Mohammad Ali Abbasi, Jiliang Tang, and Huan Liu
Computer Science and Engineering, Arizona State University
{Ali.Abbasi, Jiliang.Tang, Huan.Liu}@asu.edu

Trust-Aware Recommender Systems

Chapter 1

Trust-Aware Recommender Systems

Abstract
Recommender systems are an effective solution to the information overload problem, specially in the online world where we are constantly faced with inordinately many choices. These systems try to find the items such as books or movies that match best with users' preferences. Based on the different approaches to finding the items of interests to users, we can classify the recommender systems into three major groups. First, content based recommender systems use content information to make a recommendation. For example, such systems might recommend a romantic movie to a user that showed interest in romantic movies in her profile. Second, collaborative filtering recommender systems rely only on the past behavior of the users such as their previous transactions or ratings. By comparing this information, a collaborative filtering recommender system finds new items or users to users. In order to address the cold-start problem and fend off various types of attacks, the third class of recommender systems, namely *trust-aware recommender systems*, is proposed. These systems use social media and trust information to make a recommendation, which is shown to be promising in improving the accuracy of the recommendations. In this chapter, we give an overview of state-of-the-art recommender systems with a focus on trust-aware recommender systems. In particular, we describe the ways that trust information can help to improve the quality of the recommendations. In the rest of the chapter, we introduce *recommender systems*, then *trust in social media*, and next *trust-aware recommender systems*.

1.1 Recommender Systems

With the development of Web 2.0, information has increased at an unprecedented rate which aggravates the severity of the information overload problem for online users. For example, a search for "smartphone" returns 1,664,253 results in Amazon[1] products or a search for "best movies to watch" in Google videos[2] returns about 219,000,000 results. Due to the information overload problem, the decision-making process becomes perplexing when one is exposed to excessive information [58, 55, 19, 1]. Therefore, with the rapidly growing amount of available information and digital items on the web, it is necessary to use tools to filter this information in order to find items that are more likely to be of interest to the users.

One can use search engines to overcome the information overload problem. In this case, the user has to refine the search terms or has to pick more specific query terms to narrow down the results. Another solution to overcome the information overload problem is to use top-k recommendations. In this approach, the system keeps a list of the most popular items and utilizes the list to recommend items to the user. For example, Ted[3] is a website that uses this technique to recommend items to users. It can be seen in Figure 1.1, users can sort items bases on the different approaches such as overall popularity (*most viewed*), popularity in the past week (*most emailed this week*), or popularity in the past month (*most popular this month*) among others. Similar to search engines, top-k items are not usually customized based on users' preferences and interest. In particulate, a top-k-item system returns the same list of items to people with different preferences. Therefore, customization is the major problem associated with these two approaches.

Recommender systems are introduced to tackle the information overload, and the customization problem. Recommender systems are a subclass of information filtering systems that consider users' preferences and recommended items that match with users' preferences and interests [23]. These systems have become extremely common in recent years and are applied in a variety of applications including recommending products, social links, and digital items. The most popular ones are probably movies, music, news, books, and products in general [58, 70, 19, 26, 60]. Further, recommender systems are frequently used on recommending social links such as recommending people to follow on Twitter, befriend on social networks or dating sites [67, 37]. Furthermore, these systems are also used to accurately estimate the degree to which a particular user (from now on termed the *target user*) will like a particular item (the *target item*) [73].

Based on the type of data that recommender systems use, we can classify

[1] http://www.amazon.com
[2] https://www.google.com/#q=best+movies+to+watch&safe=active&tbm=vid
[3] http://www.ted.com/

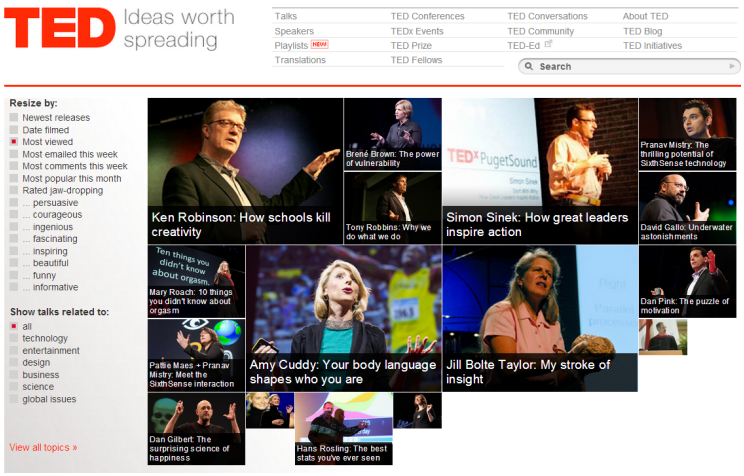

FIGURE 1.1: Ted.com uses a top-k item recommendation approach to rank items

them into two major classes: *content-based* and *collaborative filtering* based recommender systems [76, 60]. Content-based recommendation systems use items' features and characteristics to rank the items based on the user's preferences. Collaborative filtering recommendation systems rely on the user's past behavior e.g., purchases or ratings, to find similar users or items and utilize this information in order to find the items of interests to the user.

In general, recommender systems are utility functions that predict the rating of item i from the item set I for user u from the user set U in the form of $U \times I \rightarrow R$, where r_{ui} is the rating of the item i for the given user u. The task of recommender systems is to predict user u's rating for the given item i for which r_{ui} is unknown and use $\hat{r}_{ui}$ to represent the predicted rating. The ratings, $r_{u,i}$, can be any real number but often ratings are integers in the range $[1, 5]$. We use $\mathbf{R}$ to show all of the ratings. In real-world recommender systems, only a few users rate the items of interests (this number for many recommender system is less than 1%). Matrix 1.1 shows an example of a rating matrix with missing values. The goal of recommender systems is to predict these missing values.

$$\mathbf{R} = \begin{bmatrix} & 5 & 2 & & 3 & \\ 4 & & & 3 & & 4 \\ & & 2 & & & 2 \\ 5 & & & 3 & & \\ & 5 & 5 & & & 3 \end{bmatrix} \tag{1.1}$$

Algorithm 1 Content-based recommendation

1: Describe the items that may be recommended.
2: Create a profile of the user that describes the types of items the user likes
3: Compare items to the user profile to determine what to recommend. The profile is often created and updated automatically in response to feedback on the desirability of items that have been presented to the user.

1.1.1 Content-based Recommendation

Content-based recommender systems uses items' and users' features to create a profile for each item or user. For example, movie profile might include attributes such as gender, participating actors, director, and office box popularity. User profile includes demographic information and users's interests [28]. These systems use supervised machine learning to induce a classifier that can discriminate between items likely to be of interest to the user and those likely to be uninteresting [52, 5, 49]. The recommender recommends an item to a user based on a description of the item and a profile of the users' interests. Algorithm 1 shows the main steps of a content-based recommendation.

We usually use *vector space model* to represent users' and items' features. In this model, every item or user is represented as a vector.

$$i = (t_1, t_2, ..., t_n) \tag{1.2}$$

where t_j is the frequency of term j in item i. To model users or items more accurately, instead of frequency we can use *tf-idf* which can be calculated as follows:

$$tf_{t,i} = \frac{f_{t,i}}{max\{f_{z,i} : z \in i\}} \quad idf_t = \log \frac{N}{n_t} \tag{1.3}$$

$$w_{t,i} = tf_{t,i} \times idf_t \tag{1.4}$$

where $f_{t,i}$ is the frequency of term t in item i, $max\{f_{z,i} : z \in i\}$ is the maximum term frequency in item i, N is the total number of items, n_t is the number of items where term t appears. $tf_{t,i}$ denotes the frequency of term t in item i, and idf_t denotes the inverse document frequency of term t, which inversely correlates with the number of items, that term t is appeared in their descriptions. The similarity between user u and item i can be calculated using Equation 1.5.

$$sim(u, i) = \frac{\sum_{t \in T} w_{t,u} w_{t,i}}{\sqrt{\sum_{t \in T} w_{t,u}^2} \sqrt{\sum_{t \in T} w_{t,i}^2}} \tag{1.5}$$

where T indicates the set of terms that appeared in item and user description.

1.1.2 Collaborative Filtering (CF)

Collaborative filtering is the process of filtering the information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc [69]. Collaborative filtering systems use the user's past behavior, and recommend items that match their taste. Collaborative filtering recommender systems can be classified into *memory-based* and *model-based* collaborative filtering. In memory-based approach we predict the missing ratings based on similarity between users or items. In model-based approach, we use given user-item ratings to construct a model and use the model to predict missing ratings. We'll give a detailed description of these two approaches in the following sections. The main advantage of this method is that the recommender system does not need to have any information about the users and content of the items to recommend. User-item ratings are the only information the system needs to operate. The following are assumptions for collaborative filtering systems [76]:

- Users with similar ratings on some items are more likely to have similar ratings on future items, and
- Items with similar ratings in the past are more likely to have similar ratings in the future.

Figure 1.2 illustrates this approach for a small set of users and movies. The goal is recommending a new movie to Jack. In the first step, the system finds three other users that have similar movie taste as Jack's. The next step it looks for other movies that these users liked. All three of them liked "Once Upon a Time in the West", and two of them liked "Spider man". Therefore, the top recommendation would be "Once Upon a Time in the West".

1.1.2.1 Memory-based Collaborative Filtering

In a memory-based approach, the recommender system aims to predict the missing ratings based on either similarity between users or similarity between items [6, 53, 30]. The former is built upon the hypothesis that similar users have similar tastes. Hence, to make a reasonable recommendation, it finds similar users, then uses these users' taste to make a recommendation for the target user. The second approach is built upon the consistency of a user's taste. If a user liked a product, she will like similar products as well. In both approaches, the recommender system takes two steps to recommend an item to the user. First, it calculates the similarity between users or similarity between items. Then, it uses the most similar users or items to make its recommendation. Collaborative filtering uses the rating information to compute user-user or item-item similarity. *Cosine similarity* and the *Pearson correlation coefficients* are two of the most frequently used similarity measures in collaborative filtering. Given two users u, and v, and their ratings, the cosine similarity is

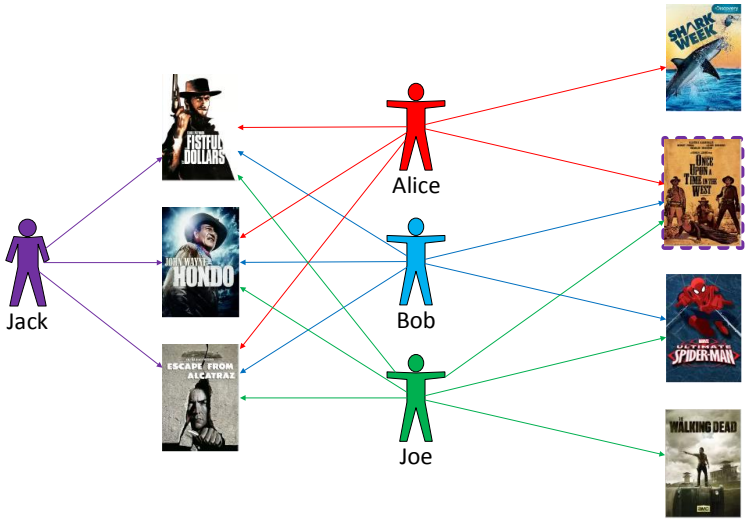

FIGURE 1.2: User-based collaborative filtering. Alice, Bob and Joe are similar to Jack based on their history (movies in the left hand side). Now we want to recommend a new movie from the list in the right hand side to Jack. Based on the past preferences of Alice, Bob, and Joe, "Once Upon a Time in the West" is liked by all of the three, while "Spider Man" is only liked by two of them. Therefore "Once Upon a Time in the West" is likely to be recommended to Jack.

represented as the following using a dot product and its magnitude:

$$similarity(u,v) = \cos(u,v) = \frac{u \cdot v}{\|u\|\|v\|} = \frac{\sum_i r_{u,i} \times r_{v,i}}{\sqrt{\sum_i (r_{u,i})^2} \times \sqrt{\sum_i (r_{v,i})^2}}, \quad (1.6)$$

where $r_{u,i}$ indicates user u's rating for item i. Pearson's correlation coefficient between two variables is defined as the covariance of the two variables divided by the product of their standard deviations.

$$sim(u,v) = \frac{\sum_i (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_i (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_i (r_{v,i} - \bar{r}_v)^2}}, \quad (1.7)$$

where $\bar{r}_u$ denotes the average rating for user u.

Item-based CF

Item-based collaborative filtering proceeds in an item-centric manner. The recommender system builds an item-item matrix determining relationships between pairs of items; then, it infers the tastes of the current user by examining the matrix and matching that user's data [57, 34]. This method was invented and first used by Amazon.com (*users who bought x also bought y*).

Algorithm 2 Item-based collaborative filtering

1: Build an item-item matrix determining relationships between pairs of items
2: Using the matrix, and the data on the current user, infer his taste

A high level algorithm for item-based collaborative filtering is described in Algorithm 2 [31]. We use Equation 1.8 to estimate $\hat{r}_{u,i}$, denoting the user u's rating for item i.

$$\hat{r}_{u,i} = \bar{r}_i + \alpha \sum_{j \in N(i)} (sim(i,j) \times (r_{u,j} - \bar{r}_j)), \tag{1.8}$$

where i and j denotes the items, $\bar{r}_i$ denotes the mean rating of item i, and α is a normalization factor. Many popular web-based service providers such as Amazon and YouTube use item-based collaborative filtering to recommend items.

User-based CF
In a user-based approach, the recommender ranks users based on the similarity among them, and uses suggestions provided by most similar users to recommend new items [75]. The user-based approach of collaborative filtering systems are not as preferred as an item-based approach due to the instability in the relationships between users. For a system which handles a large user base, even the smallest change of the user data is likely to reset the entire group of similar users. We use Equation 1.9 to predict user u's rating for item i, $\hat{r}_{u,i}$.

$$\hat{r}_{u,i} = \bar{r}_u + \alpha \sum_{v \in N(u)} (sim(u,v) \times (r_{v,i} - \bar{r}_v)), \tag{1.9}$$

where $r_{v,i}$ is observed rating of user v for item i, $\bar{r}_u$ is mean rating of user u, $\hat{r}_{u,i}$ is predicted rating of user u for item i, $N(u)$ is the set of users similar to u, $sim(u,v)$ is the similarity between users u and v, and α is normalization factor.

1.1.2.2 Model-based Collaborative Filtering

Unlike memory-based collaborative filtering which uses the similarity between users and items to predict missing ratings, model-based collaborative filtering assumes that an underlying model governs the way users rate the items [76]. In this approach, we aim to learn that model and then use it to predict the missing ratings [53]. Model-based approaches use data mining and machine learning algorithms to find patterns from user ratings. There are many model-based collaborative filtering algorithms, including Bayesian networks, factorization and latent semantic models such as singular value de-

composition, and Markov decision process-based models [62, 20, 48]. In this section we introduce matrix factorization model-based collaborative filtering.

Factorization based CF models assume that a few latent patterns influence user rating behaviors and perform a low-rank matrix factorization on the user-item rating matrix. Let $\mathbf{U}_i \in \mathbb{R}^K$ and $\mathbf{V}_j \in \mathbb{R}^K$ be the user preference vector for u_i and item characteristic vector for v_j respectively, where K is the number of latent factors. Factorization based collaborative filtering models solve the following problem

$$\min_{\mathbf{U},\mathbf{V}} \sum_{i=1}^{n} \sum_{j=1}^{m} \mathbf{W}_{ij}(\mathbf{R}_{ij} - \mathbf{U}_i \mathbf{V}_j^\top)^2 + \alpha(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2), \tag{1.10}$$

where $\mathbf{U} = [\mathbf{U}_1^\top, \mathbf{U}_2^\top, \ldots, \mathbf{U}_n^\top]^\top \in \mathbb{R}^{n \times k}$ and $\mathbf{V} = [\mathbf{V}_1^\top, \mathbf{V}_2^\top, \ldots, \mathbf{V}_m^\top]^\top \in \mathbb{R}^{m \times K}$. K is the number of latent factors (patterns), which is usually determined via cross-validation. The term $\alpha(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2)$ is introduced to avoid over-fitting, controlled by the parameter α. $\mathbf{W} \in \mathbb{R}^{n \times m}$ is a weight matrix where $\mathbf{W}_{ij}$ is the weight for the rating of user u_i for item v_j. A common way to set $\mathbf{W}$ is $\mathbf{W}_{ij} = 1$ if $\mathbf{R}_{ij} \neq 0$. The weight matrix $\mathbf{W}$ can also be used to handle the implicit feedback and encode side information such as user click behaviors [13], similarity between users and items [51, 33], quality of reviews [54], user reputation [66], and temporal dynamics [28].

1.1.3 Hybrid Recommendation

Each of the two prominent recommendation approaches in this section use a different source of information to generate the recommendation. One method to improve the quality of the recommendation is to use a combination of the approaches [7, 4, 23]. In this section, we introduce methods for combining the results from other systems to improve the quality of recommendation.

- *Monolithic hybridization.* In this approach, there is only one recommendation system. This recommendation system uses different sources and approaches to make the recommendation. So this case is a modification in the base behavior of recommender algorithms to make another one that can use the advantages of all of the other algorithms [23].
- *Parallelized hybridization.* In this approach we use more than one recommender algorithm. The hybridization process gets the results of recommender systems, combines them, and generates the final recommendation. We can use different techniques to combine the results of other recommender systems, such as *mixed* (a union of results from all recommender systems), *weighted* (a weighted combination of the results), *switching* (use results from specific recommender systems for specific tasks), and *majority voting.*
- *Pipeline hybridization.* In this approach, we also use more than one recommender system, and we put the recommender systems in a pipeline.

The result of one recommender is the input for another one. The earlier recommender can make a model of the input and pass it to the next recommender system or can generate a list of recommendations to be used by the next recommender system.

1.1.4 Evaluating Recommender Systems

There are many different recommender systems. To find the best one for a special task, we need to evaluate the results. Recommender systems can be evaluated based on the *accuracy of prediction*, *accuracy of classification*, *accuracy of ranks*, and *acceptance level* [26, 19].

Evaluating the Accuracy of Prediction To evaluate the accuracy, we need to have prediction data as well as ground truth. We get a subset of ratings as the test set, then run the algorithm and get the recommendations for missing user-item values. We can use *Mean Absolute Error (MAE)* or *Root Mean Square Error (RMSE)* to evaluate the accuracy of prediction [23].

$$MAE = \frac{1}{n} \times \sum_{(u,i) \in Testset} |\hat{r}_{u,i} - r_{u,i}| \tag{1.11}$$

MAE is the quantity used to measure the average deviation between a recommender system's output $\hat{r}_{u,i}$ and the actual rating values $r_{u,i}$ for all evaluated users from the test set of size n. Comparing with MAE, RMSE puts more stress on the deviations.

$$RMSE = \sqrt{\frac{\sum_{(u,i) \in Testset} (\hat{r}_{u,i} - r_{u,i})^2}{n}} \tag{1.12}$$

To remove the effect of the different ranges of ratings, we can normalize MAE by dividing it by $|r_{max} - r_{min}|$. The new measure is called *normalized MAE (NMAE)*, is calculated as follows.

$$NMAE = \frac{MAE}{r_{max} - r_{min}} \tag{1.13}$$

Evaluating the Accuracy of Classification We can use precision and recall to evaluate the algorithm for top-k recommendation. *Precision* is the number of correct results divided by the number of all returned results and tells us how accurate the prediction is.

$$Precision = \frac{\text{number of correctly recommended items}}{\text{total number of recommended items}} \tag{1.14}$$

Recall is the number of correct results divided by the number of results that should have been returned and tells us how complete the prediction is.

$$Recall = \frac{\text{number of correctly recommended items}}{\text{total number of relevant items}} \tag{1.15}$$

By using the *F-measure*, the evaluation results would be more comparable.

$$F_1 = \frac{2 \cdot p \cdot r}{p + r}, \tag{1.16}$$

where p and r are precision and recall, respectively.

Evaluating the Accuracy of Ranks Precision and recall define the accuracy of prediction. But ranking the scores could be used to evaluate the recommender systems in a finer level of granularity. This lets us consider the position of the recommended item in the final ranking along with the prediction results. We can use different functions to evaluate the accuracy of ranks based on this method. The following equations show an example that decreases the utility value of a recommendation based on the position of the item in the recommended item list [6].

$$rankScore_u = \sum_{i \in rec_u} (2^{\frac{rank(i)-1}{\alpha}})^{-1} \tag{1.17}$$

$$rankScore_u^{max} = \sum_{i \in test_u} (2^{\frac{index(i)-1}{\alpha}})^{-1} \tag{1.18}$$

$$rankScore'_u = \frac{rankScore_u}{rankScore_u^{max}}, \tag{1.19}$$

where rec_u denotes the list of recommended items, $rank(i)$ denotes the position of item i in the list, and $index(i)$ denotes the real position of the item i in the list. The parameter α is a half-life utility which means that an item at position α generates twice the utility as an item at position $\alpha + 1$.

Evaluating the Acceptance Level Evaluating the acceptance level of the recommendation can be more directly measure in industry through the following: *click-through rate*, *usage*, *return rate*, and *profit*, by checking whether the customer accepted the recommendation, bought the product, and did not return the product.

1.1.5 Challenges of Recommender Systems

Though, recommender systems have been very successful in past, they also encounter some challenges. Recommender systems, in particular, collaborative filtering based recommender systems, face three major challenges: *cold start*, *data sparsity*, and *attacks*.

1.1.5.1 Cold Start

Cold start is the potential problem for any data driven system, including recommender systems that try to build a model based on the existing information. Cold start is the situation that the algorithm's effectiveness is very low because items' (or users') vector do not have enough rated items to find vectors similar to them[59]. In the content-based approach, the systems must be able to match items' characteristics against relevant features from users' profiles. Therefore, it needs to construct a detailed model of users' tastes and preferences. Therefore, without having a sufficiently detailed model of users' tastes and preferences, the system would fail to match it with the appropriate items and consequently to make a commendation for users. In the collaborative filtering approach, the recommender system identifies users who share similar preferences with the active user, and recommends items which the like-minded users favored (and the active user has not yet seen). Due to the cold start problem, this approach would fail to consider items which no one in the system has rated previously. The cold start problem can be mitigated by applying hybrid approaches such as a combination of content-based and collaborative filtering approaches.

1.1.5.2 Data Sparsity

The core of many recommender systems is to find similar users or similar items. Though there are many algorithms that can solve this problem, almost all of them fail when the size of the vectors grows and passes some thresholds [8]. When the number of users or items increases, the rating matrix becomes extremely sparse. For example, IMDB has records of more than 700K movies. Even if somebody can see and rank one thousand of them, the rating matrix remains extremely sparse. In these situations, finding similar users becomes extremely difficult, and most of the existing algorithms fail to find similar users or items. One common technique to handle this problem is using *factorization methods* to reduce the size of the rating matrix and create a matrix with less number of more relevant and independent features. However, handling extremely sparse rating matrices remains an open challenge for recommender systems [56].

1.1.5.3 Attacks

Attacks are designed to drive the recommender system act in a way that the attacker wishes. It could either recommend some desired items or prevent recommending of other items. An attack against a recommender system consists of a set of attack profiles, and each contains biased rating data associated with a fictitious user identity and a target item. There are two major different types of attacks: *Push attack* or *Nuke attack* [29, 10]. In a push attack, the recommender recommends whatever the attacker wishes. The attacker creates a profile whose rating is similar to that of the target user. Given the high sim-

ilarity between the target user and the injected profile, the system will more likely use the injected profile as a source to provide recommendation to the target user. The attacker in a nuke attack aims to prevent the system from recommending a particular item or to make the recommender system unusable as a whole.

1.1.6 Summary

We have discussed two popular types of recommender systems, content-based and collaborative filtering systems. Although these approaches are highly successful and are frequently used in the real-world recommender systems, they have their limitations when handling challenges such as *cold start*, *data sparsity*, and *attacks*. There are many ongoing research projects to address these challenges. For example, using *trust information* has gained more attention from both academia and industry researchers. Next, we first introduce trust in social networks and then approaches of trust-aware recommendation systems.

1.2 Computational Models of Trust in Recommender Systems

Trust plays an important role across many disciplines, and forms an important feature of our everyday lives [39]. In addition, trust is a property that associates with the relations between people in the real world as well as users in social media. In the previous chapter, we describe the challenges of the existing recommender systems including *cold start*, *data sparsity*, and *attacks*. It has been shown that, by using trust information, we can mitigate some of the challenges. In this section, we provide a brief review of trust and solutions which we can use to measure trust among social media users in general, and users of recommender systems in particular. This trust information will be used as a base to improve classical recommender systems and construct trust-aware recommender systems, which will be covered in the next section in detail. We start the section by defining trust and its properties. The section also focuses on measuring trust values between users including both explicit and implicit trust values. Propagation and aggregation techniques of trust values are the last important concepts which we cover in this section.

1.2.1 Definition and Properties

Trust, in social context, usually refers to a situation where one party (*trustor*) is willing to rely on the actions of another party (*trustee*). It is

also attributed to relationships between people or within and between social groups, such as families, friends, communities, organizations, or companies [32]. In recommender systems, it is defined based on the other users' ability to provide valuable recommendations [18]. In this work, we adopt the definition given in [46]: *"the willingness of a party to be vulnerable to the actions of another party based on the expectation that the other will perform a particular action important to the trustor, irrespective of the ability to monitor or control that other party."*

In this chapter, we consider trust that forms only among the users of a system, and do not consider other forms of trust such as trust between users and items or users and communities.

1.2.1.1 Notations

In our settings, we use $T(u, v)$ to show there is a trust relation between the truster, u, and the trustee, v, and we use $t_{u,v}$ to indicate the value of the trust between them. The trust value can be either binary or real numbers (e.g., in the range of $[0, 1]$). Binary trust is the simplest way of expressing trust. Either two users trust each other or not. A more complicated method is continuous trust model, which assigns real values to the trust relations. In both binary trust and continuous models, 0 and 1 mean *no trust* and *full trust*, respectively. When we consider distrust, we have to add negative values of trust in range $[-1, 0)$. Therefore, we represent trust (and distrust) as a continuous variable over the range $[-1, +1]$. Many applications, such as Amazon[4] and eBay[5], use binary trust values. We use $\mathbf{T} \in \mathbb{R}^{n \times n}$ to denote the matrix representation of trust values for a system with n users. In this perspective, trust values are assigned to the relations. Moreover, we can think of trust as a property of users, or nodes in social media. In this situation, every user has a global trust value representing the users' overall trustworthiness. We use t_u to represent the global trust to user u. We use $\mathbf{t} \in \mathbb{R}^n$ to denote the vector representation of global trust values for a system with n users.

1.2.1.2 Trust Networks

A trust network is a weighted and directed graph in which nodes are users and edges are the trust between the users. In a trust network, trust is information about the social relationships and it represents as a label for the links. The strength of the edges shows the amount of trust between the users in two ends of every edge. Figure 1.3 represents a trust network.

[4] www.amazon.com
[5] www.ebay.com

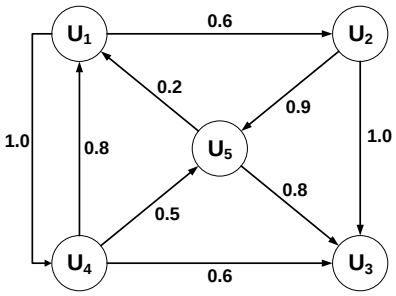

FIGURE 1.3: A trust network

The matrix representation of the trust network is given in Equation 1.20.

$$\mathbf{T} = \begin{bmatrix} 0 & 0.6 & 0 & 1.0 & 0 \\ 0 & 0 & 0.9 & 0 & 1.0 \\ 0.2 & 0 & 0 & 0 & 0.8 \\ 0.8 & 0 & 0.5 & 0 & 0.6 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.20}$$

1.2.1.3 Properties of Trust

In this section, we introduce properties of trust including: *transitivity*, *asymmetry*, *context dependence*, and *personalization* [9, 15, 18]. These properties are extracted from the definition of trust and provide the basis for the creation of algorithms that utilize trust information.

- **Transitivity** Transitivity is a key property of trust. It allows trust to be propagated along paths to reach other users. Based on the transitivity effect, if u trusts v and v trusts w, it can be inferred that u might also trust w to some extent,

$$T(u,v) \ \wedge \ T(v,w) \Rightarrow T(u,w), \tag{1.21}$$

  where $T(u,v)$ indicates trust relation between users u and v, and $t_{u,v} > 0$.

- **Composability** Transitivity describes how trust information propagates from one user to another one through a chain of connected users. Composability describes how a user should combine the different trust values received from different paths. There are different approaches we can use to aggregate trust values from different paths where the simplest way is to, first sum them up, and then normalize the values based on the predefined ranges of trust values. Figure 1.5 shows the case that user *Alice* receives trust values from two different paths.

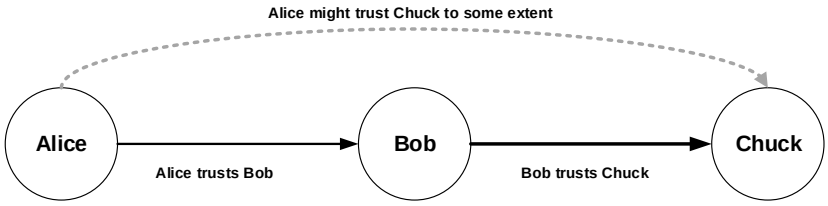

FIGURE 1.4: Transitivity of trust: Bob tells Alice that she can trust Chuck.

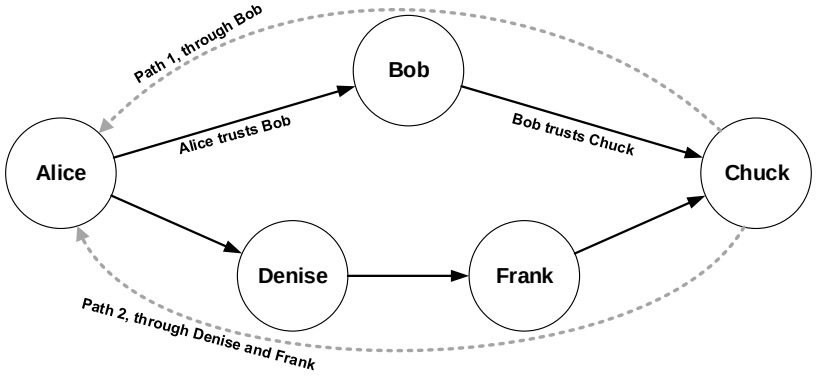

FIGURE 1.5: Composability of trust: there are two paths that Alice can use to infer trust information about Chuck. Then she aggregates the trust values from both paths.

- **Asymmetry** Trust is a subjective and personal relation between users; therefore, it creates a directed relation in social networks. In other words, if $t_{u,v}$ represents the value of trust from user u to user v, it might not necessarily be equal to the value of trust from user v to user u. Based on this attribute, user u trusting user v does not guarantee that user v also trusts user u to the same extent,

$$t_{u,v} \neq t_{v,u}. \tag{1.22}$$

- **Context Dependence** Trust is context dependent. This means that trusting someone on one topic does not guarantee trusting him in other topics. For example, a user who is trustworthy in technology might not be also trustworthy in astronomy, as well,

$$T_i(u,v) \nRightarrow T_j(u,v), \tag{1.23}$$

  where $T_i(u,v)$ indicates the existence of trust between users u and v with respect to topic i.

- **Personalization** Trust is a personalized attribute. The amount of trust one person has in another, might vary between different people. We use this property to define and formulate local trust. But when we define global trust, which is equivalent to user reputation, we violate this property. In global trust every user has only one trust value that can be used by all other users in the network. We describe global trust values in detail in Section 1.2.2,

$$t_{u,v} \neq t_{w,v}. \tag{1.24}$$

1.2.2 Global and Local Trust Metrics

Based on our earlier definition, trust is a relation between two people and trust values indicate how much one person trusts the another. Trust metrics are the methods for measuring and calculating the value of trust between users in the network. From this point of view, trust is a local attribute [44]. In addition to this local view, we can also look at trust as a global measure, where every user has a global value that indicates his/her trustworthiness in the network as a whole [42]. In this section, we elaborate on these two views of trust.

Local Trust Metrics Local trust metrics use the subjective opinion of the current user to predict the trustworthiness of other users from the current user's perspective. The trust value represents the amount of trust that the active user puts on another user. Based on this approach, different users trust the current user differently and therefore their trust value are different from each other $t_{iu} \neq t_{ju}$. Consider a network of n users. In a local trust metric, every user evaluates all other $n-1$ users and assigns them a value that represents their trustworthiness. Therefore the local trust metrics are defined as $T : U \times U \rightarrow [0, 1]$.

Global Trust Metrics Global trust metrics represent the whole community's opinion regarding the current user; therefore, every user receives only one value that represents her level of trustworthiness in the community. Trust scores in global trust metrics are calculated by the aggregation of all users' opinions regarding to the current user. Users' reputations on ebay.com is an example of using global trust in an online shopping website. ebay.com calculates user reputation based on the number of users who left positive, negative, or neutral feedback for the items sold by the current user. When the user does not have a specific opinion regarding another user, she usually relies on this aggregated trust scores. The global trust metrics are defined as $T : U \rightarrow [0, 1]$.

Global trust can be further classified into two classes of *profile-level* and *item-level* trust. The profile-level trust refers to the general definition of global trust metrics in which it assigns one trust score to every user. In item-level

trust, every user might have a different trust score for different items. This supports the third property of trust, which is context dependence.

Comparison between Local and Global Trust Metrics Global metrics are computationally less expensive and simpler than local metrics. However, local models can handle controversial topics and resistance against attacks. In some cases, such as controversial topics, it is difficult to reach an agreement among the users regarding one specific user. In these cases, there are groups of people who like and trust one specific user, and many others dislike or distrust the user. These are situations that it is highly recommended to use local trust metrics, if applicable. In a local setting, trust only propagates through users who are explicitly being trusted by the current user. Therefore, fake profiles cannot exert their influence on the final propagation and trust scores of other users. As a result, compared to global trust metrics, local trust metrics are shown to be more resistant against the attacks [44, 16]. Calculating local trust is computationally more expensive than global trust scores, as it should be calculated for every pair of users. However, they might represent a user's interests more accurately and precisely than global approach. It is notable that a majority of trust metrics proposed in literature are global and only a few used local metrics.

1.2.3 Inferring Trust Values

In a network of users, it is possible to ask users to evaluate other users and assign each user a value that indicates their trustworthiness. Although this setting is feasible in the real world, every user can only evaluate a few other users. This might be either due to users' time limitation or because a user only has a direct opinion about a few other users, e.g., on epinions.com, the average size of trusted users for every node is 1.7 [63, 43, 15, 2]. To expand the coverage of users' trust network, computational tools are designed to predict the trustworthiness of unknown users. These methods aim to predict the trust scores for almost every pair of the nodes in the network based on the existing trust information. By using trust inference algorithms we can accurately calculate how much a person should trust another person in the network.

We are interested to know if one can use available trust information to infer trust values between other pairs of the nodes in the network. Given a network of users [50], transitivity is the common assumption in all of these solutions for propagating and infering trust in the network. Based on transitivity, if user u trusts user v and user v trusts user w, we can conclude that user u might also trust user w to some extent. Trust inference techniques can be used to overcome the sparsity and cold start problems in the network.

The following are examples of how we use transitivity in everyday conversation:

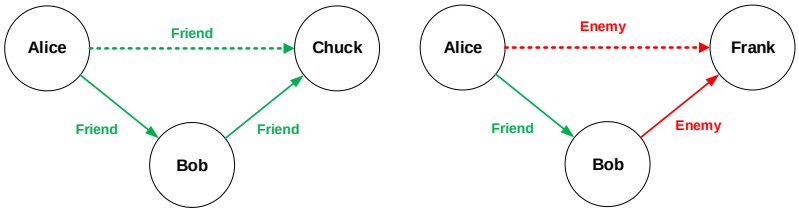

FIGURE 1.6: Trust inference is to use transitivity to calculate the trust values between users without explicit trust values.

- M friend's friend is my friend (left graph in Figure 1.6).
- My enemy's enemy is my friend (right graph Figure 1.6).

In the following section we introduce algorithms to infer trust between users in social networks. All of the algorithms use paths in the network to propagate and infer trust values. We introduce solutions to infer binary trust values as well as continuous trust values.

1.2.3.1 Inferring Trust in Binary Trust Networks

In a binary trust network, nodes are rated as either "trusted" or "not trusted". In this section, we present a simple voting algorithm to infer binary trust values. In Figure 1.7, the goal is to infer $t_{s,t}$, where s denotes the source, t denotes the target node, and $t_{s,t}$ denotes the trust value from s to t. In this algorithm, the source asks all of its "trusted" neighbors to return their ratings for the target node. The source does not ask "not trusted" neighbors, as their ratings would be unreliable for him. If the neighbor is connected to the target node, it simply returns its own trust value (either trusted or not trusted). Otherwise, it repeats the same process by asking from neighboring nodes. When a node receives more than one rating from neighbors, it averages the ratings and passes the result to the requester. In the next step, the node can either round the final ratings and pass binary trust values (as $\{0, 1\}$) or pass the real number value (in range $[0, 1]$). Finally, the source takes the average of the values received from different paths and round them. The final rating value is either 1 or 0, representing "trusted" or "not trusted", respectively.

In the graph illustrated in Figure 1.7, user S asks B and C for recommendations about T, but ignores D as he does not trust D. Neither B nor C are directly connected to T, therefore, they ask E and F for recommendation. Following the algorithm described above, B receives one "positive" and one "negative" recommendations and C receives one "positive" recommendation. The final result is one positive recommendation from C to S. Therefore, S's evaluation for T is "Trust".

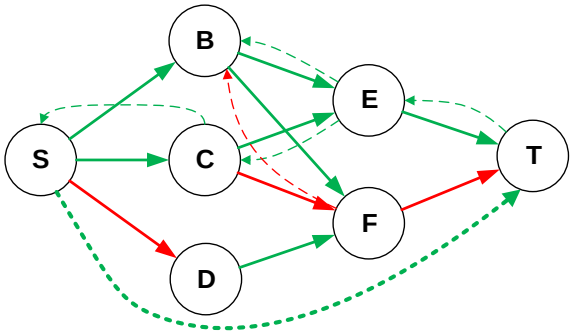

FIGURE 1.7: Inferring trust in a binary trust network

1.2.3.2 Inferring Trust in Continuous Trust Networks

In a network with continuous trust values, taking a simple average of all trust values is the most intuitive approach to infer trust values. In this approach, the source node polls each of its neighbors to obtain their ratings of the sink. Each neighbor repeats the same process. Every time a node receives ratings from more than one neighbor, it uses the average value of the ratings. In this approach, we do not consider the length of paths and weights of each neighbor, and treat them all equally. This approach is usually used as a baseline for other solutions on inferring trust in continues trust networks.

$$t_{u,v} = \frac{\sum\limits_{i \in adj(u)} t_{u,i} t_{i,v}}{\sum\limits_{i \in adj(u)} t_{u,i}}, \tag{1.25}$$

where $adj(u)$ denotes a set of users adjacent to u and $t_{u,i}$ denotes the trust between users u and v.

Trust propagation in a path [74] When two users are connected through other users located in a path, we can use propagation models to calculate trust between them. For example, in Figure 1.8 there is no direct connection between users u and w, however, they are connected through user v. We can compute the trust values along the path u, v, w as follows:

$$t_{u,w} = t_{u,v} \times t_{v,w}. \tag{1.26}$$

In general, for a path of length n trust can be computed as follows:

$$t_{u,w} = \prod_{(a_k,a_l) \in \text{ path } (u,w)} t_{a_k,a_l}, \tag{1.27}$$

where $t_{u,v}$ is the trust value between users u and v and $0 \leq t_{u,v} \leq 1$. Note that

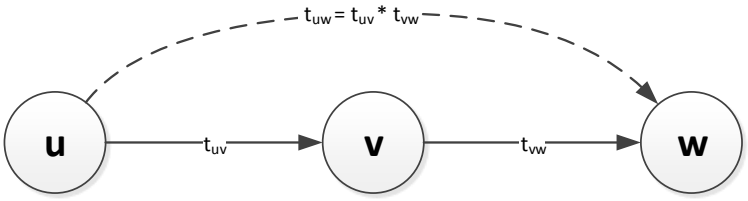

FIGURE 1.8: Trust propagation in a path

we cannot use this technique to propagate distrust. In addition to the simple average approach, we introduce some representative approaches for computing indirect trust as follows: *EigenTrust*, *Trust Propagation*, *Appleseed* and *Matrix Factorization based methods*.

EigenTrust EigenTrust algorithm is a reputation management algorithm for social networks. The algorithm provides each user in the network a unique global trust value based on the user's history of interaction between the users [25]. This approach relies on computing the principal eigenvector of the trust network to calculate trust values. The result of the algorithm is a vector of trust values, representing how much trust the source node should have for other nodes in the network. The values are ranks of the trustworthiness of individuals, not recommended trust values. The biggest challenge with this algorithm is that it needs to convert these ranks into trust values that can be assigned to the relations.

Trust Propagation In [17], a trust propagation framework is proposed with four atomic propagations - direct propagation, co-citation, transpose trust, and trust coupling as:

- if u_i trusts u_j, and u_j trusts u_k, direct propagation allows us to infer that u_i trusts u_k, and its corresponding operator is $\mathbf{T}$;
- co-citation propagation concludes that u_ℓ should trust u_j if u_i trusts u_j and u_k, and u_ℓ trust u_k. $\mathbf{T}^\top\mathbf{T}$ denotes the operator of co-citation propagation;
- in transpose trust, u_i's trust of u_j causes u_j to develop some level of trust towards u_i. The operator is shown by $\mathbf{T}^\top$.
- trust coupling suggests that u_i and u_j trust u_k, so trusting u_i should imply trusting u_j. $\mathbf{T}\mathbf{T}^\top$ denotes the operator.

$\mathbf{C}$ is defined as a single combined matrix of all four atomic propagations,

$$\mathbf{C} = \alpha_1\mathbf{T} + \alpha_2\mathbf{T}^\top\mathbf{T} + \alpha_3\mathbf{T}^\top + \alpha_4\mathbf{T}\mathbf{T}^\top, \tag{1.28}$$

where α_1, α_2, α_3, and α_4 control contributions from direct propagation, co-citation, transpose trust and trust coupling, respectively.

Let $\mathbf{C}^k$ be a matrix where $\mathbf{C}^k_{ij}$ denotes the propagation from u_i to u_j after k atomic propagations. Then, the final estimated matrix representation of the user-user trust relation $\tilde{\mathbf{T}}$ is given by [17],

$$\tilde{\mathbf{T}} = \sum_{k=1}^{K} \gamma^k \mathbf{C}^k, \tag{1.29}$$

where K denotes the number of steps of propagation and γ^k denotes a discount factor to penalize lengthy propagation steps.

Appleseed The Appleseed algorithm proposed by Ziegler and Lausen in 2004 [78]. It models trust as energy and injects it from the source node to all of the other nodes in the network along edges. The energy is fully divided among the successor nodes with respect to their local trust scores. In the Appleseed algorithm, the closer the sink is to the source and the more paths leading from the source to the node, the higher energy reaches to the sink.

Matrix factorization based method [63] A few factors can influence the establishment of trust relations. A user usually establishes trust relations with a small proportion of $\mathcal{U}$, resulting in very sparse and low-rank $\mathbf{T}$; hence, users can have a more compact but accurate representation in a low-rank space [68]. The matrix factorization model seeks a low-rank representation $\mathbf{U} \in \mathbb{R}^{n \times d}$ with $d \ll n$ for $\mathcal{U}$ via solving the following optimization problem,

$$\min_{\mathbf{U},\mathbf{V}} \quad \|\mathbf{T} - \mathbf{U}\mathbf{V}\mathbf{U}^\top\|_F^2, \tag{1.30}$$

where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix and $\mathbf{V} \in \mathbb{R}^{d \times d}$ captures the correlations among their low-rank representations such as $\mathbf{T}_{ij} = \mathbf{U}_i \mathbf{V} \mathbf{U}_j^\top$. It is easy to verify that Eq. (1.30) can model the properties of trust such as transitivity and asymmetry [68].

To avoid overfitting, we add two smoothness regularization components on $\mathbf{U}$ and $\mathbf{V}$, respectively, in Eq. (1.30),

$$\min_{\mathbf{U},\mathbf{V}} \quad \|\mathbf{T} - \mathbf{U}\mathbf{V}\mathbf{U}^\top\|_F^2 + \alpha\|\mathbf{U}\|_F^2 + \beta\|\mathbf{V}\|_F^2, \tag{1.31}$$

where α and β are non-negative, and are introduced to control the capability of $\mathbf{U}$ and $\mathbf{V}$, respectively. With the learned $\mathbf{U}$ and $\mathbf{V}$, the estimated matrix representation of the user-user trust relation $\tilde{\mathbf{T}}$ is obtained as $\tilde{\mathbf{T}} = \mathbf{U}\mathbf{V}\mathbf{U}^\top$.

1.2.3.3 Inferring Implicit Trust Values

In implicit trust models, there is no trust information in the system, and the goal is to use other information to construct a network with trust values

between the users. [77] showed that there is a relationship between people's interest similarities and trust between them. [24] found that given some predefined domain and context, people s interest similarity is a strong predictor of interpersonal trust. This means that people who have similar interests tend to be more trustful towards each other. In the light of these studies, it can be concluded that measuring the similarity in interests is a reasonable technique to measure trust among the users.

The models use different network information to calculate the trust between users. In this section we introduce the techniques of *profile similarity*, *rating history*, and *relevance* which received significant attention in the literature.

- **Profile similarity -** Similarity between users (network, content, or interests). If two people are connected, consider this as a sign for their trust. This way trust can be mutual or directed.
- **Rating history -** Calculating the similarity based on the similarity in product ratings. Users with similar ratings and similar tastes are more likely to trust each other.
- **Relevance -** Using the recommendation history to measure relevance based on the accuracy of recommendations. If user u has a record of successful recommendations to user v, this can be used as a sign that v trusts u.

There is a major difference between implicit trust and inferred trust values. In inferring explicit trust, we use transitivity to propagate existing trust information and calculate the trust information for users with missing trust information. Unlike the explicit inference method, in implicit trust inference, we do not use any other trust information in the network.

1.2.3.4 Trust Aggregation

Another aspect of trust inference is aggregating trust values inferred from other nodes. When a user asks for a recommendation and more than one neighbor answers or there is more than one recommendation per neighbor, we need to integrate the recommendations to end up with one value. To make the final choice, we can use any of the following approaches [45]:

- Average to get the average of all of the recommendations,
- Frequency-based approach to get the trust value with the highest frequency,
- Shortest path to rank recommendations based on the length of the path to the target user (i.e., shorter paths will get higher weight),
- Top-Friends suggestion to give higher score to the values recommended by closer friends, and

- Hybrid to use a combination of the above approaches.

1.2.4 Summary

In this section, we have discussed trust in social networks. We formally define trust and introduce its properties. We provide an overview on measuring explicit and implicit trust in the network and discuss algorithms for inferring trust in explicit networks. In the next section, we introduce techniques of using trust formation to improve recommender systems.

1.3 Incorporating Trust in Recommender Systems

Users can be influenced by their trustworthy friends, and are more likely to accept recommendations made by their trusted friends than recommendations from strangers. Studies also confirm that people tend to rely on recommendation from their friends and other people they trust more than those provided by recommender systems in terms of quality and usefulness, even though the recommendations given by the recommender systems have high novelty factor [61]. *Trust-aware recommender systems* employ trust information to enhance recommender systems. Merging trust information and recommender systems can improve the accuracy of recommendation as well as the users' experience.

In addition to these improvements, which will be discussed in this section, trust-aware recommender systems are capable to handle some of the challenges we encounter for classical recommender systems such as cold-start problem and responding to attacks. Trust metrics and recommendation technologies are two basis for trust-aware recommender systems [73]. The focus of the

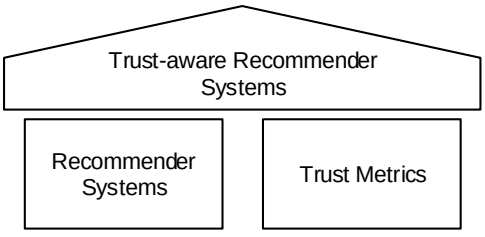

FIGURE 1.9: A trust-aware recommender system has two pillars, recommender systems and trust metrics

section is describing techniques to combine trust information with classical recommender systems. As mentioned earlier in Sections 1.3.1 and 1.3.2, most

of the trust-aware recommender systems are based on collaborative filtering approach.

Trust-aware recommender systems, utilize trust information to generate more personalized recommendations, and users receive recommendations from those who are in their trust networks (web of trust). Trust-aware recommender systems emerge from social networks where there are information about the ties between users. This information can be used to construct a user-user similarity bases. This extra information will be added to traditional recommender systems to improve the quality of the recommendation, hoping to improve the *accuracy* or *coverage*, and address some of the challenges from classical recommender systems. In a trust-aware recommender system, trust information can be used in one of the following approaches along with classical recommender systems.

- *Trust-aware memory-based CF systems:* Systems of this group use memory-based CF techniques as their basic methods.
- *Trust-aware model-based CF systems:* Systems of this group use model-based CF techniques as their basic methods.

FilmTrust and *epinions.com* are two examples of trust-aware recommender systems.

FilmTrust [14] is an example of trust-aware recommender system. The system is an online social network that combines trust, social networks, and movie ratings and reviews to generate a personalized recommendation. In FilmTrust, personalization of the website, displaying movie ratings, and ordering the reviews are based on the trust and rating information. There is a social network and for each connection, users rate how much they trust their friend's opinion about movies. The following sentence from help section of the systems, would help users on rating their friends: *"Think of this as if the person were to have rented a movie to watch, how likely it is that you would want to see that film."* Users can evaluate their friends' movie taste on a scale from 1 (low trust) to 10 (high trust). But these numbers remain private and other users can not see the actual values.

Users also rate movies and write reviews as well. These ratings are on a scale of a half star to four stars. When a user asks for a recommendation, it shows two sets of information for every movie, the overall average rating, as well as the recommended rating calculated using trust values as weights. TilmTrust, uses the trust information to weight the ratings of users in the network (similar to trust-based weighted mean).

www.epinions.com[6] is a general consumer review site. Users in this product review site can share their reviews about products. Also they can establish

[6] http://www.epinions.com

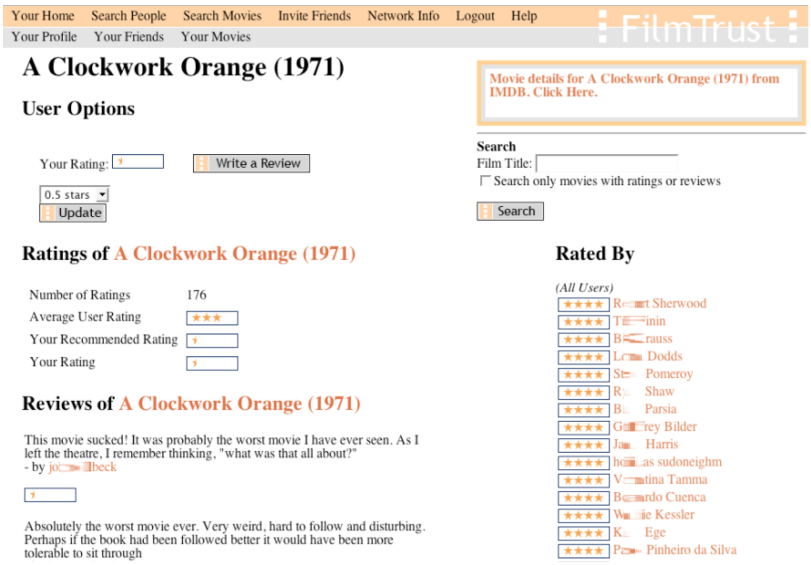

FIGURE 1.10: FilmTrust, an example of trust-aware recommender system.

their trust networks from which they may seek advice to make decisions. People not only write critical reviews for various products but also read and rate the reviews written by others. Furthermore, people can add members to their trust networks or "Circle of Trust", if they find their reviews consistently interesting and helpful.

1.3.1 Trust-aware Memory-based CF Systems

A different approach from traditional memory-based collaborative filtering is to focus on friends more than strangers. In this approach, the system incorporates trust information to boost recommendations from trusted users and depress recommendations from other users. The recommender system, uses trust information to either weight the recommendation made be all users or filter non-trusted users.

1.3.1.1 Trust-Aware Filtering

Pure collaborative approaches use all available data (users or items) to make recommendations regardless of the importance of each of them to the receiver of the recommender. In trust-aware filtering, we use trust information as a tool to filter some of the users, then we use recommendations provided by trusted users. After filtering the extra users, we use classical recommendations approaches to recommend items. Therefore, trust-aware filtering model only accepts the recommendations made by trusted users. This approach is more

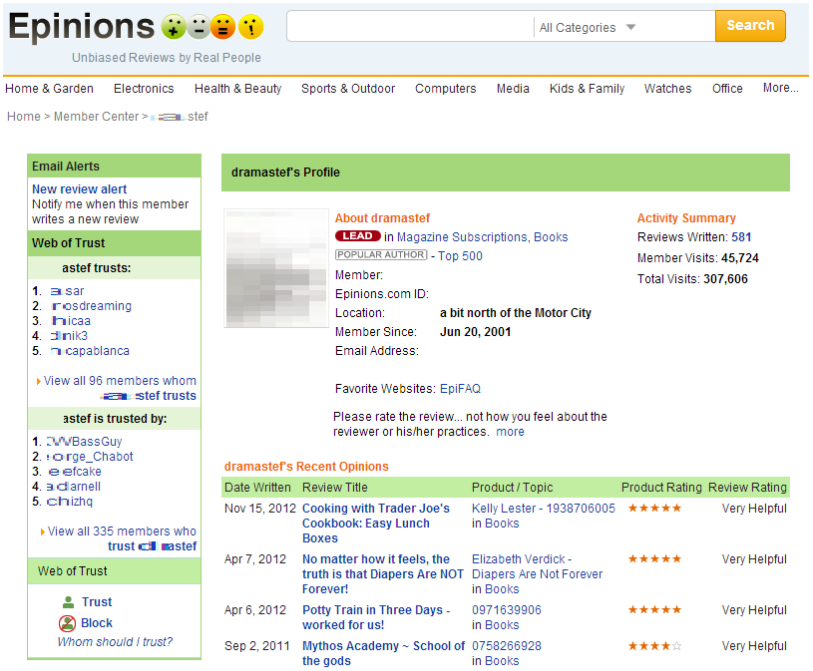

FIGURE 1.11: www.epinions.com, an example of trust-aware recommender system.

effective when the binary values of trust are available. In the case that we have continues trust values, the system filters users that the current user does not trust them or the trust value is less than a threshold value denoted by τ as follows:

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{t_{u,v}>\tau} (r_{v,i} - \bar{r}_v)}{\sum_{t_{u,v}>\tau} r_{v,i}}, \tag{1.32}$$

where $\bar{r}_u$ denotes the average rating for user u.

1.3.1.2 Trust-Aware Weighting

Despite the trust-aware filtering that excludes non-trusted users, trust-aware weighting uses trust information to weight the recommendations made by all users. In this approach, the system incorporates trust information to boost recommendations from trusted users and depress recommendations from non-trusted users. Among the trusted users, those with higher trust values will have more weight on recommending new items to the user. In trust-aware weighting approach, the recommended ratings $\hat{r}$ for user u and item i

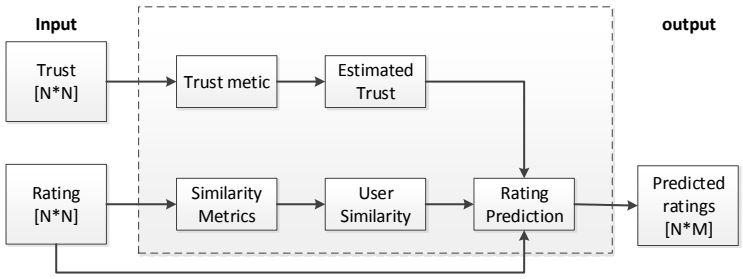

FIGURE 1.12: In trust-aware weighting, the recommender accepts recommendations from all of the users. But it weights the recommendations based on trust values.

is calculated using the following equation,

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum (r_{v,i} - \bar{r}_v) \times t_{u,v}}{\sum t_{u,v}}, \tag{1.33}$$

where $\hat{r}_{u,i}$ denotes user u's predicted rating for item i, and $t_{u,v}$ denotes the trust value from user u to user v. Another approach is to use a combination of filtering and weighting approaches as follows:

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{t_{u,v}>\tau} (r_{v,i} - \bar{r}_v) \times t_{u,v}}{\sum_{t_{u,v}>\tau} t_{u,v}}. \tag{1.34}$$

Rest of this section introduces some of the representative trust-aware memory-based recommender systems.

TidalTrust TidalTrust [15] computes trust scores by performing a modified breadth first search in trust network. In this algorithm, the goals is to compute the trust values between a source and a sink node, $t_{source,sink}$. The algorithm starts with the source node, and finds all the paths to the sink. Every path has two properties: its length and its strength. Strength shows the trust score of the source and the current node which is equal to the minimum trust scores of all edges located on the path. Among the paths, those with shortest path length will be selected. This algorithm performs in two steps. First, it finds all raters with the shortest path distance from the source user to the sink. Then, it aggregates their ratings weighted by the trust between the user and the raters. The following two principles are features integrated into TidalTrust:

- For a fixed trust value, shorter paths have lower error.
- For a fixed path length, higher trusts have lower error.

MoleTrust MoleTrust is a local trust metric proposed by Massa and Avesani

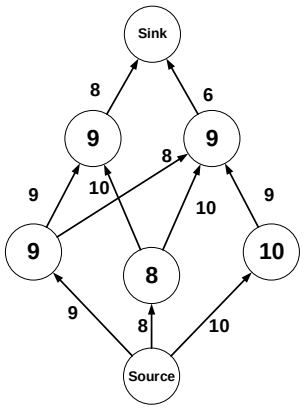

FIGURE 1.13: TidalTrust [15], finds the shortest path to the sink and returns the trust value from the path with maximum smallest trust value on the path from source to sink. In this figure, the value of trust from user t_{source} to user t_{sink} is 8.

[44]. The idea of MoleTrust is similar to TidalTrust, except that the depth of the search path for rater can be up to a *maximum-depth*. To calculate the trust scores, the algorithm performs a backward exploration.

$$t_u = \frac{\sum_{i \in predecessors} (t_i \times t_{i,u})}{\sum_{i \in predecessors} (t_i)}, \tag{1.35}$$

where t_i denotes user i's global trust value. In MoleTrust and TidalTrust, if a node does not know the sink, it asks its friends to realize how much to trust the sink. Then, it computes a trust value by a weighted average. If neighbors are directly connected to the sink, they return their ratings, otherwise they repeat same process by asking their own neighbors.

$$t_{i,s} = \frac{\sum\limits_{j \ \in \ adj(i) \ | \ t_{i,j} \ \geq \ max} t_{i,j} t_{j,s}}{\sum\limits_{j \ \in \ adj(i) \ | \ t_{i,j} \ \geq \ max} t_{i,j}}. \tag{1.36}$$

TrustWalker [21] The intuition of this system is from two key observations. First, a user's social network has little overlap with similar users, suggesting that social information provides an independent source of information. Second, ratings from strongly trusted friends on similar items are more reliable than ratings from weakly trusted neighbors on the same target item. The first observation indicates the importance of trust-based approaches, while the second observation suggests the capability of item-oriented approaches. To take

advantage of both approaches, TrustWalker proposes a random walk model to combine trust based and user oriented approaches into a coherent framework. It queries a user's direct and indirect friends' ratings for the target item as well as similar items by performing random walk in online social networks. For example, to obtain a rating for u_i to v_j, suppose that we are at a certain node u_k, then TrustWalker works as follows at each step of a random work: if u_k rated v_j, then it stops the random walk and returns $\mathbf{R}_{kj}$ as the result of random walk; otherwise, it has two choices - (1) it also stops random walk and randomly selects one of the items v_k similar to v_j rated by u_i and returns $\mathbf{R}_{ik}$, or (2) it continues random walk and walks to another user u_k in u_i's trust networks. TrustWalker employs the Pearson Correlation Coefficients of ratings expressed for items to calculate item-item similarity. Since values of Pearson Correlation Coefficients are in the range of $[-1, 1]$, only items with positive correlation with the target item are considered. The similarity between v_i and v_j is then computed as follows:

$$sim(i,j) = \frac{1}{1 + e^{-\frac{N_{ij}}{2}}} \times PCC(i,j), \tag{1.37}$$

where N_{ij} denotes the number of users who rated both v_i and v_j, and $PCC(i,j)$ denotes the Pearson Correlation Coefficient of v_i and v_j.

1.3.2 Trust-Aware Model-based CF Systems

Model-based social recommender systems choose model-based CF techniques as their basic models. Matrix factorization techniques are widely used in model based CF methods. The common rationale behind these solutions is that users' preferences are similar to or influenced by their trusted users. Systems in this class can be further divided into three groups [67]: co-factorization methods, ensemble methods, and regularization methods. Next, we review some representative systems in detail for each group.

Co-factorization methods [37, 63]: The underlying assumption of systems in this group is that the i-th user u_i should share the same user preference vector $\mathbf{u}_i$ in the rating space (rating information) and the trust relation space. Systems in this group perform a co-factorization in the user-item matrix and the user-user trust relation matrix by sharing the same user preference latent factor. For example, the representative system SoRec [37] learns the user preference matrix $\mathbf{U}$ from both rating information and trust information by solving the following optimization problem:

$$\begin{aligned}\min_{\mathbf{U},\mathbf{V},\mathbf{Z}} \|\mathbf{W} \odot (\mathbf{R} - \mathbf{U}^\top \mathbf{V})\|_F^2 + \alpha \sum_{i=1}^{n} \sum_{u_k \in \mathcal{N}_i} (\mathbf{T}_{ik} - \mathbf{u}_i^\top \mathbf{z}_k)^2 \\ + \lambda(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2 + \|\mathbf{Z}\|_F^2).\end{aligned} \tag{1.38}$$

The reconstructed matrix $\hat{\mathbf{T}}$ can be used to perform trust relation prediction [47, 63]. Therefore, one advantage of approaches in this group is that they jointly perform recommendation and trust relation prediction.

Ensemble methods [35, 64]: The basic idea of ensemble methods is that users and their trust networks should have similar ratings on items, and a missing rating for a given user is predicted as a linear combination of ratings from the user and her trust network. *STE [35]* is a representative system in this group, which models a rating from the i-th user u_i to the j-th item v_j as follows:

$$\hat{\mathbf{R}}_{ij} = \mathbf{u}_i^\top \mathbf{v}_j + \beta \sum_{u_k \in \mathcal{N}_i} \mathbf{T}_{ik} \mathbf{u}_k^\top \mathbf{v}_j, \tag{1.39}$$

where $\sum_{u_k \in \mathcal{N}_i} \mathbf{T}_{ik} \mathbf{u}_k^\top \mathbf{v}_j$ is a weighted sum of the predicted ratings for v_j from u_i's trust network, and β controls the influence from trust information. STE is finally to minimize the following term,

$$\|\mathbf{W} \odot ((\mathbf{R} - \mathbf{U}^\top \mathbf{V}) - \beta \mathbf{T} \mathbf{U}^\top \mathbf{V}))\|_F^2 \tag{1.40}$$

Regularization methods [22, 38]: Regularization methods focus on a user's preferences and assume that a user's preferences should be similar to that of her trust network. For a given user u_i, regularization methods force her preferences $\mathbf{u}_i$ to be closer to that of users in u_i's trust network $\mathcal{N}_i$. SocialMF [22] forces the preferences of a user to be closer to the average preference of the user's trust network as,

$$\min \sum_{i=1}^{n} (\mathbf{u}_i - \sum_{u_k \in \mathcal{N}_i} \mathbf{T}_{ik} \mathbf{u}_k)^2, \tag{1.41}$$

where $\sum_{u_k \in \mathcal{N}_i} \mathbf{T}_{ik} \mathbf{u}_k$ is the weighted average preference of users in u_i's trust network $\mathcal{N}_i$, and SocialMF requires each row of $\mathbf{S}$ to be normalized to 1. The authors demonstrated that SocialMF addresses the transitivity of trust in trust networks because a user's latent feature vector is dependent on the direct neighbors' latent feature vectors which can propagate in the entire network. In this case, the user's latent feature vector is dependent on possibly all users in the network. With the term to capture trust information, SocialMF solves the following optimization problem:

$$\begin{aligned} \min_{\mathbf{U},\mathbf{V}} \|\mathbf{W} \odot (\mathbf{R} - \mathbf{U}^\top \mathbf{V})\|_F^2 + \alpha \sum_{i=1}^{n} (\mathbf{u}_i - \sum_{u_k \in \mathcal{N}_i} \mathbf{T}_{ik} \mathbf{u}_k)^2 \\ + \lambda(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2) \end{aligned} \tag{1.42}$$

One advantage of these approaches is that they indirectly model the propagation of tastes in social networks, which can be used to mitigate cold-start problem and increase the coverage of items for recommendations.

1.3.3 Recommendation Using Distrust Information

Distrust information considers disagreements between users. While trust information shows to what degree one user likes recommendations made by another user, distrust information shows the opposite. By considering distrust information, recommender systems can avoid getting recommendations from users that the current user distrusts. Distrust information also helps recommender systems learn about the items that the user does not like. This comes from the idea that if a user does not trust another user, s/he does not like items recommended by the user. So the recommender system will avoid recommending those items as well. There are two strategies to exploit distrust information for recommendation including *distrust as filter* and *distrust as an indicator to reverse deviations* [71].

Distrust as a filter: This strategy is to use distrust to filter out "unwanted" individuals from collaborative recommendation processes as follows:

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{v \setminus \mathcal{D}} (r_{v,i} - \bar{r}_v) \times t_{u,v}}{\sum t_{u,v}}, \tag{1.43}$$

where $\mathcal{D}$ is the set of distrusted users.

Distrust as an indicator to reverse deviations: This strategy directly incorporates distrust into the recommendation process by considering distrust scores as negative weights as follows:

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{v \setminus \mathcal{D}} (r_{v,i} - \bar{r}_v) \times t_{u,v}}{\sum t_{u,v}} - \frac{\sum_{v \in \mathcal{D}} (r_{v,i} - \bar{r}_v) \times d_{u,v}}{\sum d_{u,v}}, \tag{1.44}$$

where $d_{u,v}$ is the distrust strength between u and v.

1.3.4 Advantages of Trust-Aware Recommendation

The main idea behind trust-aware recommendation is to use trust information to improve the quality of recommendation and to address some of the challenges from classical recommender systems specially the cold-start problem and their weakness against attacks. In this section, we introduce some of the advantages of using trust-aware recommender systems.

- *Mitigating the Cold-start problem.* New users have only a few or even no ratings. The classical recommender systems may fail to do recommendations for these users. However, trust-aware recommender systems can make recommendations as long as a new user establishes trust relations with other users. [40] showed that trust-aware recommendation yields more accurate predictions for cold-start users, compared to a classical collaborative filtering system.

- *Robust to attacks.* By using trust information, we can filter most fake profiles, as no one willfully trusts a fake profile. In this case, the system will not use these users in recommendation.

- *Increasing the coverage of recommendation.* In classical collaborative filtering, we only have access to immediate neighbors who have similarly rated a product. But trust-aware recommender systems use more information to construct the similarity matrix. Therefore, we have access to more users [21]. In addition, trust information increases the coverage of a collaborative filtering system, while maintaining the accuracy [41].

1.3.5 Research Directions of Trust-Aware Recommendation

In this section, we discuss several research directions that can potentially improve the capabilities of trust-aware recommender systems, and make social recommendations applicable to an even broader range of applications.

Temporal Information: Customer preferences for products drift over time. For example, people interested in "Electronics" at time t may shift their preferences to "Sports" at time $t + 1$. Temporal information is an important factor in recommender systems. Some of the traditional recommender systems already considered temporal information [12, 27]. Temporal dynamics in data can have a more significant impact on accuracy than designing more complex learning algorithms [27]. Exploiting temporal information for recommender systems is still challenging due to the complexity of users' temporal patterns [27].

Trust relations also vary over time. For example, there might be case where new trust relations are added, while existing trust relations become inactive or are deleted. The variation of both ratings and trust relations further exacerbate the difficulty of exploiting temporal information for social recommendation. A preliminary study of the impact of the changes in both ratings and trust relations on recommender systems demonstrates that temporal information can benefit trust recommendation [65].

Distrust in Recommendation: Currently most existing trust-aware recommender systems use trust relations. However, distrust in recommendation is still in the early stages of development and an active area of exploration. Authors in [3] found that negative relations such as distrust are even more important than positive relations, revealing the importance of distrust relations for recommendation. There are several works exploiting distrust [36, 72] in social recommender systems. They treat trust and distrust separately, and simply use distrust in an opposite way to trust, such as filtering distrusted users or considering distrust relations as negative weights. However, trust and distrust are shaped by different dimensions of trustworthiness. Further, trust affects behavior intentions differently from distrust [11]. Furthermore, distrust relations are not independent of trust relations [73]. A deeper understanding of distrust and its relations with trust can help to develop efficient trust-aware recommender systems by exploiting both trust and distrust.

1.4 Conclusion

Recommender systems aim to overcome the information overload problem in a way that covers users' preferences and interests. Trust-aware recommender systems are an attempt to incorporate trust information into classical recommender systems to improve their recommendations and address some of their challenges including the cold start problem and their weakness against the attacks.

Bibliography

[1] Mohammad Ali Abbasi, Sun-Ki Chai, Huan Liu, and Kiran Sagoo. Real-world behavior analysis through a social media lens. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, pages 18–26. Springer, 2012.

[2] Mohammad Ali Abbasi and Huan Liu. Measuring user credibility in social media. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, pages 441–448. Springer, 2013.

[3] Zeinab Abbassi, Christina Aperjis, and Bernardo A Huberman. Friends versus the crowd: tradeoffs and dynamics. *HP Report*, 2013.

[4] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[5] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[6] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

[7] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[8] Robin Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer, 2007.

[9] Christiano Castelfranchi and Rino Falcone. *Trust theory: A socio-cognitive and computational model*, volume 18. John Wiley & Sons, 2010.

[10] Paul-Alexandru Chirita, Wolfgang Nejdl, and Cristian Zamfir. Preventing shilling attacks in online recommender systems. In *Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 67–74. ACM, 2005.

[11] Jinsook Cho. The mechanism of trust and distrust formation and their relational outcomes. *Journal of Retailing*, 82(1):25–35, 2006.

[12] Yi Ding and Xue Li. Time weight collaborative filtering. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 485–492. ACM, 2005.

[13] Yi Fang and Luo Si. Matrix co-factorization for recommendation with rich side information and implicit feedback. In *Proceedings of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, pages 65–69. ACM, 2011.

[14] Jennifer Golbeck and James Hendler. Filmtrust: Movie recommendations using trust in web-based social networks. In *Proceedings of the IEEE Consumer communications and networking conference*, volume 96. Citeseer, 2006.

[15] Jennifer Ann Golbeck. Computing and applying trust in web-based social networks. 2005.

[16] Marco Gori and Ian Witten. The bubble of web visibility. *Communications of the ACM*, 48(3):115–117, 2005.

[17] Ramanthan Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of trust and distrust. In *Proceedings of the 13th international conference on World Wide Web*, pages 403–412. ACM, 2004.

[18] Guibing Guo, Jie Zhang, Daniel Thalmann, Anirban Basu, and Neil Yorke-Smith. From ratings to trust: an empirical study of implicit trust in recommender systems. 2014.

[19] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[20] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):89–115, 2004.

[21] Mohsen Jamali and Martin Ester. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 397–406. ACM, 2009.

[22] Mohsen Jamali and Martin Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 135–142. ACM, 2010.

[23] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender systems: an introduction*. Cambridge University Press, 2010.

[24] Carlos Jensen, John Davis, and Shelly Farnham. Finding others online: reputation systems for social online spaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 447–454. ACM, 2002.

[25] Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, pages 640–651. ACM, 2003.

[26] Joseph A Konstan. Introduction to recommender systems: Algorithms and evaluation. *ACM Transactions on Information Systems (TOIS)*, 22(1):1–4, 2004.

[27] Yehuda Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.

[28] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[29] Shyong K Lam and John Riedl. Shilling recommender systems for fun and profit. In *Proceedings of the 13th international conference on World Wide Web*, pages 393–402. ACM, 2004.

[30] Daniel Lemire and Anna Maclachlan. Slope one predictors for online rating-based collaborative filtering. *Society for Industrial Mathematics*, 5:471–480, 2005.

[31] Daniel Lemire and Anna Maclachlan. Slope one predictors for online rating-based collaborative filtering. In *SDM*, volume 5, pages 1–5. SIAM, 2005.

[32] J David Lewis and Andrew Weigert. Trust as a social reality. *Social forces*, 63(4):967–985, 1985.

[33] Yanen Li, Jia Hu, ChengXiang Zhai, and Ye Chen. Improving one-class collaborative filtering by incorporating rich user information. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 959–968. ACM, 2010.

[34] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[35] Hao Ma, Irwin King, and Michael R Lyu. Learning to recommend with social trust ensemble. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 203–210. ACM, 2009.

[36] Hao Ma, Michael R Lyu, and Irwin King. Learning to recommend with trust and distrust relationships. In *Proceedings of the third ACM conference on Recommender systems*, pages 189–196. ACM, 2009.

[37] Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. Sorec: social recommendation using probabilistic matrix factorization. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 931–940. ACM, 2008.

[38] Hao Ma, Dengyong Zhou, Chao Liu, Michael R Lyu, and Irwin King. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296. ACM, 2011.

[39] Stephen Paul Marsh. Formalising trust as a computational concept. 1994.

[40] P. Massa and P. Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 17–24. ACM, 2007.

[41] Paolo Massa and Paolo Avesani. Trust-aware collaborative filtering for recommender systems. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, pages 492–508. Springer, 2004.

[42] Paolo Massa and Paolo Avesani. Controversial users demand local trust metrics: An experimental study on epinions. com community. In *Proceedings of the National Conference on artificial Intelligence*, volume 20, page 121. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[43] Paolo Massa and Paolo Avesani. *Trust-aware recommender systems*. PhD thesis, 2007.

[44] Paolo Massa and Paolo Avesani. Trust metrics on controversial users: Balancing between tyranny of the majority. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 3(1):39–64, 2007.

[45] Paolo Massa and Bobby Bhattacharjee. Using trust in recommender systems: an experimental analysis. In *Trust Management*, pages 221–235. Springer, 2004.

[46] Roger C Mayer, James H Davis, and F David Schoorman. An integrative model of organizational trust. *Academy of management review*, 20(3):709–734, 1995.

[47] Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. In *Machine Learning and Knowledge Discovery in Databases*, pages 437–452. Springer, 2011.

[48] Bamshad Mobasher, Robin Burke, and Jeff J Sandvig. Model-based collaborative filtering as a defense against profile injection attacks. In *AAAI*, volume 6, page 1388, 2006.

[49] Raymond J Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 195–204. ACM, 2000.

[50] John O'Donovan and Barry Smyth. Trust in recommender systems. In *Proceedings of the 10th international conference on Intelligent user interfaces*, pages 167–174. ACM, 2005.

[51] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *Eighth IEEE International Conference on Data Mining*, pages 502–511. IEEE, 2008.

[52] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[53] David M Pennock, Eric Horvitz, Steve Lawrence, and C Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory-and model-based approach. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 473–480. Morgan Kaufmann Publishers Inc., 2000.

[54] Sindhu Raghavan, Suriya Gunasekar, and Joydeep Ghosh. Review quality aware collaborative filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 123–130. ACM, 2012.

[55] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[56] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, DTIC Document, 2000.

[57] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[58] J Ben Schafer, Joseph Konstan, and John Riedi. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166. ACM, 1999.

[59] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002.

[60] Bracha Shapira. *Recommender systems handbook*. Springer, 2011.

[61] Rashmi R Sinha and Kirsten Swearingen. Comparing recommendations made by online systems and friends. In *DELOS workshop: personalisation and recommender systems in digital libraries*, volume 106, 2001.

[62] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[63] Jiliang Tang, Huiji Gao, Xia Hu, and Huan Liu. Exploiting homophily effect for trust prediction. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 53–62. ACM, 2013.

[64] Jiliang Tang, Huiji Gao, and Huan Liu. mTrust: Discerning multi-faceted trust in a connected world. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 93–102. ACM, 2012.

[65] Jiliang Tang, Huiji Gao, Huan Liu, and Atish Das Sarma. etrust: Understanding trust evolution in an online world. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 253–261. ACM, 2012.

[66] Jiliang Tang, Xia Hu, Huiji Gao, and Huan Liu. Exploiting local and global social context for recommendation. In *IJCAI*, 2013.

[67] Jiliang Tang, Xia Hu, and Huan Liu. Social recommendation: a review. *Social Network Analysis and Mining*, 3(4):1113–1133, 2013.

[68] Jiliang Tang and Huan Liu. Coselect: Feature selection with instance selection for social media data. In *SDM*, 2013.

[69] Loren Terveen and Will Hill. Beyond recommender systems: Helping people help each other. *HCI in the New Millennium*, 1:487–509, 2001.

[70] Shari Trewin. Knowledge-based recommender systems. *Encyclopedia of library and information science*, 69(Supplement 32):69, 2000.

[71] Patricia Victor, Chris Cornelis, Martine De Cock, and Ankur Teredesai. Trust-and distrust-based recommendations for controversial reviews. In *Web Science Conference (WebSci'09: Society On-Line)*, number 161, 2009.

[72] Patricia Victor, Chris Cornelis, Martine De Cock, and Ankur M Teredesai. A comparative analysis of trust-enhanced recommenders for controversial items. In *Proc. of the International AAI Conference on Weblogs and Social Media*, pages 342–345, 2009.

[73] Patricia Victor, Martine De Cock, and Chris Cornelis. Trust and recommendations. In *Recommender systems handbook*, pages 645–675. Springer, 2011.

[74] Frank Edward Walter, Stefano Battiston, and Frank Schweitzer. A model of a trust-based recommendation system on a social network. *Autonomous Agents and Multi-Agent Systems*, 16(1):57–74, 2008.

[75] Jun Wang, Arjen P De Vries, and Marcel JT Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 501–508. ACM, 2006.

[76] Reza Zafarani, Mohammad Ali Abbasi, and Huan Liu. *Social Media Mining: An Introduction*. Cambridge University Press, 2014.

[77] Cai-Nicolas Ziegler. Investigating correlations of trust and interest similarity-do birds of a feather really flock together? *submitted to the Journal of Artificial Intelligence Research*, 2005.

[78] Cai-Nicolas Ziegler. *Towards decentralized recommender systems.* PhD thesis, Citeseer, 2005.