has been applied to a particular item
num-apps	per tag	Number of times tag has been applied across all items
num-users	per tag	Number of users who have applied the tag across all items
num-searches	per tag	Number of searches for the tag
num-search-users	per tag	Number of users who have searched for the tag
tag-share	per item-tag	Fraction of item tags for a particular tag value
avg-fraction-items-tagged	per tag	Average across all users, the fraction of all the items tagged by the user that have the tag
apps-per-item	per tag	Average number of times they are applied to their items
num-tag-words	per tag	Number of words in the tag
tag-length	per tag	Number of letters in the tag

11.4.3 Selecting Valuable Tags

Tags are generally quite noisy because of the open way in which they are contributed and used. In many cases, users might use non-standard vocabulary or misspellings to tag items. This can result in a large fraction of noisy and irrelevant tags. If irrelevant tags are used, then it can have a detrimental effect on many recommendation applications. Therefore, it is helpful to preselect a smaller number of tags. The pre-selection of tags also helps reduce the complexity of the mining process from a computational point of view. Therefore, tag-selection algorithms generally order the tags based on simple criteria and then preselect the top-ranked tags based on these criteria.

Many tagging sites use a simple methodology, referred to as *num-item-apps*, in which the number of people who have added a particular tag to an item is used as an estimate of how much other people would like to see that tag in the future. This can also be considered a proxy for the value of the tag. There are other intuitive features that are commonly used to estimate the quality of tags. For example, some tags may be globally valuable, whereas the value of other tags may be specific to particular items. A number of such features have been proposed in [536] for evaluating the quality of tags. A list of some of these features is provided in Table 11.1. In each case, the specificity of the tag (global or local) is also indicated. It is noteworthy that some of the features in [536] assume that users have rated the tags themselves with thumbs-up or thumbs-down ratings. Such information may not always be available in all systems, and therefore these features are not included in Table 11.1. An experimental methodology for evaluating these features is discussed in [535, 536]. It was found that features such as *num-item-apps*, *tag-share*, and *avg-fraction-items-tagged*, provided good performance. On the other hand, some features, such as *num-apps*, *num-users*, and *tag-length* did not provide the best performance. Furthermore, combining the five best features into a single feature called *all-implicit* provided better performance than any of the individual features. More details are provided in [535, 536] on the inference of this particular feature.

Aside from these methods, it is also possible to use the feature selection methods in section 4.3.4 of Chapter 4. The first step is to convert the tag-cube into a 2-dimensional item-tag slice by aggregating all the item-tag frequencies of the various users. By treating each tag as a "term," this approach results in a term-document matrix. Any of the methods discussed in section 4.3.4 may be used to select the most discriminative tags.

11.4.4 Social-Tagging Recommenders with No Ratings Matrix

This case can also be viewed as a special case of the multidimensional model in context-sensitive recommender systems. The tag-cube can be viewed as a multidimensional cube, in which the tags represent the context. Therefore, the context-sensitive model can be used to resolve these queries. In fact, the tensor factorization models used for context-sensitive ranking [495, 496] are not very different in principle from those used in tag recommendation [497, 498]. A detailed discussion of the multidimensional model for context-sensitive systems is provided in section 8.2 in Chapter 8.

As discussed earlier, queries in social-tagging recommenders can be formulated in a variety of ways, where one might recommend item, tags, or users. The tagging cube is 3-dimensional, and one might recommend along any of the dimensions. Among these various forms, the recommendation of tags is the most common. The reason for this is that the recommendation of tags has a benefit both to the user and to the platform hosting the tagging system:

1. *Utility to hosting platform:* As tags are non-standard, different users may describe the same resource using different keywords. Recommending tags for a specific item helps in consolidating the descriptions. Such a consolidation of the underlying description helps the system to collect better tags, and therefore improve the quality of the recommendations.

2. *Utility to user:* Users may either be recommended a tag specific to an item, or they may be recommended a tag specific to their own interests. The recommendation of item-specific tags is motivated by the fact that users may find it burdensome to assign tags to items. When relevant tags are recommended for a given item, it makes their job easier and also makes it more likely that they will participate in the tagging process. This is, in turn, beneficial in collecting more tagging data. User-specific recommendation of tags is beneficial because tags often serve the goal of organizing the items in a personalized way for various users. For example, Figure 11.6 might represent an image-browsing environment such as Flickr. If Ann is recommended the tag *sky* based on her other tags, then by clicking on this tag she might be able to discover other items of interest to her. It is also possible to combine tagging data with ratings matrices to make high-quality recommendations.

The following section reviews the variety of methods that have been proposed for recommendations in social-tagging systems.

11.4.4.1 Multidimensional Methods for Context-Sensitive Systems

The multidimensional methods discussed in section 8.2 of Chapter 8 can be used to build social-tagging recommenders. The basic idea is to project the data along a particular pair of dimensions for queries along two dimensions and use prefiltering methods for contextual queries along three dimensions.

For example, in order to recommend the best tags to a particular user, one can aggregate the frequencies of the tags over various items. In other words, one determines the number of times that a user used a particular tag over all items. This results in a 2-dimensional user-tag matrix of non-negative frequencies. Any traditional collaborative filtering algorithm can be used on this matrix in order to recommend tags to a user. Such an approach is best for recommending tags to users, but they do not use the item context; nonetheless, this approach is quite useful in real-life settings. Since tags serve the dual function of indexes

to resources, the tags can be used by users to discover resources they might be interested in. Similarly, aggregating frequencies along the tag dimension leads to a user-item matrix. This matrix can be used to recommend items to users.

One disadvantage of using these aggregation methods is that the information along one of the dimensions is ignored. It is also possible to combine the information from all the dimensions during recommendation. Suppose that we want to recommend the best tags or the best items to a particular target user. One way of doing this would be to compute the similarity of users to the target user based on the aggregated user-tag matrix. This computation can also be performed with the aggregated user-item matrix. A linear combination of these two criteria is used to generate the most similar users to the target user. Then, the standard prediction method (cf. Equation 2.4 of Chapter 2) for user-based prediction can be leveraged to recommend either the most relevant items or the most relevant tags to the target. A similar approach can be used for item-based collaborative filtering methods by starting with a target item and finding the most similar items on the basis of either the aggregated user-item matrix or the aggregated tag-item matrix.

Another useful query is the recommendation of items to a user for a particular tag context. The prefiltering and postfiltering methodologies (cf. sections 8.3 and 8.4 of Chapter 8) for context-sensitive systems can be used to achieve this goal. For example, if one wanted to recommend movies related to the tag "animation," then the slice of the tag-cube corresponding to "animation" can be extracted. This process results in a 2-dimensional user-item matrix that is specific to animation movies. Traditional collaborative filtering algorithms can be applied to this matrix in order to make recommendations. One challenge with the use of the approach is that the extracted user-item slice might be too sparse. In order to address the sparsity issue, one can group related tags with the use of *tag clustering*. For example, a tag cluster might contain "animation," "children," "for kids," and so on. The user-item tag frequencies over these related tags can be added together to create an aggregated user-item matrix, which is less sparse. Recommendations can be performed over this aggregated matrix. Tag-clustering methods are proposed in [70, 215, 542]. Although the works in [70, 215, 542] explore the use of tag clustering for content-based methods, such techniques can also be used to improve the effectiveness of collaborative filtering applications.

Finally, the class of tensor factorization methods have found increasing popularity in social tagging. These methods are discussed in section 8.5.2 of Chapter 8, as a special case of context-sensitive systems. A particularly popular method, which is discussed in that section, is the *Pairwise Interaction Tensor Factorization (PITF)* method. In addition, these methods have been generalized to the notion of factorization machines, which can be viewed as generalizations of large classes of latent factor models. Refer to section 8.5.2.1.

11.4.4.2 Ranking-Based Methods

Ranking-based methods use the *PageRank* methodology in order to make recommendations in the presence of tags. A detailed description of ranking methods is provided in section 10.2 of Chapter 10. The two notable methods in this regard are *FolkRank* [256], and *SocialRank* [602]. The main difference between *SocialRank* and *FolkRank* is that *SocialRank* also uses content-centric similarity between the objects in the ranking process. For example, links might be added between pairs of images based on their content-centric similarities. Furthermore, *SocialRank* can be applied to arbitrary social media networks, rather than the tagging hypergraph. Therefore, *SocialRank* makes significant changes to the *PageRank* algorithm in order to balance the effects of the different modalities. The method

can, nevertheless, be applied to a folksonomy as well. *FolkRank* is specifically designed to work with the tagging hypergraphs created in folksonomies. As *SocialRank* is already discussed in section 10.2.3.2 of Chapter 10, we will focus only on the *FolkRank* method in this description.

FolkRank is a simple adaptation of personalized *PageRank* (cf. 10.2.2 of Chapter 10). The first step in applying *FolkRank* is to extract a tripartite graph from the tag hypergraph. The tripartite graph $G = (N, A)$ is extracted from the tag hypergraph as follows:

1. Each tag, user, and item becomes a node in graph G. In other words, each $i \in N$ is a user, tag, or item. Therefore, for m users, n items, and p tags, the graph G contains $(m + n + p)$ nodes.

2. For each hyperedge between a tag, user, and item, undirected edges are added between each pair of entities. Therefore, three edges are added for each hyperedge.

The personalized *PageRank* method is then applied directly to this network. The personalization vector of section 10.2.2 is set in such a way that preferred items, users, or tags have a higher probability of restart. By setting the restart probability in different ways, one can query for specific users, tags, items, user-item pairs, user-tag pairs, or tag-item pairs. The responses to the queries can also be obtained in all modalities.

As a result of the process, highly ranked tags, users, and items provide different views of relevant nodes in the network. An important aspect of *FolkRank* is that it takes global popularity (reputation) into account in addition to the user-specific relevance. This is because all ranking mechanisms tend to favor highly connected nodes. For example, a tag that is used heavily will always be ranked highly even in personalized *PageRank* mechanisms. The value of the restart probability regulates the trade-off between specificity and popularity. Therefore, a *differential* version of *FolkRank* has also been developed, in which these effects are canceled out. The basic idea of the differential version is to perform the following steps:

1. *PageRank* is performed on the extracted tripartite graph with no bias. In other words, the restart probabilities of all nodes have the same value of $1/(m+n+p)$. Recall that the tag-cube is of size $m \times n \times p$ and the number of nodes in the network is $(m+n+p)$. Let the resulting probability vector be given by $\overline{\pi_1}$.

2. Personalized *PageRank* is performed by setting an increased bias value for the specific user-item combination being queried. For example, consider the case where a particular user-item combination is queried. The restart probability for the queried user node can be set to be proportional to $(m + 1)/(2m + 2n + p)$, the restart probability for the queried item node can be set to be proportional to $(n + 1)/(2m + 2n + p)$, and the restart probability of the remaining nodes can be set to be proportional to $1/(2m + 2n + p)$. Let the resulting probability vector be given by $\overline{\pi_2}$.

3. The relevance of the various nodes in all modalities can be extracted from the vector $\overline{\pi_2} - \overline{\pi_1}$. The values may be either positive or negative, depending on the level of similarity or dissimilarity.

The main advantage of such an approach is that it cancels out the global popularity effects to a large extent.

11.4.4.3 Content-Based Methods

Content-based methods can be used in order to make the recommendations of both items and tags to users. In order to recommend items to users, a user-specific training data set

can be created in which each item is described by its tag frequencies over the m users. These frequencies can be represented in tf-idf format. For a given user, the training data contains all the items that she has tagged, as well as a negative sample of items she has not tagged. These are objects for which the tagging frequency needs to be learned. The feature variables and the dependent variable (for the learning process) correspond to the tf-idf representation of each item, and the number of tags the user has placed on each item. Note that the dependent variable is 0 for negative samples. A regression-based model is applied to this training data in order to make predictions.

A similar approach can be used to recommend tags to users instead of recommending items to users. The main difference is that tags are represented as tf-idf vectors of items instead of the other way around. The training data is now generated using tags as objects that need to be classified. Therefore, tags have labels attached to them, based on the number of times that the user has used them on different items. This training model is used to predict the interest of the user in the tags for which the user interest is unknown. A comparison of various content-based methods for tag recommendation is provided in [264].

An item recommendation algorithm, based on tag clustering, was presented in [542]. The clusters are created using the tf-idf representation of the tags in terms of items. In other words, each tag is treated as a vector of item frequencies; then these vectors are used to create m clusters. The clustering process provides an intermediate representation in terms of which the user interest and item relevance are measured and integrated.

Let the interest of the ith user in the sth cluster be denoted by $ucW(i, s)$, and the relevance of the jth item (resource) to the sth cluster be denoted by $rcW(j, s)$. The value $ucW(i, s)$ is computed as the fraction of the tags of user i belonging to the sth cluster, and the value of $rcW(j, s)$ is computed as the fraction of the tags of item j belonging to the sth cluster. Then, the overall interest $I(i, j)$ of user i in item j is computed as follows:

$$I(i,j) = \sum_{s=1}^{m} ucW(i,s) \times rcW(j,s) \tag{11.15}$$

The computation of the interest with the use of clusters as an intermediate step is shown in Figure 11.7. Note that this interest can be used to rank items for users. The basic idea is that clusters provide a robust summary of the sparse user-item tagging behavior, which can be used to make high-quality interest computation.

In addition, the work in [542] uses the approach to provide personalized item responses to user tag queries. For example, if Mary searches for "animation," she might not be recommended the same movies that Bob would be suggested for the same query. For a given queried tag q, its similarity $S(j, q)$ to item j is defined in terms of the relative frequency f_{jq} with which the item j is tagged with q, in comparison with the frequency of other tags on item j:

$$S(j,q) = \frac{f_{jq}}{\sqrt{\sum_s f_{js}^2}} \tag{11.16}$$

Although the value $S(j, q)$ can directly be used to rank items in response to the search of a particular user i, the results are personalized by using the user interest $I(i, j)$ of the searcher. The value of $I(i, j)$ is computed using Equation 11.15. The query results are, therefore, ordered by $S(j, q) \times I(i, j)$ instead of ordering them by $S(j, q)$. It is noteworthy that recommending items for tag queries does not necessarily require the use of user-specific personalization, because one can simply use $S(j, q)$ to rank items. Furthermore, recommending tags for items does not require the use of personalization either. One can simply use the tagging characteristics of the items in order to make recommendations of tags to users. In such

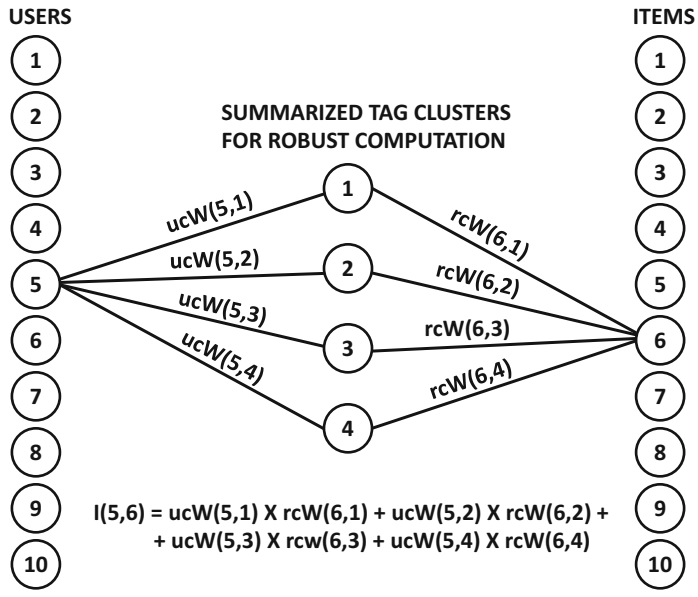


Figure 11.7: Using clusters as a bridge to compute the interest of users for items. The example illustrates the computation of the interest of user 5 for item 6. Such a computation can be performed for any user-item pair.

.

cases, the recommended tag will not depend on the querying user, but rather will depend on the queried item. In fact, the earliest works on tag recommendation use co-occurrence statistics between tags and items to make recommendations. The results are, therefore, not dependent on the user making the query.

Similarly, an approach proposed in [316] uses latent Dirichlet allocation (LDA) to make recommendations based on content-centric topic modeling by treating each item as a "document" containing tag (or "word") frequencies. Similar to traditional topic modeling on documents, the approach shows that the qth tag is related to item j by the following relationship:

$$P(\text{Tag } = q|\text{Item } = j) = \sum_{s=1}^{K} P(\text{Tag } = q|\text{Topic } = s) \cdot P(\text{Topic } = s|\text{Item } = j) \quad (11.17)$$

Here, K represents the total number of topics, which is a user-defined parameter. Note that the left-hand side of Equation 11.17. provides the recommendation probability for ranking purposes, whereas the quantities on the right-hand side can be estimated from the parameter learning process of LDA methods. It is not necessary to use LDA for topic modeling. For example, the simpler Probabilistic Latent Semantic Analysis (PLSA) model may be used in lieu of LDA. Note that it is also possible to personalize this recommendation by treating the sets of tags of a *user* as "documents" and clustering these users into topics with the use of topic modeling After these topics have been determined, we can compute the relevance of the various tags to each user as follows:

$$P(\text{Tag } = q|\text{User } = i) = \sum_{s=1}^{K} P(\text{Tag } = q|\text{Topic } = s) \cdot P(\text{Topic } = s|\text{User } = i) \quad (11.18)$$

Note that Equation 11.18 uses a different set of topics than Equation 11.17; the former clusters the users, whereas the latter clusters the items. A linear combination of Equations 11.17 and 11.18 can be used to determine the relevance of tag q to item j, given the personalized context of user i. The weights of the linear combination will decide the trade-off between user-specificity and item-specificity.

There are other ways of combining user-specificity and item-specificity by directly adapting some of the Bayesian ideas in topic modeling [315]. Specifically, we can directly compute the *personalized* and *item-specific* recommendation probability $P(\text{Tag} = q|\text{User} = i, \text{Item} = j)$. This probability can be simplified with the use of the naive Bayes rule as follows:

$$P(\text{Tag} = q|\text{User} = i, \text{Item} = j) = \frac{P(\text{User} = i, \text{Item} = j|\text{Tag} = q) \cdot P(\text{Tag} = q)}{P(\text{User} = i, \text{Item} = j)} \tag{11.19}$$

$$\approx \frac{P(\text{User} = i|\text{Tag} = q) \cdot P(\text{Item} = j|\text{Tag} = q) \cdot P(\text{Tag} = q)}{P(\text{User} = i, \text{Item} = j)} \tag{11.20}$$

$$\propto P(\text{User} = i|\text{Tag} = q) \cdot P(\text{Item} = j|\text{Tag} = q) \cdot P(\text{Tag} = q) \tag{11.21}$$

Note that we have ignored the term $P(\text{User} = i, \text{Item} = j)$ in the denominator to a constant of proportionality. This is because we wish to *rank* the different tags in order of their recommendation probability for a *specific* user and item, which are already fixed. Therefore, this constant term can be ignored for ranking purposes.

Now, the terms $P(\text{User} = i|\text{Tag} = q)$ and $P(\text{Item} = j|\text{Tag} = q)$ in the right-hand side of the aforementioned equations can be expressed in terms of user recommendation and item recommendation probabilities using Bayes rule:

$$P(\text{User} = i|\text{Tag} = q) = \frac{P(\text{User} = i)P(\text{Tag} = q|\text{User} = i)}{P(\text{Tag} = q)} \tag{11.22}$$

$$P(\text{Item} = j|\text{Tag} = q) = \frac{P(\text{Item} = j)P(\text{Tag} = q|\text{Item} = j)}{P(\text{Tag} = q)} \tag{11.23}$$

Therefore, on substituting these terms in Equation 11.21, we obtain the following:

$$P(\text{Tag} = q|\text{User} = i, \text{Item} = j) \propto \frac{P(\text{Tag} = q|\text{User} = i) \cdot P(\text{Tag} = q|\text{Item} = j)}{P(\text{Tag} = q)} \tag{11.24}$$

The terms on the right-hand side can be estimated easily in a data-driven manner, as in any Bayes classifier. For example, the value of $P(\text{Tag} = q)$ can be estimated as the fraction of non-empty cells in the tag-cube for which the qth tag has been specified. The value of $P(\text{Tag} = q|\text{User} = i)$ can be estimated as the fraction of non-empty cells of the slice of the tag-cube for user i that correspond to the qth tag. The value of $P(\text{Tag} = q|\text{Item} = j)$ can be estimated as the fraction of the non-empty cells of the slice of the tag-cube for item j that correspond to the qth tag. Laplacian smoothing is often used to avoid overfitting.

The probabilities in Equation 11.24 are used to rank the tags for the specific user-item combination. The work in [315] also discusses a simpler frequency-based model for performing the recommendation.

11.4.5 Social-Tagging Recommenders with Ratings Matrix

Tags have significant potential in improving the quality of recommendations when they are used in addition to item ratings. For example, consider a scenario where Mary has watched

many movies such as *Shrek* and *Lion King*, which are annotated with the tag "animated" on a rating site such as IMDb. However, Mary might not have tagged any of these movies in the tag-cube, and these preferences are derived from the ratings matrix.

Now consider the case where a movie such as *Despicable Me* is also tagged as "animated," although Mary has not watched this movie yet. In such a case, it is reasonable to assume that Mary might also be interested in watching the movie *Despicable Me.* Although the ratings matrix might also provide the same prediction, the chances of prediction error are reduced when tag information is incorporated because it provides an independent source of information. In particular, this is true when a movie is new and there are too few ratings or tags to make reliable predictions about user preference. In such cases, the ratings and tags can complement each other to make more robust decisions. In most cases, tagging systems contain *implicit* ratings (e.g., whether or not a user has viewed an item) in the ratings matrix. This is because sites such as last.fm automatically log data about the items that a user might have consumed. Note that the implicit ratings are an independent source of information because a user might view an item, even though they might not tag it. In this section, we will study both the case of implicit and explicit ratings.

The most straightforward approach is to use hybrid recommender systems to combine the predictions based on tags and ratings. For example, any of the methods discussed in section 11.4.4 can be used to make item predictions based purely on tags. Furthermore, any traditional collaborative filtering algorithm can be used to make predictions based on ratings. A weighted average of the two ratings can be used to make the final prediction. The weights can be learned using the method discussed in section 6.3 of Chapter 6 on hybrid recommender systems. However, such an approach does not integrate the two sources of prediction very tightly. Better results may be obtained with algorithms that tightly integrate the various sources of data in the recommendation process.

11.4.5.1 Neighborhood-Based Approach

The method in [603] works with implicit feedback data sets in which the ratings matrices are assumed to be unary. This is quite commonly the case in social-tagging systems. For example, in a site such as last.fm, user accesses of items are available, but explicit ratings are not available. The paper treats missing entries as 0 values. Therefore, the ratings matrix R is treated as a binary matrix rather than as a unary matrix.

The approach of [603] augments the $m \times n$ ratings matrix R with data from the $m \times n \times p$ tag cube F, by creating additional pseudo-users or pseudo-items. For example, user-based collaborative filtering can be performed on a ratings matrix with an extended set of items. In order to create a ratings matrix R_1 with the item dimension extended, each tag is treated as a pseudo-item. Furthermore, the value of a user-tag combination is assumed to be 1, if the user has used that tag at least once (possibly over multiple items). Otherwise, the value is equal to 0. Note that there are $m \times p$ user-tag combinations. One can then append these $m \times p$ combinations to the $m \times n$ ratings matrix R by treating the tags as new pseudo-items. This results in an extended matrix R_1 of size $m \times (n + p)$. The similarity between a user i and other users is computed using this extended matrix. The similarity computation is enriched because of the additional columns containing user-tag activity information. The item ratings of the user i are computed using the number of 1 values in the peer group of i. These predicted ratings $\hat{r}_{ij}^{user}$ are normalized to sum to 1 over the different values of item index j so that they represent probabilities of accessing (or buying) various items. Note that ratings represent frequency of activity in these implicit feedback settings.

The item-based approach can be extended in a similar way. In this case, a $p \times n$ matrix corresponding to tag-item combinations is created. A value in this matrix is 1 if the item is tagged at least once. The tags are now treated as pseudo-users, and appended as rows to the original ratings matrix R. This results in an extended matrix R_2 of size $(m + p) \times n$. This extended matrix is used to perform similarity computations in item-based collaborative filtering. The predicted ratings $\hat{r}_{ij}^{item}$ for a given user i are then normalized so that they sum to 1 over all j. Therefore, the predicted ratings represent probabilities of accessing or buying items in this case as well.

After performing user-based and item-based collaborative filtering, the ratings predictions of the two cases are fused using a parameter $\lambda \in (0, 1)$:

$$\hat{r}_{ij} = \lambda \cdot \hat{r}_{ij}^{user} + (1 - \lambda) \cdot \hat{r}_{ij}^{item} \tag{11.25}$$

The optimal value of λ can be chosen using cross-validation. The results in [603] showed improvements over traditional collaborative filtering when tag information was also used. The fusion of user-based and item-based methods was necessary to achieve the improvements from the incorporation of tagging.

11.4.5.2 Linear Regression

The method in [535] uses linear regression to integrate the tags into the recommendation process. As tags are generally less statistically precise in identifying user preference than are ratings, it is important to select only valuable tags for the recommendation process. To achieve this goal, the methodology described in section 11.4.3 may be used. The basic approach in [535] fuses the information in the user ratings in order to enrich the information about the tag preferences for various items. For example, if a user has rated *Lion King* and *Shrek* highly, and both movies are tagged as "animation," it can be inferred that the user is likely to be interested in movies with this tag. The first step is to determine the relevance weighting between an item and a tag. For example, any of the item-tag specific quantifications in Table 11.1 may be used. Then, if q_{jk} is the relevance of item j to tag k, then the item-preference value is further transformed with the sigmoidal function:

$$v_{jk} = \frac{1}{1 + \exp(-q_{jk})}$$

Then, the user preference u_{ik} of user i for tag k is computed by combining the tag-item relevance with the user interest in the items. The user interest in the items may be inferred using the ratings matrix $R = [r_{ij}]$. The preference u_{ik} of user i for tag k may be inferred as follows:

$$u_{ik} = \frac{r_{ij} \cdot v_{jk}}{\sum_{s=1}^{n} r_{is} \cdot v_{sk}} \tag{11.26}$$

Items that are not rated by user i are ignored in the numerator and denominator. When ratings are not available, the value of u_{ik} can also be indirectly inferred from the frequency of the user's visits, clicks, buys, or tags on items. For example, the value of r_{ij} in Equation 11.26 can be replaced with the number of times the user has tagged item j (not necessarily with the tag k).

A simple approach to predict a preference score p_{ij} of an item j for user i is to determine all the tags T_j of that item and average the value of u_{ir} over all tags $r \in T_j$:

$$p_{ij} = \frac{\sum_{r \in T_j} u_{ir} \cdot v_{jr}}{\sum_{r \in T_j} v_{jr}} \tag{11.27}$$

Note that the value p_{ij} might not lie in the range of ratings. Nevertheless, it can still be used to *rank* items for a user.

A more effective approach to predict ratings is to use *linear regression*. The basic idea in linear regression is to assume that the rating r_{ij} of user i for item j is based on a linear relationship, which is true for fixed j and varying values of i:

$$r_{ij} = \sum_{r \in T_j} u_{ir} \cdot w_{jr} \quad \forall i : r_{ij} \text{ is observed} \tag{11.28}$$

The (unknown) coefficient w_{jr} represents the importance of tag r for item j, and it can be learned using regression over all the ratings that are observed for the item j. The main difference from Equation 11.27 is that instead of using a heuristic value of v_{jr} as the weight of tag r (specific to item j), we are *learning* w_{jr} using linear regression on the ratings matrix. The resulting approach is generally superior because of the greater level of supervision. As the regression training process includes all users that have rated item j, the approach does use the collaborative power of the ratings in different users. Furthermore, this method also provides superior results to conventional collaborative filtering algorithms because of its use of side information available in the tags. Combining this methodology with a simple matrix factorization method into a hybrid system provided even better results [535]. It was shown that regression support vector machines provide the best results for the training process, although least-squares regression provides a simpler alternative. Linear regression methods are discussed in section 4.4.5 of Chapter 4.

11.4.5.3 Matrix Factorization

A matrix factorization approach, referred to as *TagiCoFi* [673], uses a variation of the methods discussed in Chapter 3 to approximately factorize the ratings matrix R into two matrices, an $m \times q$ matrix U and an $n \times q$ matrix V. This condition can be expressed as follows:

$$R \approx UV^T \ \forall \text{ observed entries of } R \tag{11.29}$$

This condition can be imposed by approximately minimizing the Frobenius norm $g(U, V, R) = ||R - UV^T||^2$ over the observed entries of R.

In addition, a similarity constraint is imposed over the user factor matrices U, so that users with similar tagging behavior have similar factors. Let S_{ij} be the similarity between users i and j and let $\overline{u_i}$ be the ith row of U. The computation of S_{ij} from the tagging behavior will be described later. Then, in order to ensure that users with similar tagging behavior have similar factors, we would like to also minimize the following factor similarity objective $f(U)$:

$$f(U) = \sum_{i=1}^{m} \sum_{j=1}^{m} S_{ij} ||\overline{u_i} - \overline{u_j}||^2 \tag{11.30}$$

As we have two different criteria, defined by the objective functions $g(U, V, R)$ and $f(U)$, the balancing parameter β can be introduced to minimize $g(U, V, R) + \beta f(U)$. In addition, we have the standard regularization term in matrix factor factorization, which is given by the sum of the Frobenius norms of the factor matrices. This regularization term is

$\lambda\left(||U||^2 + ||V||^2\right)$, where λ is the regularization parameter. Summing up these different terms, we derive the following objective function:

$$\text{Minimize } J = \underbrace{||R - UV^T||^2}_{\text{Observed entries in } R} + \underbrace{\beta \cdot \sum_{i=1}^{m}\sum_{j=1}^{m} S_{ij}||\overline{u_i} - \overline{u_j}||^2}_{\text{Tagging similarity objective}} + \underbrace{\lambda\left(||U||^2 + ||V||^2\right)}_{\text{Regularizer}}$$

As in the case of all matrix factorization methods, a gradient descent method is used to determine the factor matrices U and V. The values of β and λ can be computed using cross-validation methods.

It is noteworthy that this approach is technically similar to a social regularization approach [382] discussed in section 11.3.8.2 on trustworthy recommender systems. In that approach, a trust matrix T is used to add the similarity term $\sum_{i,j:t_{ij}>0} t_{ij}||\overline{u_i} - \overline{u_j}||^2$ to the objective function. Here, the tagging similarity matrix is used to add the term $\sum_{i=1}^{m}\sum_{j=1}^{m} S_{ij}||\overline{u_i} - \overline{u_j}||^2$. In other words, the trust/homophily t_{ij} between users i and j is replaced with the tagging similarity S_{ij} between users i and j. Thus, minor variations of the same technical model can be used to address diverse social recommendation scenarios. Furthermore, instead of forcing the user factors to be more similar, one can also force the item-factors to be more similar based on tagging behavior (see Exercise 5).

Computing Tagging Similarity

The aforementioned approach requires the computation of the tagging similarity S_{ij} between the users i and j. First, the tf-idf matrix is generated from the tag-cube F in which the number of times the user has used a particular tag is computed. In other words, the number of 1s of a particular user-tag combination over all items are summed. Thus, a frequency vector is generated for each of the m users. This frequency is then normalized with the standard tf-idf normalization used in information retrieval. The work in [673] proposes two different methods for computing the similarity:

1. *Pearson similarity:* The Pearson correlation coefficient ρ_{ij} is computed over all tags used by user i and user j. Tags not used by either user are ignored. The sigmoidal function is used to transform the correlation coefficient into a non-negative similarity value S_{ij} in $(0, 1)$:
$$S_{ij} = \frac{1}{1 + \exp(-\rho_{ij})} \tag{11.31}$$

2. *Cosine similarity:* The standard cosine similarity between the frequency vectors is used as the similarity value. Refer to Chapter 4 for a discussion of the similarity function.

3. *Euclidean similarity:* The Euclidean distance d_{ij} is computed between the similarity vectors, and then a Gaussian kernel is applied to the distance to transform it into a similarity value in $(0, 1)$:
$$S_{ij} = \exp\left(-\frac{d_{ij}^2}{2\sigma^2}\right) \tag{11.32}$$
Here, σ is a user-controlled parameter, which can be chosen using cross-validation.

In the results reported in [673], the Pearson similarity provided the best performance, whereas the Euclidean similarity provided the worst.

11.4.5.4 Content-Based Methods

Social tagging methods provide a straightforward way of using content-based methods. The frequency vector of the tags on a movie can be viewed as its description. The movies that the user has rated are treated as the training examples on the feature space defined by the tags. The ratings are treated as the class labels. A training model, which is specific to the user, is constructed using this training data. The model is used to predict the ratings of the other movies. Either a classification or a regression model may be used depending on whether the ratings are unary or interval-based. Such content-based models can also be combined with any of the aforementioned collaborative systems.

A simple content-based model for recommendations on the IMDb data set was presented in [584]. It uses the notion of tag clouds to represent the tag-based descriptions of movies. The various keywords are weighted according to their relevance and then combined with ratings to make a final prediction. One challenge with the use of content-based methods is that the tags are very noisy with a significant amount of synonymy. Linguistic methods were used in [178] for disambiguation, and then combined with a naive Bayes classifier. It is also useful to leverage the feature selection methods discussed in Chapter 4 to improve the representation quality.

11.5 Summary

Social information can be used in a wide variety of ways in recommender systems. The standard multidimensional model can be used to incorporate social information in a limited way. Trust-centric methods can be used to create robust recommender systems. Unsupervised methods use trust propagation and aggregation methods to incorporate trust into the recommender system. Supervised methods use link prediction and matrix factorization for more effective performance. Supervised methods are generally considered the state-of-the-art today. Incorporating trust knowledge can make the systems attack resistant and also help in avoiding cold-start issues.

In recent years, social tagging systems have become an avenue for users to collaboratively tag resources on the Web with free-form descriptions. These descriptions are also referred to as folksonomies, which are represented as tag cubes. Such descriptions are useful in terms of the rich content-centric knowledge they contain about user interests. Tag cubes can either be used on a stand-alone basis, or they can be combined with ratings matrices to make recommendations. The former class of methods shares similarities with the multidimensional model for recommendations. The latter class of methods can be based on either collaborative or content-based methods. A variety of techniques, such as neighborhood methods, linear regression, and matrix factorization, have been developed for this scenario.

11.6 Bibliographic Notes

Overviews of trust-based recommender systems may be found in [221, 588, 616, 646]. The doctoral dissertation of Jennifer Golbeck [222] provides several seminal algorithms on the topic. The correlation between homophily in social networks and the notion of trust has been shown in [224, 681]. In these cases, trust relationships can be computationally inferred from Web-based social networks. The work in [187] showed how to infer trust relationships directly from ratings data [187], although there is some debate on whether this notion of trust is generally accepted as it is conventionally used in the literature. One of the earliest

works in applying trust-based methods to such networks was proposed in the context of movie recommendations [223, 225]. The Filmtrust system [225] showed how to use trust metrics for movie recommendations. The work in [592] studies the predictability of distrust relationships from interaction data. Other sites that collect trust-based information include Epinions [705], Moleskiing [461], and Slashdot [706].

Trust metrics play a key role in recommendations in trust networks [344, 680]. The work of [680] provides a good overview of the relevant trust metrics. Although much of the work on trust networks focuses on trust (positive) relationships only, some recent work also discusses the use of both trust and distrust relationships [241, 287, 590, 593, 614, 680]. Furthermore, most of these methods have only discussed methods for (positive) trust propagation, with the exception of the work in [287], which has proposed methods for distrust aggregation as well. The interaction between trust and distrust concepts is studied in [590, 591] in the context of the recommendation and link prediction problems. Methods for trust propagation with multiplicative methods are discussed in [241, 509]. A variety of other trust propagation methods include the use of decay factors along paths [240], using only shortest paths [222], distance from a fixed propagation horizon [403], spreading factors [682, 683], rules [345, 597], and semantic distances [1]. It was shown in [227] through experiments that transitively propagated trust values are more accurately inferred with the use of shortest paths, rather than using all paths. This observation formed the basis of the *TidalTrust* algorithm. Sophisticated methods for de-emphasizing shorter paths include the *Appleseed* algorithm [682], in which a spreading activation model is used. Trust is modeled as energy, which is injected from the source node. The energy is divided among subsequent nodes based on the trust scores along edges. The amount of energy reaching the sink provides the total amount of trust. Clearly, if the sink is connected to the source with many short paths, then more energy will reach the sink. The *EigenTrust* algorithm [292] uses the principal eigenvectors of the trust network to calculate trust values of the source node for all other nodes. However, the approach provides a ranking of the trustworthiness rather than actual trust values. The exploitation of homophily effects for trust propagation is discussed in [594], in which a matrix factorization model is introduced.

A second important aspect of trust computation is that of aggregation. Aggregation rules in social networks are discussed in [1, 221, 222, 287, 449, 615]. The work in [405] discusses methods for weighting the different components of the aggregation based on path length or based on closeness of friendships.

The combination of propagation and aggregation leads to the creation of trust metrics [221, 344]. The *Advogato* trust metric is discussed in [344], and it is one of the classical metrics used in the literature for many applications beyond recommender systems. The trust metrics discussed in this chapter are specialized to recommendation algorithms. The best description of the *TidalTrust* algorithm together with a pseudocode may be found in [222]. The *MoleTrust* algorithm is described in [406]. The effectiveness of *MoleTrust* in the presence of cold-start is shown in [403, 404]. The *TrustWalker* approach is presented in [269], and an axiomatic approach to trust-based recommendation is provided in [48]. The use of link prediction in signed and unsigned networks for recommendation is studied in [157, 324, 325, 580, 581]. The work in [157] is notable because it shows the connections between the matrix factorization methods for link prediction, and the matrix factorization methods for collaborative filtering. The *SoRec* algorithm is proposed in [381], and the *LOCALBAL* algorithm in [594]. The use of both trust and distrust relationships in matrix factorization methods was explored in [383]. The *SocialMF* algorithm was discussed in [270], whereas the similarity-based regularization approach was proposed in [382]. An ensemble method using matrix factorization, known as *social trust ensemble (STE)*, was presented in [384].

The utility of recommender systems for controversial items and users has been studied in several works [222, 406, 617]. It is generally accepted that trust-based methods are particularly useful in such cases. The effectiveness of such systems in the presence of cold-start is shown in [403, 404]. The attack-resistant nature of trust-aware systems is discussed in [344].

A general survey on social-tagging techniques may be found in [237]. A survey of tagging recommender systems is provided in [671], although most of the works discussed in this survey do not use a ratings matrix for the recommendation process. Finally, the recommender systems handbook contains an overview of social-tagging recommender systems [401]. One of the earliest works on tag recommendations was provided in [553], in which simple methods such as co-occurrence, voting, and summing are used to perform recommendations. A hierarchical clustering method for content-based recommendation was proposed in [542]. Probabilistic latent factor models are presented in [316]. Some of the works [135, 179, 584] focus primarily on content-based systems.

Tensor-based methods for tag recommendation are presented in [497, 498, 582, 583]. The notion of factorization machines has found significant popularity in these cases [493, 496]. A particularly notable pairwise approach is the PITF method [496]. A method has been proposed in [487] to leverage latent factor models for application to mining algorithms in the presence of tags. Although this work is not specifically focused on recommender systems, the underlying latent factor models can be used in virtually any application, including recommender systems. Machine learning methods for tag recommendation algorithms are discussed in [250, 555, 556]. Among these works, the techniques in [556] are designed to perform the tag recommendation in real time. Tag clustering methods [70, 215, 542] are often used to alleviate the sparsity problem in collaborative filtering applications. A weighted hybrid method for social tagging methods is discussed in [216].

Various evaluations of tag recommendation methods are provided in [264, 277]. Methods for evaluating tag quality are discussed in [536]. Only a small number of systems today combine the power of ratings matrices with that of social tags [535, 603, 673]. Content-based methods for combining ratings with tagging data are discussed in [179, 584]. For specific data domains, such as music, valuable insights can sometimes be gleaned from the music files for the recommendation process [191]. A solution to the cold-start problem with social tags is discussed in [672].

11.7 Exercises

1. Implement the neighborhood-based method for recommending tags for an item to a user with the use of a linear combination of results obtained on the user-tag matrix and the item-tag matrix.

2. Discuss the relationship of the Katz measure for link prediction with trust propagation and aggregation methods.

3. Implement the gradient descent method of section 11.3.8.

4. The method in section 11.4.5.3 forces the user factors to be more similar based on user-tagging similarity.

   (a) Design a method that forces item factors to be more similar based on item-tagging similarity.

   (b) Design a method that forces both user and item factors to be more similar based on corresponding similarities in user and item tagging.