Foundations and Trends® in
Human–Computer Interaction
Vol. 4, No. 2 (2010) 81–173

DOI: 10.1561/1100000009

now
the essence of knowledge

Collaborative Filtering Recommender Systems

By Michael D. Ekstrand, John T. Riedl
and Joseph A. Konstan

Contents

Foundations and Trends® in
Human–Computer Interaction
Vol. 4, No. 2 (2010) 81–173

DOI: 10.1561/1100000009

now
the essence of knowledge

Collaborative Filtering Recommender Systems

Michael D. Ekstrand[1], John T. Riedl[2] and Joseph A. Konstan[3]

University of Minnesota, 4-192 Keller Hall, 200 Union St., Minneapolis, MN 55455, USA
[1] *ekstrand@cs.umn.edu;* [2] *riedl@cs.umn.edu;* [3] *konstan@cs.umn.edu*

Abstract

Recommender systems are an important part of the information and e-commerce ecosystem. They represent a powerful method for enabling users to filter through large information and product spaces. Nearly two decades of research on collaborative filtering have led to a varied set of algorithms and a rich collection of tools for evaluating their performance. Research in the field is moving in the direction of a richer understanding of how recommender technology may be embedded in specific domains. The differing personalities exhibited by different recommender algorithms show that recommendation is not a one-size-fits-all problem. Specific tasks, information needs, and item domains represent unique problems for recommenders, and design and evaluation of recommenders needs to be done based on the user tasks to be supported. Effective deployments must begin with careful analysis of prospective users and their goals. Based on this analysis, system designers have a host of options for the choice of algorithm and for its embedding in the surrounding user experience. This paper discusses a wide variety of the choices available and their implications, aiming to provide both practicioners and researchers with an introduction to the important issues underlying recommenders and current best practices for addressing these issues.

1

Introduction

Every day, we are inundated with choices and options. What to wear? What movie to rent? What stock to buy? What blog post to read? The sizes of these decision domains are frequently massive: Netflix has over 17,000 movies in its selection [15], and Amazon.com has over 410,000 titles in its Kindle store alone [7]. Supporting discovery in information spaces of this magnitude is a significant challenge. Even simple decisions — what movie should I see this weekend? — can be difficult without prior direct knowledge of the candidates.

Historically, people have relied on recommendations and mentions from their peers or the advice of experts to support decisions and discover new material. They discuss the week's blockbuster over the water cooler, they read reviews in the newspaper's entertainment section, or they ask a librarian to suggest a book. They may trust their local theater manager or news stand to narrow down their choices, or turn on the TV and watch whatever happens to be playing.

These methods of recommending new things have their limits, particularly for information discovery. There may be an independent film or book that a person would enjoy, but no one in their circle of acquaintances has heard of it yet. There may be a new indie band in another city whose music will likely never cross the local critic's

radar. Computer-based systems provide the opportunity to expand the set of people from whom users can obtain recommendations. They also enable us to mine users' history and stated preferences for patterns that neither they nor their acquaintances identify, potentially providing a more finely-tuned selection experience.

There has been a good deal of research over the last 20 years on how to automatically recommend things to people and a wide variety of methods have been proposed [1, 140]. Recently, the *Recommender Systems Handbook* [122] was published, providing in-depth discussions of a variety of recommender methods and topics. This survey, however, is focused primarily on *collaborative filtering*, a class of methods that recommend items to users based on the preferences other users have expressed for those items.

In addition to academic interest, recommendation systems are seeing significant interest from industry. Amazon.com has been using collaborative filtering for a decade to recommend products to their customers, and Netflix valued improvements to the recommender technology underlying their movie rental service at $1M via the widely-publicized Netflix Prize [15].

There is also a growing interest in problems surrounding recommendation. Algorithms for understanding and predicting user preferences do not exist in a vacuum — they are merely one piece of a broader user experience. A recommender system must interact with the user, both to learn the user's preferences and provide recommendations; these concerns pose challenges for user interface and interaction design. Systems must have accurate data from which to compute their recommendations and preferences, leading to work on how to collect reliable data and reduce the noise in user preference data sets. Users also have many different goals and needs when they approach systems, from basic needs for information to more complex desires for privacy with regards to their preferences.

In his keynote address at the 2009 ACM Conference on Recommender Systems, Martin [90] argued that the algorithms themselves are only a small part of the problem of providing recommendations to users. We have a number of algorithms that work fairly well, and while there is room to refine them, there is much work to be done on

user experience, data collection, and other problems which make up the whole of the recommender experience.

1.1 History of Recommender Systems

The capacity of computers to provide recommendations was recognized fairly early in the history of computing. Grundy [123], a computer-based librarian, was an early step towards automatic recommender systems. It was fairly primitive, grouping users into "stereotypes" based on a short interview and using hard-coded information about various sterotypes' book preferences to generate recommendations, but it represents an important early entry in the recommender systems space.

In the early 1990s, collaborative filtering began to arise as a solution for dealing with overload in online information spaces. Tapestry [49] was a manual collaborative filtering system: it allowed the user to query for items in an information domain, such as corporate e-mail, based on other users' opinions or actions ("give me all the messages forwarded by John"). It required effort on the part of its users, but allowed them to harness the reactions of previous readers of a piece of correspondence to determine its relevance to them.

Automated collaborative filtering systems soon followed, automatically locating relevant opinions and aggregating them to provide recommendations. GroupLens [119] used this technique to identify Usenet articles which are likely to be interesting to a particular user. Users only needed to provide ratings or perform other observable actions; the system combined these with the ratings or actions of other users to provide personalized results. With these systems, users do not obtain any direct knowledge of other users' opinions, nor do they need to know what other users or items are in the system in order to receive recommendations.

During this time, recommender systems and collaborative filtering became an topic of increasing interest among human–computer interaction, machine learning, and information retrieval researchers. This interest produced a number of recommender systems for various domains, such as Ringo [137] for music, the BellCore Video Recommender [62] for movies, and Jester [50] for jokes. Outside of computer

science, the marketing literature has analyzed recommendation for its ability to increase sales and improve customer experience [10, 151].

In the late 1990s, commercial deployments of recommender technology began to emerge. Perhaps the most widely-known application of recommender system technologies is Amazon.com. Based on purchase history, browsing history, and the item a user is currently viewing, they recommend items for the user to consider purchasing.

Since Amazon's adoption, recommender technology, often based on collaborative filtering, has been integrated into many e-commerce and online systems. A significant motivation for doing this is to increase sales volume — customers may purchase an item if it is suggested to them but might not seek it out otherwise. Several companies, such as NetPerceptions and Strands, have been built around providing recommendation technology and services to online retailers.

The toolbox of recommender techniques has also grown beyond collaborative filtering to include content-based approaches based on information retrieval, bayesian inference, and case-based reasoning methods [132, 139]. These methods consider the actual content or attributes of the items to be recommended instead of or in addition to user rating patterns. Hybrid recommender systems [24] have also emerged as various recommender strategies have matured, combining multiple algorithms into composite systems that ideally build on the strengths of their component algorithms. Collaborative filtering, however, has remained an effective approach, both alone and hybridized with content-based approaches.

Research on recommender algorithms garnered significant attention in 2006 when Netflix launched the Netflix Prize to improve the state of movie recommendation. The objective of this competition was to build a recommender algorithm that could beat their internal CineMatch algorithm in offline tests by 10%. It sparked a flurry of activity, both in academia and amongst hobbyists. The $1 M prize demonstrates the value that vendors place on accurate recommendations.

1.2 Core Concepts, Vocabulary, and Notation

Collaborative filtering techniques depend on several concepts to describe the problem domain and the particular requirements placed

on the system. Many of these concepts are also shared by other recommendation methods.

The information domain for a collaborative filtering system consists of *users* which have expressed preferences for various *items*. A preference expressed by a user for an item is called a *rating* and is frequently represented as a (*User*, *Item*, *Rating*) triple. These ratings can take many forms, depending on the system in question. Some systems use real- or integer-valued rating scales such as 0–5 stars, while others use binary or ternary (like/dislike) scales.[1] Unary ratings, such as "has purchased", are particularly common in e-commerce deployments as they express well the user's purchasing history absent ratings data. When discussing unary ratings, we will use "purchased" to mean that an item is in the user's history, even for non-commerce settings such as web page views.

The set of all rating triples forms a sparse matrix referred to as the *ratings matrix*. (*User*, *Item*) pairs where the user has not expressed a preference for (rated) the item are unknown values in this matrix. Figure 1.1 shows an example ratings matrix for three users and four movies in a movie recommender system; cells marked '?' indicate unknown values (the user has not rated that movie).

In describing use and evaluation of recommender systems, including collaborative filtering systems, we typically focus on two tasks. The first is the *predict* task: given a user and an item, what is the user's likely preference for the item? If the ratings matrix is viewed as a sampling of values from a complete user–item preference matrix, than the predict task for a recommender is equivalent to the matrix missing-values problem.

	Batman Begins	*Alice in Wonderland*	*Dumb and Dumber*	*Equilibrium*
User A	4	?	3	5
User B	?	5	4	?
User C	5	4	2	?

Fig. 1.1 Sample ratings matrix (on a 5-star scale).

[1] The scale is ternary if "seen but no expressed preference" is considered distinct from "unseen".

The second task is the *recommend* task: given a user, produce the best ranked list of n items for the user's need. An n-item recommendation list is not guaranteed to contain the n items with the highest predicted preferences, as predicted preference may not be the only criteria used to produce the recommendation list.

In this survey, we use a consistent mathematical notation for referencing various elements of the recommender system model. The universe consists of a set U of users and a set I of items. I_u is the set of items rated or purchased by user u, and U_i is the set of users who have rated or purchased i. The rating matrix is denoted by $\mathbf{R}$, with $r_{u,i}$ being the rating user u provided for item i, $\mathbf{r}_u$ being the vector of all ratings provided by user u, and $\mathbf{r}_i$ being the vector of all ratings provided for item i (the distinction will be apparent from context). $\bar{r}_u$ and $\bar{r}_i$ are the average of a user u or an item i's ratings, respectively. A user u's preference for an item i, of which the rating is assumed to be a reflection, is $\pi_{u,i}$ (elements of the user-item preference matrix $\mathbf{\Pi}$). It is assumed that $r_{u,i} \approx \pi_{u,i}$; specifically, $\mathbf{R}$ is expected to be a sparse sample of $\mathbf{\Pi}$ with the possible addition of noise. The recommender's prediction of $\pi_{u,i}$ is denoted by $p_{u,i}$.

1.3 Overview

This survey aims to provide a broad overview of the current state of collaborative filtering research. In the next two sections, we discuss the core algorithms for collaborative filtering and traditional means of measuring their performance against user rating data sets. We will then move on to discuss building reliable, accurate data sets; understanding recommender systems in the broader context of user information needs and task support; and the interaction between users and recommender systems.

2

Collaborative Filtering Methods

Collaborative filtering (CF) is a popular recommendation algorithm that bases its predictions and recommendations on the ratings or behavior of other users in the system. The fundamental assumption behind this method is that other users' opinions can be selected and aggregated in such a way as to provide a reasonable prediction of the active user's preference. Intuitively, they assume that, if users agree about the quality or relevance of some items, then they will likely agree about other items — if a group of users likes the same things as Mary, then Mary is likely to like the things they like which she hasn't yet seen.

There are other methods for performing recommendation, such as finding items similar to the items liked by a user using textual similarity in metadata (*content-based filtering* or CBF). The focus of this survey is on collaborative filtering methods, although content-based filtering will enter our discussion at times when it is relevant to overcoming a particular recommender system difficulty.

The majority of collaborative filtering algorithms in service today, including all algorithms detailed in this section, operate by first generating predictions of the user's preference and then produce their recommendations by ranking candidate items by predicted preferences. Often this prediction is in the same scale as the ratings provided by

users, but occasionally the prediction is on a different scale and is meaningful only for candidate ranking. This strategy is analagous to the common information retrieval method of producing relevance scores for each document in a corpus with respect to a particular query and presenting the top-scored items. Indeed, the recommend task can be viewed as an information retrieval problem in which the domain of items (the corpus) is queried with the user's preference profile.

Therefore, this section is primarily concerned with how various algorithms predict user preference. In later sections we will discuss recommendation strategies that diverge from this structure, but in actual implementation they frequently start with a preference-ranked list of items and adjust the final recommendation list based on additional criteria.

2.1 Baseline Predictors

Before discussing true collaborative filtering altorighms, let us first consider some baseline prediction methods. These methods are useful for establishing non-personalized baselines against which personalized algorithms can be compared, as well as for pre-processing and normalizing data for use with more sophisticated algorithms. Baseline algorithms that do not depend on the user's ratings can also be useful for providing predictions for new users. We denote the baseline prediction for user u and item i by $b_{u,i}$.

The simplest baseline is to predict the average rating over all ratings in the system: $b_{u,i} = \mu$ (where μ is the overall average rating). This can be enhanced somewhat by predicting the average rating by that user or for that item: $b_{u,i} = \bar{r}_u$ or $b_{u,i} = \bar{r}_i$.

Baselines can be further enhanced by combining the user mean with the average deviation from user mean rating for a particular item [58, 79, 110, 115]. Generally, a baseline predictor of the following form can be used:

$$b_{u,i} = \mu + b_u + b_i \tag{2.1}$$

b_u and b_i are user and item baseline predictors, respectively. They can be defined simply by using average offsets as follows, computing

subsequent effects within the residuals of previous effects [14, 58, 115]:

$$b_u = \frac{1}{|I_u|} \sum_{i \in I_u} (r_{u,i} - \mu) \tag{2.2}$$

$$b_i = \frac{1}{|U_i|} \sum_{u \in U_i} (r_{u,i} - b_u - \mu) \tag{2.3}$$

The baseline can be further regularized, providing a more reasonable estimate of user and item preferences in the face of sparse sampling, with the incorporation of damping terms β_u and β_i [44][1]:

$$b_u = \frac{1}{|I_u| + \beta_\mathrm{u}} \sum_{i \in I_u} (r_{u,i} - \mu) \tag{2.4}$$

$$b_i = \frac{1}{|U_i| + \beta_\mathrm{i}} \sum_{u \in U_i} (r_{u,i} - b_u - \mu) \tag{2.5}$$

This adjustment causes the baseline predicted ratings to be closer to global mean when the user or item has few ratings, based on the statistical principle that the more ratings a user has given or an item has received, the more is known about that user or item's true mean rating. Funk [44] found that 25 was a useful value for the damping terms.

Additional baselines can be computed and added, and the baselines can be made more sophisticated to deal with various effects [14, 80, 115]. If an item or user is new and therefore has no ratings, its baseline can be set to 0, effectively assuming that it is an average user or item.

The baseline predictors are not restricted to simple ANOVA-style functions. They can also be learned as more general parameters with gradient descent or other parameter estimation techniques, potentially as a part of learning a larger model [14, 79, 80].

Baseline predictors effectively capture effects of user bias, item popularity, and can be applied to more exotic but increasingly-important factors such as time [80, 115]. If the baseline is subtracted from the ratings matrix to yield a normalized ratings matrix $\hat{\mathbf{R}}$, all that remains

[1] Funk's formulation involved an additional term in the numerator; since we are computing mean offsets rather than means, that term cancels out.

for collaborative filtering to do is to efficiently capture the interaction effect between users and items. Further, the missing values of $\hat{\mathbf{R}}$ are 0 rather than unknown, simplifying some computations and allowing the matrix to be handled by standard sparse matrix packages.

2.2 User–User Collaborative Filtering

User–user collaborative filtering, also known as *k-NN collaborative filtering*, was the first of the automated CF methods. It was first introduced in the GroupLens Usenet article recommender [119]. The Ringo music recommender [137] and the BellCore video recommender [62] also used user-user CF or variants thereof.

User–user CF is a straightforward algorithmic interpretation of the core premise of collaborative filtering: find other users whose past rating behavior is similar to that of the current user and use their ratings on other items to predict what the current user will like. To predict Mary's preference for an item she has not rated, user–user CF looks for other users who have high agreement with Mary on the items they have both rated. These users' ratings for the item in question are then weighted by their level of agreement with Mary's ratings to predict Mary's preference.

Besides the rating matrix $\mathbf{R}$, a user–user CF system requires a similarity function $s: U \times U \to \mathbb{R}$ computing the similarity between two users and a method for using similarities and ratings to generate predictions.

2.2.1 Computing Predictions

To generate predictions or recommendations for a user u, user–user CF first uses s to compute a neighborhood $N \subseteq U$ of neighbors of u. Once N has been computed, the system combines the ratings of users in N to generate predictions for user u's preference for an item i. This is typically done by computing the weighted average of the neighboring users' ratings i using similarity as the weights:

$$p_{u,i} = \bar{r}_u + \frac{\sum_{u' \in N} s(u,u')(r_{u',i} - \bar{r}_{u'})}{\sum_{u' \in N} |s(u,u')|} \tag{2.6}$$

Subtracting the user's mean rating $\bar{r}_u$ compensates for differences in users' use of the rating scale (some users will tend to give higher ratings than others). Equation (2.6) can also be extended to normalize user ratings to z-scores by dividing the offset from mean rating by the standard deviation σ_u of each user's ratings, thereby compensating for users differing in rating spread as well as mean rating [58]:

$$p_{u,i} = \bar{r}_u + \sigma_u \frac{\sum_{u' \in N} s(u,u')(r_{u',i} - \bar{r}_{u'})/\sigma_{u'}}{\sum_{u' \in N} |s(u,u')|} \tag{2.7}$$

Weighted averaging is not the only mechanism that has been used for computing predictions, but it is the most common. Ringo used no weighting, performing an unweighted average over ratings by neighboring users [137]. The BellCore system used a multivariate regression over the users in the neighborhood to generate predictions [62]. Weighted averaging is by far the most common, however, as it is simple and works well in practice. It is also consistent with Social Choice theory [111], grounding it in models of human behavior and an axiomatic, first-principles development of collaborative filtering. Social choice theory deals with the preferences of individuals and of a society as a whole. Given several properties that a recommender should exhibit within the social choice framework, Pennock et al. show that weighted averaging is the only aggregation method which produces consistent results.

There remains the question of how many neighbors to select. In some systems, such as the original GroupLens, $N = U \backslash \{u\}$ (all users are considered as neighbors); in other systems, neighborhoods are selected for each item based on a similarity threshold or neighborhood size such that N_i is the k users most similar to u who have rated the target item i. Limiting neighborhood size can result in more accurate predictions, as the neighbors with low correlation introduce more noise than signal into the process [58]. The particular threshold to use is domain- and system-specific, so analysis of a relevant data set is needed to choose the neighborhood size for a particular deployment. In offline analysis of available movie ratings data, Herlocker et al. found $k = 20$ to be a good value; values in the 20–50 range are a reasonable starting point in many domains. Lathia et al. [86] propose dynamically adapting the neighborhood size used for each user to minimize the error in their

predictions as new data is added to the system. Randomly sampling candidate users for the neighborhood can decrease the time required to find neighbors, at the possible expense of accuracy [62].

2.2.2 Computing User Similarity

A critical design decision in implementing user–user CF is the choice of similarity function. Several different similarity functions have been proposed and evaluated in the literature.

Pearson correlation. This method computes the statistical correlation (Pearson's r) between two user's common ratings to determine their similarity. GroupLens and BellCore both used this method [62, 119]. The correlation is computed by the following:

$$s(u,v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_u \cap I_v} (r_{v,i} - \bar{r}_v)^2}}$$

Pearson correlation suffers from computing high similarity between users with few ratings in common. This can be alleviated by setting a threshold on the number of co-rated items necessary for full agreement (correlation of 1) and scaling the similarity when the number of co-rated items falls below this threshold [58, 59]. Experiments have shown a threshold value of 50 to be useful in improving prediction accuracy, and the threshold can be applied by multiplying the similarity function by $\min\{|I_u \cap I_v|/50, 1\}$.

Constrained Pearson correlation. Ringo solicited ratings from its users on a 7-point scale, providing a rating guide that fixed 4 as a neutral (neither like nor dislike) value r_z. With an absolute reference, it is possible to correlate absolute like/dislike rather than relative deviation (as the standard Pearson r does). This led Shardanand and Maes [137] to propose the constrained Pearson correlation:

$$s(u,v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - r_z)(r_{v,i} - r_z)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{u,i} - r_z)^2} \sqrt{\sum_{i \in I_u \cap I_v} (r_{v,i} - r_z)^2}}$$

Spearman rank correlation. The Spearman rank correlation coefficient is another candidate for a similarity function [58]. For the Spearman correlation, the items a user has rated are ranked such that their highest-rated item is at rank 1 and lower-rated items have higher ranks. Items with the same rating are assigned the average rank for their position. The computation is then the same as that of the Pearson correlation, except that ranks are used in place of ratings.

Cosine similarity. This model is somewhat different than the previously described approaches, as it is a vector-space approach based on linear algebra rather than a statistical approach. Users are represented as $|I|$-dimensional vectors and similarity is measured by the cosine distance between two rating vectors. This can be computed efficiently by taking their dot product and dividing it by the product of their L_2 (Euclidean) norms:

$$s(u,v) = \frac{\mathbf{r}_u \cdot \mathbf{r}_v}{\|\mathbf{r}_u\|_2 \|\mathbf{r}_v\|_2} = \frac{\sum_i r_{u,i} r_{v,i}}{\sqrt{\sum_i r_{u,i}^2} \sqrt{\sum_i r_{v,i}^2}}$$

Unknown ratings are considered to be 0; this causes them to effectively drop out of the numerator. If the user mean baseline is subtracted from the ratings prior to computing the similarity, cosine similarity is equivalent to Pearson correlation when the users have rated the same set of items and decreases as $\frac{|I_u \cap I_v|^2}{|I_u||I_v|}$ decreases.

The threshold-based damping described for Pearson correlation can, in principle, be applied to any similarity function, although the benefits of doing so have not been studied. An alternative method for damping is to add a large constant, such as 100, to the denominator; this attenuates the similarity of users with small sets of co-rated items.

Other similarity functions, such as mean-squared difference [137], have been proposed, but have not seen significant adoption. In general, Pearson correlation has been found to provide the best results [22, 58], although results from the Ringo system suggest that constrained Pearson correlation may provide some improvement when items are rated on an absolute scale [137].

	Batman Begins	*Alice in Wonderland*	*Dumb and Dumber*	*Equilibrium*
User A	4	?	3	5
User B	?	5	4	?
User C	5	4	2	?
User D	2	4	?	3
User E	3	4	5	?

Fig. 2.1 Example ratings matrix (on a 5-star scale).

2.2.3 Example

Consider the ratings matrix in Figure 2.1 (an extension of the sample matrix in Figure 1.1). We want to find User C's prediction for *Equilibrium* ($p_{C,e}$) with the following configuration:

- Pearson correlation.
- Neighborhood size of 2.
- Weighted average with mean offset (Equation (2.6)).

C's mean rating is 3.667. There are only two users who have rated *Equilibrium*, and therefore only two candidate users for the neighborhood: A and D. $s(C,A) = 0.832$ and $s(C,D) = -0.515$. The prediction $p_{C,e}$ is therefore computed as follows:

$$\begin{aligned} p_{C,e} &= \bar{r}_C + \frac{s(C,A)(r_{A,e} - \bar{r}_A) + s(C,D)(r_{D,e} - \bar{r}_D)}{|s(C,A)| + |s(C,D)|} \\ &= 3.667 + \frac{0.832 \cdot (5-4) + -0.515 \cdot (2-3)}{0.832 + 0.515} \\ &= 4.667 \end{aligned}$$

2.3 Item–Item Collaborative Filtering

User–user collaborative filtering, while effective, suffers from scalability problems as the user base grows. Searching for the neighbors of a user is an $O(|U|)$ operation (or worse, depending on how similarities are computing — directly computing most similarity functions against all other users is linear in the total number of ratings). To extend collaborative filtering to large user bases and facilitate deployment on e-commerce sites, it was necessary to develop more scalable

algorithms. *Item–item collaborative filtering*, also called *item-based* collaborative filtering, takes a major step in this direction and is one of the most widely deployed collaborative filtering techniques today.

Item–item collaborative filtering was first described in the literature by Sarwar et al. [130] and Karypis [71], although a version of it seems to have been used by Amazon.com at this time [87]. Rather than using similarities between users' rating behavior to predict preferences, item–item CF uses similarities between the rating patterns of items. If two items tend to have the same users like and dislike them, then they are similar and users are expected to have similar preferences for similar items. In its overall structure, therefore, this method is similar to earlier content-based approaches to recommendation and personalization, but item similarity is deduced from user preference patterns rather than extracted from item data.

In its raw form, item–item CF does not fix anything: it is still necessary to find the most similar items (again solving the k-NN problem) to generate predictions and recommendations. In a system that has more users than items, it allows the neighborhood-finding to be amongst the smaller of the two dimensions, but this is a small gain. It provides major performance gains by lending itself well to pre-computing the similarity matrix.

As a user rates and re-rates items, their rating vector will change along with their similarity to other users. Finding similar users in advance is therefore complicated: a user's neighborhood is determined not only by their ratings but also by the ratings of other users, so their neighborhood can change as a result of new ratings supplied by any user in the system. For this reason, most user–user CF systems find neighborhoods at the time when predictions or recommendations are needed.

In systems with a sufficiently high user to item ratio, however, one user adding or changing ratings is unlikely to significantly change the similarity between two items, particularly when the items have many ratings. Therefore, it is reasonable to pre-compute similarities between items in an item–item similarity matrix. The rows of this matrix can even be truncated to only store the k most similar items. As users change ratings, this data will become slightly stale, but the users will

likely still receive good recommendations and the data can be fully updated by re-computing the similarities during a low-load time for the system.

Item–item CF generates predictions by using the user's own ratings for other items combined with those items' similarities to the target item, rather than other users' ratings and user similarities as in user–user CF. Similar to user–user CF, the recommender system needs a similarity function, this time $s: I \times I \to \mathbb{R}$, and a method to generate predictions from ratings and similarities.

2.3.1 Computing Predictions

In real-valued ratings domains, the similarity scores can be used to generate predictions using a weighted average, similar to the procedure used in user–user CF. Recommendations are then generated by picking the candidate items with the highest predictions.

After collecting a set S of items similar to i, $p_{u,i}$ can be predicted as follows [130]:

$$p_{u,i} = \frac{\sum_{j \in S} s(i,j) r_{u,j}}{\sum_{j \in S} |s(i,j)|} \tag{2.8}$$

S is typically selected to be the k items most similar to j that u has also rated for some neighborhood size k. Sarwar et al. [130] found $k = 30$ produced good results on the MovieLens data set.

Equation (2.8) as it stands suffers from two deficiencies. The first comes to light when it is possible for similarity scores to be negative and ratings are constrained to be nonnegative: some of the ratings averaged together to compute the prediction may be negative after weightings. While this will not affect the relative ordering of items by predicted value, it will bias the predicted values so they no longer map back to the user rating domain. This can be corrected either by thresholding similarities so only items with nonnegative similarities are considered or by averaging distance from the baseline predictor:

$$p_{u,i} = \frac{\sum_{j \in S} s(i,j)(r_{u,j} - b_{u,i})}{\sum_{j \in S} |s(i,j)|} + b_{u,i} \tag{2.9}$$

The other difficulty is with non-real-valued ratings scales, particularly the unary scales common on e-commerce sites without ratings data. In this case, the averaging does not work: if all similarities are positive and $r_{u,i} = 1$ if user u has purchased item i, then Equation (2.8) always evaluates to 1. With negative similarities, it is similarly ill-behaved. To work around this, we can compute pseudo-predictions $\tilde{p}_{u,i}$ with a simple aggregation of the similarities to items in the user's purchase history I_u. Summation has been tested and shown to perform well (Equation (2.10)); other aggregations such as mean or max may also be considered or used in practice.

$$\tilde{p}_{u,i} = \sum_{j \in I_u} s(i,j) \tag{2.10}$$

$\tilde{p}_{u,i}$ is not in a meaningful scale to predict any particular user behavior, but the predict task is typically not as important in unary contexts. This pseudo-prediction can, however, be used to rank candidate items for recommendation, forming a good basis for using item–item CF to recommend items based on user purchase histories [38, 71, 87].

It is also possible to use weights other than the similarity function for computing the final prediction. Bell and Koren [14] proposed a formulation that computes item weights directly by estimating, for each user–item pair u, i, the solution to the linear equation $\mathbf{Aw} = \mathbf{b}$. The solution $\mathbf{w}$ is such that w_j is the optimal weight to use for u's rating of j in computing their rating of i, and $\mathbf{A}$ and $\mathbf{b}$ are given as follows:

$$a_{j,k} = \sum_{v \neq u} \pi_{v,j} \pi_{v,k} \tag{2.11}$$

$$b_j = \sum_{v \neq u} \pi_{v,j} \pi_{v,i} \tag{2.12}$$

The computed weights, differing for each user–item pair, are then used to compute the prediction $p_{u,i} = \sum_{j \in S} w_j r_{u,j}$.

2.3.2 Computing Item Similarity

The item–item prediction process requires an item–item similarity matrix $\mathbf{S}$. This matrix is a standard sparse matrix, with missing values

being 0 (no similarity); it differs in this respect from $\mathbf{R}$, where missing values are unknown.

As in user–user collaborative filtering, there are a variety of methods that can be used for computing item similarities.

Cosine similarity. Cosine similarity between item rating vectors is the most popular similarity metric, as it is simple, fast, and produces good predictive accuracy.

$$s(i,j) = \frac{\mathbf{r}_i \cdot \mathbf{r}_j}{\|\mathbf{r}_i\|_2 \|\mathbf{r}_j\|_2}$$

Conditional probability. For domains with unary ratings (such as shopping site purchase histories), Karypis [71] proposed a similarity function based on conditional probabilities: $s(i,j) = \Pr_B[j \in B | i \in B]$ where B is a user's purchase history. With a scaling parameter α to balance for frequently occuring items, this formula becomes

$$s(i,j) = \frac{\text{Freq}(i \wedge j)}{\text{Freq}(i)(\text{Freq}(j))^\alpha}$$

α serves as a damping factor to compensate for $\Pr_B[j \in B | i \in B]$ being high solely due to a high marginal probability $\Pr_B[j \in B]$ (if j is an item that many people purchase, it will be similar to most items; α reduces this effect to bring items that are more uniquely similar to the top). This normalization hinders the ability to generate true predictions, but this is not a problem in unary domains due to the use of pseudo-predictions.

Notably, this function is not symmetric ($s(i_1, i_2)$ may not be the same as $s(i_2, i_1)$). Care must be taken that it is used in the correct orientation.

Pearson correlation. Pearson correlation has also been proposed for item–item recommendation, but does not seem to work as well as cosine similarity [130].

In order to optimize the recommender's performance, it is important to normalize the ratings prior to computing the similarity matrix [14, 130]. In real-valued domains variation in ratings due to user ratings

bias (e.g., two users liking and disliking similar films, but one being a cynic who rates average films 2.5/5 and the other an enthusiast who rates the average film at 4/5) and item bias allows the collaborative filter to focus on the more nuanced differences in user preference for particular items. This can be accomplished by subtracting a baseline predictor from all ratings prior to computing similarities (e.g., compute similarities over ratings $\hat{r}_{u,i} = r_{u,i} - \mu - b_u - b_i$).

When applying cosine similarity in unary ratings domains, it can be useful to normalize each user's ratings vector $\mathbf{r}_u$ to the unit vector prior to computing item similarities. The effect of this adjustment is that users who have purchased fewer items have more impact on the similarity of the items they have purchased than users who have purchased many items [38, 71].

Similarly, normalizing item similarity vectors (rows of $\mathbf{S}$) to unit vectors can be beneficial. This causes items with sparser neighborhoods (fewer similar items) to have more influence in computing the final predictions [38, 71].

2.3.3 Pre-computing and Truncating the Model

Due to the relatively static nature of item similarities when $|U| \gg |I|$ (particularly when there are more ratings-per-item than ratings-per-user), it is feasible to pre-compute item–item similarities and cache the k' most similar items to each item. Prediction can then be performed quickly by looking up the similarity list for each item rated by the current user and aggregating their similarities into a predicted preference. Caching more items than are used in the similarity computation (so $k' > k$) is useful to increase the likelihood of having k similar items after items already rated by the user have been removed from the candidate set.

Pre-computation and truncation is essential to deploying collaborative filtering in practice, as it places an upper bound on the number of items which must be considered to produce a recommendation and eliminates the query-time cost of similarity computation. It comes with the small expense of reducing the number of items for which predictions can be generated (the *coverage* of the recommender), but the

unrecommendable items will usually have low predicted preferences anyway.

2.3.4 Example

Again using the example ratings in Figure 2.1, let us now compute C's prediction for *Equilibrium* using item–item CF with cosine similarity. We compute the length (L_2 norm) of each movie's vector, its norm with the vector r_e for *Equilibrium*, and finally the cosine similarity:

Movie	$\|\mathbf{r}\|_2$	$\mathbf{r} \cdot \mathbf{r}_e$	$\frac{\mathbf{r} \cdot \mathbf{r}_e}{\|\mathbf{r}\|_2 \|\mathbf{r_e}\|_2}$
Batman Begins (b)	7.348	24	0.607
Alice in Wonderland (a)	8.544	8	0.174
Dumb and Dumber (d)	7.348	15	0.382
Equilibrium (e)	5.385		

User C has rated all three other movies, but we will only use the most similar two when computing the prediction:

$$\begin{aligned} p_{C,e} &= \frac{s(b,e) \cdot r_{C,b} + s(d,e) \cdot r_{C,d}}{|s(b,e)| + |s(d,e)|} \\ &= \frac{0.607 * 5 + 0.382 * 2}{0.607 + 0.382} \\ &= 3.84 \end{aligned}$$

2.4 Dimensionality Reduction

In both of the traditional collaborative filtering algorithms so far described, there are hints of viewing the user–item ratings domain as a vector space. With this view, however, the vectors are of extremely high dimension: an item is a $|U|$-dimensional vector with missing values of users' preferences for it (similarly, a user is a $|I|$-dimensional vector). Further, there is redundancy in these dimensions, as both users and items will usually be divisible into groups with similar preference profiles (e.g., many science fiction movies will be liked to similar degrees by the same set of users). It is therefore natural to ask whether the dimensionality of the rating space can be reduced — can we find a smaller number of dimensions, ideally a constant number k, so that items and users can be represented by k-dimensional vectors?

In particular, we want to identify a set of k *topics* so that user preferences for items can be expressed as a combination of the user's interest in a topic (*topic interest*) and the extent to which each item is relevant to the topic (*topic relevance*). Some recommendation methods do this explicitly using content features, tags attached to the items, or user clustering or stereotyping. Latent semantic analysis, a technique pioneered in information retrieval, provides a way to do this decomposition using only the rating data. The topics are considered to be latent in the rating data.

In information retrieval, a document corpus can be represented as a term-document matrix where each cell is the number of times the given term occurs in a particular document. This results in high-dimensional representations of terms and documents, further complicated by the problems of synonymy (different terms having the same or similar meaning), polysemy (the same term having different meanings), and noise (documents or queries using terms incorrectly). Latent semantic analysis (LSA, also called latent semantic indexing or LSI) deals with these problems by using dimensionality reduction, in the form of truncated singular value decomposition (SVD), to extract the semantic relationships between documents latent in their use of vocabulary [16, 36]. SVD-based dimensionality reduction has since been adapted to collaborative filtering by Billsus and Pazzani [18], Sarwar et al. [128, 131], and many others.

2.4.1 Defining Singular Value Decomposition

For a matrix $\mathbf{M}$, its SVD is the factorization of $\mathbf{M}$ into three constituent matrices such that $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{T}^{\mathrm{T}}$, $\mathbf{\Sigma}$ is a diagonal matrix whose values σ_i are the *singular values* of the decomposition, and both $\mathbf{U}$ and $\mathbf{T}$ are orthogonal. What this accomplishes is introducing an intermediate vector space represented by $\mathbf{\Sigma}$. If M is the ratings matrix, $\mathbf{\Sigma}\mathbf{T}^{\mathrm{T}}$ transforms vectors from item-space into the intermediate vector space.

In the pure form of the SVD, $\mathbf{U}$ is $m \times \hat{k}$, $\mathbf{\Sigma}$ is $k \times \hat{k}$, and $\mathbf{V}$ is $n \times \hat{k}$, where $\mathbf{M}$ is $m \times n$ and has rank $\hat{k}$; this is not a significant gain. $\mathbf{\Sigma}$ can, however, be truncated by only retaining the k largest singular values to yield $\mathbf{\Sigma}_k$. The resulting decomposition is an approximation

of M. Further, using the Frobenius norm as the measure of error, it is the best possible rank-k approximation [36].

This truncation simultaneously achieves two goals. First, it decreases the dimensionality of the vector space, decreasing the storage and computational requirements for the model. Items and users can each be represented by k-dimensional vectors. Second, by dropping the smaller singular values, small perturbances as a result of noise in the data are eliminated, leaving only the strongest effects or trends in the model. In collaborative filtering, this noise can come as a result of other factors besides sheer preference playing a role in a user's rating; decreasing the impact of noise improves our ability to provide high-quality recommendations.

Computing the SVD of the ratings matrix results in the following factorization, with $m = |U|$, $n = |I|$, and $\mathbf{\Sigma}$ a $k \times k$ diagonal matrix (also shown in Figure 2.2):

$$\mathbf{R} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{T}^{\mathrm{T}} \tag{2.13}$$

Once the rank-k SVD of Equation (2.13) has been computed, it can be interpreted as an expression of the topic preference-relevance model. The rows of the $|U| \times k$ matrix $\mathbf{U}$ are the users' interest in each of the k inferred topics, and the rows of $\mathbf{I}$ are the item's relevance for each topic. The singular values in $\mathbf{\Sigma}$ are weights for the preferences, representing the influence of a particular topic on user–item preferences across the system. A user's preference for an item, therefore, is the weighted sum of the user's interest in each of the topics times that item's relevance to the topic.

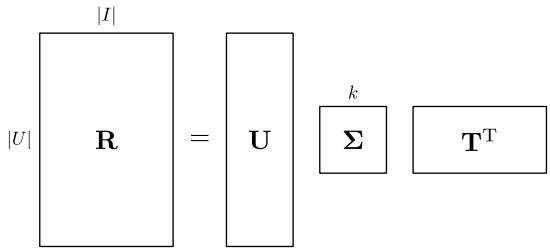

Fig. 2.2 Singular value decomposition of the ratings matrix.

2.4.2 Computing and Updating the SVD

In order to use SVD (or any matrix factorization method), it is necessary to first compute the matrix factorization. There are a variety of algorithms for computing singular value decompositions, including Lanczos' algorithm, the generalized Hebbian algorithm, and expectation maximization [51, 81, 127].

Singular value decomposition is only well-defined when the matrix is complete. Therefore, to factor the ratings matrix, the missing values must be filled with some reasonable default (a method called *imputation*). Sarwar et al. [131] found the item's average rating to be a useful default value (they tried user average as well, but item average performed better). Alternatively, the SVD can be computed over the normalized ratings matrix $\hat{\mathbf{R}}$ (see Section 2.1) and the missing values considered to be 0.

Several methods have been proposed that compute an estimate of the SVD only on the known ratings, dropping the requirement to impute or otherwise account for missing ratings. Kurucz et al. [81] propose a least-squares method that learns a regression for each user. Another method that has become quite popular in the last few years is gradient descent [44, 110]. This method trains each topic f in turn, using the following update rules (λ is the learning rate, typically 0.001):

$$\Delta u_{j,f} = \lambda(r_{u,i} - p_{u,i})i_{k,f} \tag{2.14}$$

$$\Delta i_{k,f} = \lambda(r_{u,i} - p_{u,i})u_{j,f} \tag{2.15}$$

The gradient descent method for estimating the SVD also allows for regularization to prevent overfitting the resulting model. The resulting model will not be a true SVD of the rating matrix, as the component matrices are no longer orthogonal, but tends to be more accurate at predicting unseen preferences than the unregularized SVD. The regularization is accomplished by adding an additional term to the update rules in Equations (2.14) and (2.15) [44]. γ is the regularization factor, typically 0.1–0.2.

$$\Delta u_{j,f} = \lambda((r_{u,i} - p_{u,i})i_{k,f} - \gamma u_{j,f}) \tag{2.16}$$

$$\Delta i_{k,f} = \lambda((r_{u,i} - p_{u,i})u_{j,f} - \gamma i_{k,f}) \tag{2.17}$$

Prior to computing the SVD, the ratings can additionally be normalized, e.g., by subtracting the user's mean rating or some other baseline predictor. This can improve both accuracy [131] and accelerate convergence of iterative methods [44].

Once the SVD is computed, it is necessary to update it to reflect new users, items, and ratings. A commonly used method for updating the SVD is *folding-in*; it works well in practice and allows users who were not considered when the ratings matrix was factored to receive recommendations and predictions [16, 129]. Folding-in operates by computing a new user-preference or topic-relevance vector for the new user or item but not recomputing the decomposition itself.

For a user u, folding-in computes their topic interest vector $\mathbf{u}$ such that $\pi_u \approx \mathbf{u}\mathbf{\Sigma}\mathbf{T}^{\mathrm{T}}$. Given their rating vector $\mathbf{r}_u$, $\mathbf{u}$ can therefore be calculated as $\mathbf{u} = (\mathbf{\Sigma}\mathbf{T}^T)^{-1}\mathbf{r}_u = \mathbf{T}\mathbf{\Sigma}^{-1}\mathbf{r}_u$. Setting unknown ratings to 0 causes the folding-in process to ignore them, and a new user preference vector is now available. The same process works for item vectors, except $\mathbf{U}$ is substituted for $\mathbf{T}$.

As user and item vectors are updated with the folding-in process, the accuracy of the SVD will diminish over time. It is therefore necessary to periodically re-compute the complete factorization. In a deployed system, this can be done off-line during low-load times [129].

Brand [21] proposed an alternative method for building and maintaining the SVD based on rank-1 updates. His method produces fast, real-time updates of the SVD, bootstrapping the SVD with a dense portion of the data set. This dense portion can be extracted by sorting users and items to make a dense corner in the ratings matrix.

2.4.3 Generating Predictions

The predicted preference of user u for item i can be computed as the weighted dot product of the user-topic interest vector $\mathbf{u}$ and the item-topic relevance vector $\mathbf{i}$:

$$p_{u,i} = \sum_f u_f \sigma_f i_f \tag{2.18}$$

Recommendation can be done by ranking items in order of predicted preference. The user's preference $\mathbf{p}_u$ for all items can be computed

efficiently by multiplying their topic-interest vector with the scaled item matrix $\mathbf{\Sigma T}^{\mathrm{T}}$:

$$\mathbf{p}_u = \mathbf{u\Sigma T}^{\mathrm{T}} \tag{2.19}$$

Recommendations and predictions for users who were not considered in the factorization can be computed by applying the folding-in process to compute a user preference vector given a rating vector.

2.4.4 Computing Similarities

Singular value decomposition can be used to generate user and item neighborhoods as well as predictions. User–user similarity can be computed using a vector similarity metric between the two users' topic interest vectors (rows in $\mathbf{U}$). Since $\mathbf{U}$ is orthogonal, a dot product between two rows suffices to compute their cosine similarity. Item similarities can be found similarly by operating on rows of $\mathbf{T}$ [36, 131].

These similarities can be used to compute neighborhoods which in turn produce recommendations [50, 128, 131]. Using neighborhoods to generate recommendations can be particularly useful in e-commerce domains with unary rating sets, as recommendations can be computed by summing the similarities of items to each of the items in the user's shopping cart.

2.4.5 Normalization in SVD

As with item–item CF, it is beneficial to normalize ratings by subtracting baseline predictors prior to computing the model [44, 110]. When doing this, however, it is necessary to also use the baseline when computing predictions. In item–item CF, since the similarity computations are only used as weights applied to averaging the user's other ratings, normalization in the prediction step is less important; with matrix factorization, the normalization is encoded into the decomposed matrix and therefore must be reversed at the prediction stage. If the baseline predictor $b_{u,i}$ has been subtracted from ratings prior to model computation, the prediction rule then becomes

$$p_{u,i} = b_{u,i} + \sum_f u_f \sigma_f i_f$$

The baseline predictor can be incorporated into the model learning process when using a gradient descent method, thus allowing for the recommender to learn user and item biases that may differ somewhat from plain means [79, 110].

2.4.6 Principle Component Analysis

Singular value decomposition is not the only matrix factorization method used for collaborative filtering. The Eigentaste algorithm used in Jester uses *principle component analysis* (PCA) on the dense matrix of user ratings for the "gauge set" of jokes rated by all users [50]. Eigentaste normalizes the subset of $\mathbf{R}$ consisting of the ratings of gauge set jokes by the mean and standard deviation of the ratings of each joke, resulting in a normalized ratings matrix $\hat{\mathbf{R}}$ ($\hat{r}_{u,i} = (r_{u,i} - \mu_i)/\sigma_i$, where μ_i is the mean rating for item i and σ_i the standard deviation of ratings of i). PCA then computes the correlation matrix $\mathbf{C} = \frac{1}{|U|-1}\hat{\mathbf{R}}^{\mathrm{T}}\hat{\mathbf{R}}$ and a factorization $\mathbf{C} = \mathbf{E}^{\mathrm{T}}\Lambda\mathbf{E}$, where Λ is a diagonal matrix with λ_i being the ith eigenvalue of $\mathbf{C}$. The top k eigenvalues are retained, and the resulting factorization projects users into k-dimensional space. Eigentaste then clusters users in the k-dimensional space (with $k = 2$) by recursive rectangular clustering and recommends jokes based on the preferences of other users in a user's cluster.

2.5 Probabilistic Methods

Besides the probabilistic item similarity functions discussed in *Computing Item Similarity*, several fully probabilistic formulations of collaborative filtering have been proposed and gained some currency. These methods generally aim to build probabilistic models of user behavior and use those models to predict future behavior. The core idea of probabilistic methods is to compute either $P(i|u)$, the probability that user u will purchase or view item i, or the probability distribution $\mathbf{P}(r_{u,i}|u)$ over user u's rating of item i (and the related problem $E[r_{u,i}]$, the expected value of u's rating of i).

Cross-sell [73] uses pairwise conditional probabilities with the naïve Bayes assumption to do recommendation in unary e-commerce domains. Based on user purchase histories, the algorithm estimates

$P(a|b)$ (the probability that a user purchases a given that they have purchased b) for each pair of items a, b. The user's currently-viewed item or shopping basket is combined with these pairwise probabilities to recommend items optimizing the expected value of site-defined objective functions (such as profit or sales lift).

Personality diagnosis [112] is a probabilistic user model that assumes that a user's ratings are a combination of their preference and Gaussian noise ($r_{u,i} \sim N(\pi_{u,i}, \sigma)$, where σ is a parameter to the recommendation model). For a user u, the probability of their true preference profile being equal to that of each other user u' ($P(\boldsymbol{\pi}_u = \mathbf{r}_{u'})$) is calculated and used to compute a probability distribution over ratings for items u has not yet rated. Prediction and recommendation are both done by computing the expected value of the user's rating using the resulting distribution.

2.5.1 Probabilistic Matrix Factorization

Probabilistic latent semantic analysis (PLSA, also called PLSI or probabilistic latent semantic indexing in the information retrieval literature) is a matrix factorization technique similar to singular value decomposition but arising from statistical probability theory rather than linear algebra [64, 65]. Jin et al. [67] applied it to mining web usage patterns; this led to its later application to recommender systems as a means to protect collaborative filtering results from manipulation by users Mobasher et al. [103].

The basis of PLSA is a probabilistic mixture model of user behavior, diagrammed with plate notation in Figure 2.3(a). PLSA decomposes the probability $P(i|u)$ by introducing a set Z of latent factors. It assumes that the user selects an item to view or purchase by selecting a factor $z \in Z$ with probability $P(z|u)$ and then selecting an item with probability $P(i|z)$; $P(i|u)$ is therefore $\sum_z P(i|z)P(z|u)$. This has the effect of representing users as a *mixture* of preference profiles or feature preferences (represented as a probability distribution over factors), and attributing item preference to the preference profiles rather than directly to the users. The probabilities can be learned using approximation methods for Bayesian inference such as expectation

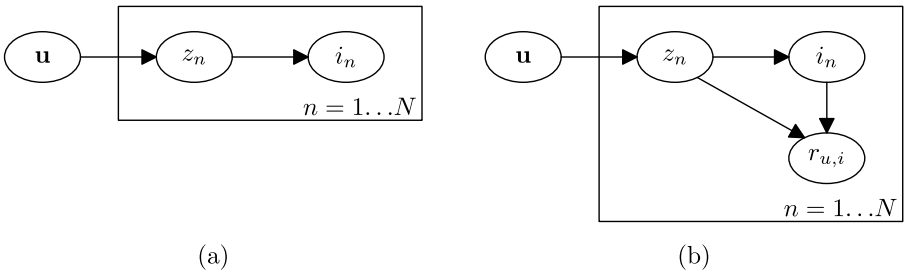

Fig. 2.3 PLSI generative model: (a) models user purchases and (b) models real-valued ratings varying by item.

maximization [37]. The probabilities can be stored in matrices, so that the preference matrix $\mathbf{P}$ (where $p_{u,i} = P(i|u)$) is decomposed into

$$\mathbf{P} = \hat{\mathbf{U}}\mathbf{\Sigma}\hat{\mathbf{T}}^{\mathrm{T}}$$

$\hat{\mathbf{U}}$ is the matrix of the mixtures of preference profiles for each user (so $\hat{u}_{u,z} = P(z|u)$) and $\hat{\mathbf{T}}$ is the matrix of preference profile probabilities of selecting various items. $\mathbf{\Sigma}$ is a diagonal matrix such that $\sigma_z = P(z)$ (the marginal probability or global weight of factor z). This factorization allows prediction to be done meaningfully in unary domains by considering the probability that u will purchase i, in contrast to item–item unary recommendation where the psuedo-predictions were only useful for ranking candidate items.

Hofmann [64] discusses in more detail the particular relationship between PLSA and SVD. The end result — a factorization of the ratings matrix into three component matrices — is quite similar, but the SVD is determined by minimizing error while PLSA is computed by maximizing the predictive power of the model. In the SVD, $\mathbf{U}$ and $\mathbf{T}$ are orthogonal matrices leading to a clear geometric or vector-space interpretation of the model, while the PLSA component matrices $\hat{\mathbf{U}}$ and $\hat{\mathbf{T}}$ are stochastic matrices with a natural probabilistic interpretation mapping more directly to generative models of user behavior.

PLSA can be extended to model ratings as well as purchases by introducing the rating $r_{u,i}$ as a new random variable and assuming that it is dependent on the latent topic z and either the specific user or item under consideration, as shown in Figure 2.3(b) [65]. If the user

is used, then it is assumed that u's preference for i is determined only by their preference for the latent factors z generating i; if the item is used, then all users with the same preference for a factor z will have the same preference for any item generated by that factor, with variation dependent only on the individual items. The resulting networks can again be learned by expectation maximization.

Blei et al. [20] extended the PLSI aspect model by using a Dirichlet prior over user preferences for topics. This method is called *latent Dirichlet allocation* and yields a more thorough generative model that accounts not only for users purchasing items but also for users coming into the system's knowledge. Users, represented by their preference for factors (the distribution $P(z|u)$), are considered to be instances of a random variable drawn from a Dirichlet distribution. This model, depicted in Figure 2.4, requires two parameters to be learned. For a model with k factors, these are α, a k-dimensional vector parameterizing the Dirichlet distribution from which users are drawn, and β, the $k \times |I|$ topic-item probability matrix (representing $P(i|z)$).

Another probabilistic method related to matrix factorization is *factor analysis* [27]. This method learns a probabilistic variant of a linear regression $\mathbf{R}^{\mathrm{T}} = \Lambda\mathbf{U} + \mathbf{N}$, where Λ is an $|I| \times k$ model matrix; $\mathbf{U}$, a $k \times |U|$ matrix of user preferences for each of k different factors, and $\mathbf{N}$, a Gaussian noise matrix. By using expectation maximization to learn Λ and the variance of the distribution from which the elements of $\mathbf{N}$ are drawn, a model can be computed for deducing user preferences for factors from ratings and using those preferences to produce recommendations.

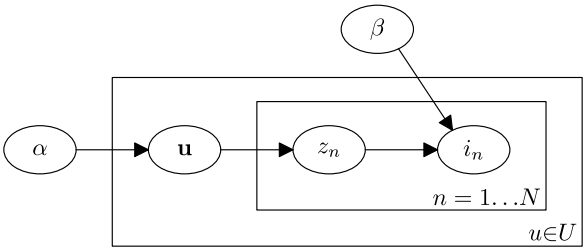

Fig. 2.4 LDA model of users and purchases.

2.5.2 Machine Learning Methods

A variety of other methods, usually probabilistic, from the machine learning and artificial intelligence literature have also been applied to recommendation. Some formulations of recommendation can be studied as classifier problems [22]. Recommenders have been built using Bayesian networks [30, 157], Markov decision processes [136], and neural networks [125].

2.6 Hybrid Recommenders

It is natural to consider combining several different recommender algorithms into a hybrid recommender system [24]. In some applications, hybrids of various types have been found to outperform individual algorithms [146]. Hybrids can be particularly beneficial when the algorithms involved cover different use cases or different aspects of the data set. For example, item–item collaborative filtering suffers when no one has rated an item yet, but content-based approaches do not. A hybrid recommender could use description text similarity to match the new item with existing items based on metadata, allowing it to be recommended anyway, and increase the influence of collaborative filtering as users rate the item; similarly, users can be defined by the content of the items they like as well as the items themselves. Fab used such an approach, matching items against both the content of items liked by the user and the content of items liked by similar users [13].

Burke [24] provides a thorough analysis of hybrid recommender systems, grouping them into seven classes:

- *Weighted* recommenders take the scores produced by several recommenders and combine them to generate a recommendation list (or prediction) for the user.
- *Switching* recommenders switch between different algorithms and use the algorithm expected to have the best result in a particular context.
- *Mixed* recommenders present the results of several recommenders together. This is similar to weighting, but the results are not necessarily combined into a single list.

- *Feature-combining* recommenders use multiple recommendation data sources as inputs to a single meta-recommender algorithm.
- *Cascading* recommenders chain the output of one algorithm into the input of another.
- *Feature-augmenting* recommenders use the output of one algorithm as one of the input features for another.
- *Meta-level* recommenders train a model using one algorithm and use that model as input to another algorithm.

Hybrid recommenders proved to be quite powerful in the Netflix Prize: the winning entry was a hybrid of some 100 separate algorithms. One important development to come from the prize competition is *feature-weighted linear stacking*, a new weighting hybrid method used by some of the top-ranking teams. Many weighting algorithms use a linear combination of the predictions of many recommenders:

$$p_{u,i} = \alpha_1 p_{u,i}^{(1)} + \cdots + \alpha_n p_{u,i}^{(n)} \tag{2.20}$$

Feature-weighted linear stacking replaces each coefficient α_j with a function g_j of item meta-features, such as number of ratings or genre, to alter the blending ratio of the various algorithms' predictions on an item-by-item basis:

$$p_{u,i} = g_1(i) p_{u,i}^{(1)} + \cdots + g_n(i) p_{u,i}^{(n)} \tag{2.21}$$

This allows, for example, the relative weights of item–item CF and actor list similarity to shift based on the number of ratings an item has received.

2.7 Selecting an Algorithm

The various algorithms presented each have their strengths and weaknesses. User–user and item–item CF algorithms are well-understood, easy to implement, and provide reasonable performance in most cases. User-based algorithms tend to be more tractable when there are more items than users, and item-based when the situation is reversed. Both methods also require minimal offline computation at the expense of

somewhat greater computational demands when recommendations are needed (precomputing item–item CF improves query-time performance at the expense of more offline computation). User–user CF seems to provide greater serendipity in its recommendations; in the case of MovieLens, this resulted in greater user satisfaction.

Matrix factorization methods require expensive offline model computation steps. The resulting models, however, are very fast for online use (prediction is simply a dot product). They are also capable of reducing the impact of ratings noise and can hamper the ability of users to manipulate the output of the system by decreasing their direct impact on each others' ratings. Probabilistic models are applicable when the recommendation process should follow models of user behavior.

In the end, the appropriate algorithm for a particular application should be selected based on a combination of the characteristics of the target domain and context of use, the computational performance requirements of the application, and the needs of the users of the system. *User Information Needs* discusses various needs users may have from the recommender system. Understanding more fully what needs are best met by what algorithms is a subject of ongoing research.

3

Evaluating Recommender Systems

When developing a recommender system, either a new algorithm or a new application, it is useful to be able to evaluate how well the system works. Since recommendation is usually a means to some other goal (user satisfaction, increased sales, etc.), testing ultimately needs to take this into account and measure the intended effect. However, it can be costly to try algorithms on real sets of users and measure the effects. Further, measuring some desired effects may be intractable or impossible, resulting in the need for plausible proxies.

In the recommender systems literature, offline algorithmic evaluations frequently play a major role. There has also been work on evaluation methods themselves [61]. It is common to use offline analysis to pre-test algorithms in order to understand their behavior prior to user testing as well as to select a small set of candidates for user testing from a larger pool of potential designs. Since user trials can be expensive to conduct, it is useful to have methods for determining what algorithms are expected to perform the best before involving users. Offline evaluation is also beneficial for performing direct, objective comparisons of different algorithms in a reproducible fashion.

This section describes commonly used data sets for offline evaluations of recommender algorithms, the basic structure of these

evaluations, and metrics that are used to evaluate performance. It concludes with a brief discussion of online evaluations of recommender performance.

3.1 Data Sets

Many recommender systems are developed in particular contexts, and their evaluations will be on data sets relevant to that context or on internal data sets. There are, however, several data sets that are publicly available and are widely used in evaluating recommenders. They form a basis on which the raw numeric performance of new algorithms can be compared against known performance of existing systems in a consistent environment, and can serve as a preliminary testing domain in building a system for which no directly relevant data is available.

One of the early publicly available data sets was the EachMovie data set. The DEC Systems Research Center operated a movie recommender system called EachMovie; when the system was shut down in 1997, an anonymized version of the data set was made available.[1] This data set consisted of 2.8 M user ratings of movies.

The MovieLens movie recommender, operated by GroupLens research, was bootstrapped from the EachMovie data set [35]. It has since made three data sets available: one with 100 K timestamped user ratings of movies, another with 1 M ratings, and a third containing 10 M ratings and 100 K timestamped records of users applying tags to movies.[2] The ratings in the MovieLens data set are user-provided star ratings, from 0.5 to 5 stars; older data sets have a granularity of 1-star, while newer ones have 1/2 star granularity.

The Jester data set is a set of user ratings of jokes collected from the Jester joke recommender system [50]. It consists of two data sets: one has ratings of 100 jokes from 73,421 users between April 1999 and May 2003, and the other has ratings of 150 jokes from 63,974 users between Nov 2006 and May 2009. These ratings are continuous ratings in the range $[-10, 10]$.[3] As a result of the "gauge set" required by the

[1] The EachMovie data set was later withdrawn in 2004.
[2] The MovieLens data sets are available at `http://www.grouplens.org/node/73`.
[3] The Jester data set is available at `http://eigentaste.berkeley.edu/dataset/`.

Eigentaste algorithm, a portion of the Jester data set is dense: there is a small set of jokes that almost every user has rated.

The BookCrossing data set is a collection of book ratings with some demographic information about users, collected from the book sharing and discussion site `bookcrossing.com` [156].[4] It contains over 1.1 M ratings from 279 K users for 271 K books. The ratings are a mix of explicit real-valued (1–10) and implicit unary ratings.

The Netflix data set, made available in 2006 as a part of the Netflix Prize [15], has been widely used as a large-scale data set for evaluating recommenders. It consists of over 100 M datestamped ratings of 17 K movies from 480 K users, with some perturbation applied to the ratings in an attempt to preserve privacy. The data set was withdrawn in late 2009 after publicity surrounding research to de-anonymize anonymized data sets [41, 105].

There are also other data sets in circulation from various sources. Yahoo! Research operates one such collection, providing a variety of data sets collected from Yahoo! services.

3.2 Offline Evaluation Structure

The basic structure for offline evaluation is based on the train-test setup common in machine learning. It starts with a data set, typically consisting of a collection of user ratings or histories and possibly containing additional information about users and/or items. The users in this data set are then split into two groups: the training set and the test set. A recommender model is built against the training set. The users in the test set are then considered in turn, and have their ratings or purchases split into two parts, the *query set* and the *target set*. The recommender is given the query set as a user history and asked to recommend items or to predict ratings for the items in the target set; it is then evaluated on how well its recommendations or predictions match with those held out in the query. This whole process is frequently repeated as in k-fold cross-validation by splitting the users into k equal

[4] The BookCrossing data set is available at `http://www.informatik.uni-freiburg.de/~cziegler/BX/`.

sets and using each set in turn as the test set with the union of all other sets as the training set. The results from each run can then be aggregated to assess the recommender's overall performance, mitigating the effects of test set variation [53].

This form of offline analysis has formed the basis of many evaluations of collaborative filtering, starting with [22]. Herlocker et al. [61] used this method as the basis for a comparative analysis of various evaluation methods and metrics.

Further refinements to the offline evaluation model take advantage of the temporal aspects of timestamped data sets to provide more realistic offline simulations of user interaction with the service. A simple temporal refinement is to use time rather than random sampling to determine which ratings to hold out from a test user's profile [53]; this captures any information latent in the order in which the user provided ratings. Further realism can be obtained by restricting the training phase as well, so that in predicting a rating or making a recommendation at time t, the recommendation algorithm is only allowed to consider those ratings which happened prior to t [25, 53, 85]. This comes at additional computational expense, as any applicable model must be continually updated or re-trained as the evaluation works its way through the data set, but allows greater insight into how the algorithm performs over time.

3.3 Prediction Accuracy

When testing recommender algorithms that are based on predicting user preference against a data set that expresses real-valued user ratings (such as the star ratings in MovieLens or the continuous ratings in Jester), measuring the accuracy of predicted values with respect to the held-out ratings is a natural starting point for evaluating recommender output.

A straightforward method of measuring the recommendation quality is to measure the *mean absolute error* (MAE) [22, 58, 59, 112, 137], sometimes also called absolute deviation. This method simply takes the mean of the absolute difference between each prediction and rating for all held-out ratings of users in the test set. If there are n held-out

ratings, the MAE is computed as follows:

$$\frac{1}{n}\sum_{u,i}|p_{u,i} - r_{u,i}| \tag{3.1}$$

MAE is in the same scale of the original ratings: on a 5-star scale represented as by integers in $[1,5]$, an MAE of 0.7 means that the algorithm, on average, was off by 0.7 stars. This is useful for understanding the results in a particular context, but makes it difficult to compare results across data sets as they have differing rating ranges (an error of 0.7 is more consequential when ratings are in $[1,5]$ than when they are in $[-10,10]$). The *normalized mean absolute error* (NMAE) [50] is sometimes used to address this deficiency. NMAE normalizes errors by dividing by the range of possible ratings (r_{high} and r_{low} are the maximum and minimum ratings in the system, respectively), resulting in a metric in the range $[0,1]$ for all rating scales:

$$\frac{1}{n(r_{\text{high}} - r_{\text{low}})}\sum_{u,i}|p_{u,i} - r_{u,i}| \tag{3.2}$$

NMAE results are harder to interpret in terms of the original ratings scale but are comparable across data sets with different ratings scales. They are therefore useful in measuring the impact of aspects of the data set on the performance of a recommender.

Root mean squared error (RMSE) [61] is a related measure that has the effect of placing greater emphasis on large errors: on a 5-star scale, the algorithm is penalized more for being 2 stars off in a single prediction than for being off by 1/4 of a star 8 times. It is computed like MAE, but squares the error before summing it:

$$\sqrt{\frac{1}{n}\sum_{u,i}(p_{u,i} - r_{u,i})^2} \tag{3.3}$$

Like MAE, RMSE is in the same scale as the original ratings. Famously, it was the measure chosen for the Netflix Prize: the \$1 M prize was awarded for a 10% improvement in RMSE over Netflix's internal algorithm. It can also be normalized for the rating scale like NMAE by multiplying by $1/(r_{\text{high}} - r_{\text{low}})$.

Equations (3.1)–(3.3) all describe measuring the accuracy across the entire data set at once. They can also be adapted to compute the average error for each user and aggregate the average errors to compute a score across the entire test set. Overall and per-user aggregation can result in different relative performance measures for algorithms. Aggregating by user provides a measurement of how users will experience the system, while overall aggregation gets at how the system will perform in general. Examining the error distribution by user or by item can be useful for identifying particular strong or weak areas for the recommender, either in user taste or item type.

All three of these prediction accuracy measures effectively measure the same thing. Which one to use depends on how the results are to be compared and presented (MAE vs. NMAE) or whether the additional penalty for large errors is desired ([N]MAE vs. [N]RMSE). They are only useful, however, for evaluating a recommender on the *predict* task. They also have the drawback of treating errors equivalently across items, while the error on low-importance items is likely to have less impact on user experience than the error on popular or important items.

To compute a single metric of predictive accuracy, we recommend RMSE due to its increased penalty for gross prediction errors. If metrics need to be compared between data sets, it can be normalized in the same manner as NMAE.

Predictive accuracy metrics focus on the predict task, but can also be indicative of a recommender's performance for recommendation as well. Koren [79] showed that, for a selection of algorithms, the ones with better RMSE performance also were more consistent at recommending movies users liked over randomly selected movies.

3.4 Accuracy Over Time

MAE and related metrics provide a static view of the predictive performance of a recommender system outside of time. Often, though, the performance of the recommender over time is interesting: does the recommender generate better results as more users use it? For individual users, how does accuracy change as they add more ratings?

This has led to the proposal of temporal versions of MAE and RMSE. Time-averaged RMSE [85] requires that predictions be performed in temporal sequence using only ratings before the current time t, as discussed by Gunawardana and Shani [53], and computes the RMSE over all predictions generated up to time t. Equation (3.4) shows this computation, with n_t being the number of ratings computed up through time t and $t_{u,i}$ being the time of rating $r_{u,i}$.

$$\text{TA-RMSE}_t = \sqrt{\frac{1}{n_t} \sum_{p_{u,i}:t_{u,i} \leq t} (p_{u,i} - r_{u,i})^2} \tag{3.4}$$

Lathia [84] also proposes windowed and sequential versions of this metric, restricting the time-averaging to ratings with in a window or simply computing the average error in a window, respectively. Burke [25] proposes Temporal MAE, an MAE version of Lathia's sequential RMSE, and another metric called Profile MAE which provides a user-centric view of error over time. Profile MAE computes how individual users experience the recommender, showing how the error in the predictions changes as they add ratings. It is also computed by considering each rating in temporal order, but averages error over the number of items in the user's profile at the time of prediction rather than by user or time window.

3.5 Ranking Accuracy

For contexts where ranked lists of items are produced and item rankings can be extracted from the user's rating data, the algorithm's capacity to produce rankings that match the user's can be measured.

Two ways of measuring this are with Pearson correlation or Spearman rank correlation coefficients. As noted by Herlocker et al. [61], however, the Spearman rank correlation metric suffers when the recommender makes distinctions between items that the user ranks at the same level. These metrics also penalize inaccuracy in the tail of the recommendation list, which the user will likely never see, to the same degree as inaccuracy in the top-predicted items.

Another metric for evaluating ranking accuracy is the *half-life utility* metric [22]. It measures the expected utility of a ranked

recommendation list, based on the assumption that users are more likely to look at items higher in the list; this assumption is reasonable for many real systems such as e-commerce sites. It requires two paremeters: a half-life α, such that the probability of a user viewing a recommendation with rank α is 0.5, and a default rating d. d should be selected to indicate neutrality, providing neither benefit for the user; it can be the user's mean rating, the system's overall mean rating, or some fixed neutral/ambivalent point in the rating scale. In unary domains, where this metric is most frequently applied, d can be 0 for "not purchased", with purchased items having a rating $r_{u,i} = 1$.

The half-life expected utility R_u of the recommendation list for a user u is defined in Equation (3.5). k_i is the 1-based rank at which item i appears.

$$R_u = \sum_i \frac{\max(r_{u,i} - d, 0)}{2^{(k_i - 1)/(\alpha - 1)}} \tag{3.5}$$

In order to measure the performance of a recommender across users, this metric is normalized by the maximum achievable utility $R_u^{\max}$ for each user, resulting in a value in $[0, 1]$ representing the fraction of potential utility achieved. $R_u^{\max}$ is the utility achieved by listing the best items for the user in order. The normalization is needed to compensate for different users having different potential utilities; one way this can happen is as a result of some users having more purchases to predict than others. The overall score R is given by Equation (3.6).

$$R = \frac{\sum_u R_u}{\sum_u R_u^{\max}} \tag{3.6}$$

Half-life utility can be difficult to apply and compare, as it depends on choosing α and d appropriately. The work of Agichtein et al. [2] is helpful for approximating α in the absence of domain-specific data; their analysis of search query logs shows that 50% of search result clicks are on the top 5 items, making $\alpha = 5$ a reasonable starting point.

Half-life utility suffers from the further drawback of penalizing recommendations of mildy-disliked and adamantly hated items equivalently if d is a rating value indicating ambivalence about an item [61]. However, in domains suggesting natural values for the parameters, its

focus on providing the highest-utility items at the beginning of the list make it a powerful tool for measuring the ability of a recommender to provide maximum value to users. Variants of it can be particularly useful if more nuanced utility functions are available to substitute for $\max(r_{u,i} - d, 0)$.

3.6 Decision Support Metrics

Recommender performance, particularly in unary domains such as purchase histories, can also be evaluated in the precision-recall framework of information retrieval derived from statistical decision theory [126]. This framework examines the capacity for a retrieval system to accurately identify resources relevant to a query, measuring separately its capacity to find all relevant items and avoid finding irrelevant items. Figure 3.1 shows the *confusion matrix*, depicting the relationship of these concepts. Recommenders are evaluated with retrieval metrics by operating in a unary purchase domain, holding some items out of the user's profile, and considering the held-out purchased items to be relevant to a query consisting of their remaining profile [22, 61, 131]. These metrics can also be applied in non-unary ratings domains by using a rating threshold to distinguish between liked and disliked items.

To be applied robustly, all of these measures require a fully-coded corpus where every item is known to be either relevant or irrelevant to the query under consideration. In recommender evaluation, however, that data is not generally available — if an item has not been purchased, it could be because the user does not like the item, or it could be because they were unaware of it. If the recommender returns an item outside the held-out target set, it still could be a relevant item. Care must be taken, therefore, when designing and evaluating precision-recall assessments

	Relevant	**Irrelevant**
Retrieved	TP	FP
Not retrieved	FN	TN

Fig. 3.1 Retrieval confusion matrix.

	Relevant	Irrelevant
Retrieved	**TP**	FP
Not retrieved	FN	TN

(a) Precision: $\frac{\text{TP}}{\text{TP+FP}}$

	Relevant	Irrelevant
Retrieved	**TP**	FP
Not retrieved	FN	TN

(b) Recall: $\frac{\text{TP}}{\text{TP+FP}}$

Fig. 3.2 Precision (a) and recall (b) in the confusion matrix.

of recommender systems; nevertheless, it can be a useful framework for understanding recommender performance.

The core metrics in precision-recall evaluations are the *precision* P, the fraction of items returned by the recommender that are purchased ($\frac{TP}{TP+FP}$), and the *recall* R, the fraction of purchased items returned by the recommender ($\frac{TP}{TP+FN}$) [126]. Figure 3.2 shows how these metrics relate to the confusion matrix. Since recommenders and retrieval systems typically return ranked lists, these are usually computed for some fixed recommendation list length N or sometimes for a threshold score. For a given system, precision and recall are inversely related and dependent on N, so comparing results between recommenders can be cumbersome [61]; however, the combination can provide more insight into the particular behavior of an algorithm than single-number measures. Different user tasks require different tradeoffs between precision and recall — a user looking for a movie recommendation likely only cares that they get a good movie (high precision), while a lawyer looking for legal precedent needs to find all relevant cases (high recall). Herlocker et al. [61] classify these two disparate needs as "Find Good Items" and "Find All Good Items".

Precision and recall can be simplified into a single metric, the F_1 metric [124, 153], which has been used to evaluate recommender systems [131]:

$$F_1 = \frac{2PR}{P + R}$$

F_1 blends precision and recall with equal weight. The resulting number makes comparison between algorithms and across data sets easy, but does not facilitate more nuanced comparisons where one of the two measures is more important than the other.

	Relevant	Irrelevant
Retrieved	**TP**	FP
Not retrieved	FN	TN

(a) Sensitivity: $\frac{\text{TP}}{\text{TP+FN}}$

	Relevant	Irrelevant
Retrieved	TP	FP
Not retrieved	FN	**TN**

(b) Specificity: $\frac{\text{TN}}{\text{TP+FN}}$

Fig. 3.3 Sensitivity (a) and specificity (b) in the confusion matrix.

Precision and recall also ignore the lower right quadrant of the confusion matrix — the true negatives, those items correctly identified as irrelevant or disliked. In order to incorporate that quadrant into a metric, a similar pair of metrics called *sensitivity* and *specificity* can be used [83]. Figure 3.3 shows these metrics. Sensitivity is equivalent to recall (the true positive rate) while specificity measures the fraction of irrelevant items correctly discarded (the true negative rate).

A related but more comprehensive comparison method is the *relative operating characteristic* (ROC, also called receiver operating characteristic) curve [142]. This curve is derived from sensitivity and specificity and plots sensitivity against the complement of the specificity. As the length of the recommendation list increases, more items are included, until the entire item set (including all relevant and irrelevant items) has been returned. ROC curves effectively describe the relationship of precision and recall across the range of their tradeoff, showing how the recall of the system increases as precision requirements decrease. Figure 3.4 shows some sample ROC curves. A recommender which randomly returns items will, on average, plot a straight diagonal line; curves above the line are better than random. The further to the upper-left corner of the plot the curve reaches, the better the recommender performs. A perfect recommender (which returns all target items before any irrelevant items) will plot a vertical line to (0.0,1.0) and a horizontal line to (1.0,1.0).

The ROC curve can also be used to derive a single numeric measure of performance by computing the area under the curve, called Swet's A-measure or AUC; a perfect recommender will achieve an AUC of 1. AUC is limited, as a given recommender system will operate a particular ratio of precision to recall or will move along the curve as the user explores longer recommendation lists. It is useful, however, for

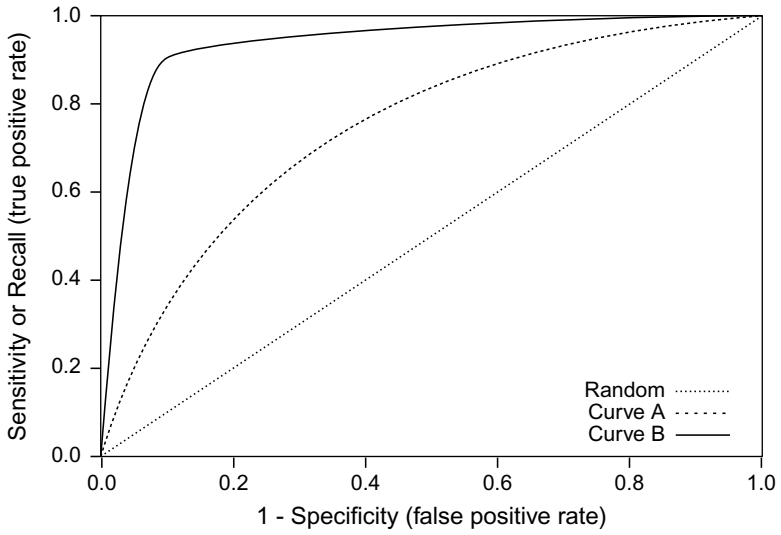

Fig. 3.4 Sample ROC curves.

quantifying the tradeoff in precision necessary to increase recall with a particular system, measuring in some sense the "efficiency" of the recommender.

ROC curves are good for performing detailed assessments of the performance of particular algorithms. Comparing them, however, is challenging, particularly for large numbers of algorithms. Therefore, precision/recall, F_1, and AUC are useful for assessing relative performance of various algorithms in decision support contexts at the expense of oversimplifying recommendation performance.

3.7 Online Evaluation

Offline evaluation, while useful for comparing recommender performance and gaining understanding into how various algorithms behave, is limited in a variety of ways [88, 89, 98]. It is limited to operating on data regarding past behavior — a leave-N-out recommender trial measures only the ability to recommend items the user found somehow. Further, there is a selection bias in historical rating data, as users are more likely to consume and therefore be able to rate items

they perceive to be good. Time-based methods compensate for this somewhat, but offline evaluations are incapable of measuring the recommender's ability to recommend good items that the user did not eventually find somehow. They also cannot measure how users respond to recommendations or how recommendations impact business concerns. Recommendation algorithms with similar numeric performance have been known to return observably different results [97, 146], and a decrease in error may or may not make the system better at meeting the user's needs. The only way to accurately assess these factors is to do an online evaluation — to try a system with real users.

Online evaluations come in a variety of forms. Field trials — where a recommender is deployed in a live system and the users' subsequent interactions with the system are recorded — enable the designer to see how users interact with the recommender and change their behavior in "real" environments. Since many recommenders are provided in web-based contexts, page views, interface clicks, rating behavior, and purchase behavior can be measured and used to assess the impact of the recommender. Field trials, frequently in the form of A/B tests, are commonly used in commercial recommender deployments where sites have already established user bases and can test a new algorithm or other site change on a small subset of their users [76, 77]. Amazon and Google are well-known for using A/B testing in refining and evaluating their services. Field trials can also be used to measure other aspects of system performance such as recommender accuracy in response to a change in the system [117]. When combined with user surveys, they can be used to assess user satisfaction and other subjective qualities. Recommender effectiveness with respect to business concerns can also be measured in field trials using a variety of measurements used to evaluate marketing campaigns [31].

Virtual lab studies are another common method for evaluating recommender systems. They can be used without access to a live user base, and are therefore useful for testing new ideas for which no deployed application exists or is available. They generally have a small number of users who are invited to participate, and can take the form of interaction with a full application [19] or a survey structured around the recommender [29, 39, 118, 146]. Virtual lab studies, while they provide

a great deal of flexibility and control in designing an experiment for testing a recommender, create an artificial interaction that may affect the applicability of the results to real applications.

Traditional in-lab studies are also used to evaluate recommender systems [141], but virtual lab studies and field trials are generally more common.

Many uses of online evaluation, such as measuring accepted recommendations, page views, conversions, or sales lift, are well-understood. Well-constructed surveys can help get at many other aspects of recommender quality and utility. Determining ways to measure more subtle aspects of recommender quality and user satisfaction is the subject of ongoing research. Shani and Gunawardana [135] describe many aspects of online and offline evaluations, and more work is needed to come up with good ways to assess recommender's ability to meet user's needs. *User Information Needs* discusses in more detail what some of these needs are and surveys research on addressing them.

4

Building the Data Set

In order to provide recommendations to users, simply having a recommender algorithm is not sufficient. The algorithm needs a data set on which to operate — a ratings matrix (or its functional equivalent) must somehow be obtained from the users of a system. Brusilovsky [23] provides a summary of how this is necessary in the broader space of adaptive user experiences: the system must collect data about users in order to model their preferences and needs and ultimately provide them with a customized experience (in the present case, customized recommendations or preference predictions).

The need for preference data can be decomposed into two types of information needs. The system needs to know something about users' preferences: what items (or types of items) does each user prefer? It also needs to know something about items: what kinds of users like or dislike each item? This breakdown is perhaps clearest when thinking of matrix decomposition models for recommendation: the user–item preferences are factored into a set of characteristics, user preferences for those characteristics, and those characteristics' applicability to various items. These needs are also present in other models to varying extents: the item–item CF model, for instance, needs to know what items are

liked by the same users (a combination of "what users like this item?" and "what other items are liked by these users?") as well as the current user's preferences.

These information needs come up in the recommender systems literature primarily under the guise of the *cold-start problem*. The cold-start problem, in general, is the problem of providing recommendations when there is not yet data available on which to base those predictions. It takes two flavors:

- *Item cold-start*, where a new item has been added to the database (e.g., when a new movie or book is released) but has not yet received enough ratings to be recommendable [133].
- *User cold-start*, where a new user has joined the system but their preferences are not yet known [34].

The bootstrap problem, where a system has no information about any user preferences, is an extreme intersection of both aspects of the cold-start problem.

Not only does a recommender system need data, it needs that data to be of high quality in order to provide useful recommendations. If it does not accurately know user preferences, its ability to generate useful recommendations can suffer — prediction error is lower-bounded by the error in user ratings [62]. Rating error ($|r_{u,i} - \pi_{u,i}|$, the difference between users' ratings and their true preferences) can be divided into two categories: natural noise, where user ratings differ from true preference due to error in the rating collection process, and malicious noise, discrepant ratings introduced for the purpose of manipulating the system's recommendations or predictions [109]. In this section, we are only concerned with natural noise; malicious noise will be taken up in *Robustness* when we discuss security implications of recommender systems.

4.1 Sources of Preference Data

Preference data (ratings) comes from two primary sources. Explicit ratings are preferences the user has explicitly stated for particular items. Implicit "ratings" are inferred by the system from observable user activity, such as purchases or clicks.

Many recommender systems, in both academic and commercial settings, obtain ratings by having users explicitly state their preferences for items. These stated preferences are then used to estimate the user's preference for items they have not rated. From a system-building perspective, this is a straightforward method and avoids potentially difficult inference problems for user preferences. It suffers, however, from the drawback that there can, for many reasons, be a discrepancy between what the users say and what they do.

Preferences can also be inferred from user behavior [106]. In the Usenet domain, this has been examined by using newsgroup subscriptions [69] and user actions such as time spent reading, saving or replying, and copying text into new articles [104]. Konstan et al. found a substantial correlation between the time a user spent reading a Usenet article and his/her rating of that article [78], further supporting the idea that observable user behavior can be a useful source of preference information. Consumption behavior has also been extended beyond reading domains: while not a collaborative recommender system, the Intelligent Music Management System plugs in to a user's music player and automatically infers the user's preference for various songs in their library as they skip them or allow them to play to completion [52]. The PHOAKS system mined Usenet articles for another form of implicit preference data: mentions of URLs. It used the frequency with which URLs are mentioned in Usenet articles to provide recommendations for web resources [63]. In e-commerce domains, implicit feedback in the form of page views and item purchases is the primary source of preference data for recommendation [87].

System designers and analysts must be careful when using implicit data to understand the mapping between system-visible user identities (frequently accounts) and actual people. Users of e-commerce systems, for example, will often purchase items as gifts for others. Those purchases and related activity do not necessarily communicate anything about the user's tastes. Users will also share accounts on many systems; a family may have a single Amazon.com account, for instance, thus providing the system with an ill-defined aggregate of several people's preferences. Netflix deals with this problem by allowing each account to have multiple profiles with separate ratings, queues, and recommendations.

O'Mahony et al. [109] argue that observed or inferred preference information is expected to be noisier than explicitly provided data. It is possible, however, that inference can build a more accurate long-term profile of a user's taste then they themselves can articulate.

Explicit and implicit rating data are not mutually exclusive. Systems can incorporate both into the recommendation process. Most recommender systems are capable of collecting some implicit information at least through page views. As rating mechanisms become increasingly prevalent in a variety of web sites, particularly e-commerce sites, further hybrid data becomes available to the operator of those services. Amazon.com makes use of both explicit and implicit data in providing its various recommendations.

Rating data can also be obtained from external sources, although this data may not be identifiably connected to users of the recipient system. In systems with few users or brand-new systems facing the bootstrapping problem, external data sources can be particularly valuable in getting initial preference data to actually be able to make recommendations. MovieLens did this by bootstrapping with the EachMovie data set [35].

Preference data can also be taxonomized based on its relationship to the user's experience of the item. Table 4.1 shows how the experience relation relates to implicit and explicit data with some examples.

- *Consumption ratings* are provided when the user has just consumed (or is consuming) the item in question. These ratings are particularly common in online streaming media environments, such as the Pandora music recommender or NetFlix instant streaming.

Table 4.1. Types of preference data.

	Consumption	Memory	Expectation
Explicit	Pandora song ratings, GroupLens Usenet article ratings	Netflix or MovieLens movie ratings	House advertisement ratings
Implicit	Streaming video watch time	Views of discussion pages related to previous purchase	Dwell time on new car advertisement

- *Memory ratings* are provided based on the user's memory of experiencing the item. When a user watches a movie at the theater and then rates it on MovieLens or MoviePilot, they are providing a memory rating.
- *Expectation ratings* are provided when the user has not yet experienced the item (and may never experience it) but is willing to provide a rating anyway. One example is rating housing options — the user will likely only have the means and opportunity to truly experience one housing situation, but may provide ratings on other house advertisements indicating their level of appeal to the user.

In some ways, consumption ratings are the most reliable, as the item is fresh in the user's mind. Memory ratings are also based on experience, so the user has a fuller set of data on which to base their rating than in the expectation case, but their impression of the item may not be accurately remembered or may be influenced by various external factors [68]. Expectation ratings are the most limited, as the user has not experienced the item and thus is rating it based only on the information presented about it combined with whatever prior knowledge they may have. They can still be useful, however, particularly in high-cost domains.

4.2 Rating Scales

Mapping user preference to a rating scale is not an easy task. While utility theory shows that there is a real-valued mapping for the preferences of any rational agent [149], it is not necessarily easy for users to express their preferences in this way. Further, preference is not a function only of the item: contextual factors such as time, mood, and social environment can influence a user's preference, so provided ratings may contain information about the context in which the user experienced the item and, in the case of memory ratings, intervening events or changes of mind as well as their preference for the item itself.

Recommender systems over the years have used (and continue to use) a wide variety of different scales and methods for collect-

Fig. 4.1 The MovieLens 5-star rating interface. The page also contains a legend, defining 1 to 5 stars as "Awful", "Fairly Bad", "It's OK", "Will Enjoy", and "Must See".

ing ratings [34]. Many of these systems are a numeric scale of some kind: GroupLens used a 5-star scale [119], Jester uses a semi-continuous -10 to $+10$ graphical scale [50], and Ringo used a 7-star scale [137]; Figure 4.1 shows a 5-star scale as deployed in MovieLens. These scales are appealing to the system designer, as they provide an integral or real-valued rating that is easy to use in computations. They also benefit from the body of research on Likert scales in the survey design and analysis literature [9, 43, 45].

Explicitly numeric scales are not the only scales in use. Some systems use a "like"/"dislike" method — the Pandora music recommender provides thumbs-up and thumbs-down buttons which can be used to request more songs like the current song or ban the current song from the station. The Yahoo Launch music player used a real-valued "like" rating (0–100) in combination with a "ban" button [34]. MovieCritic.com had users rank films and derived ratings from these rankings.

Cosley et al. [34] found that users prefer finer-grained scales when rating movies, and that scale had little impact on user ratings (users rated items consistently even when using different scales). That work also found that the rating scale had little impact on prediction accuracy. Therefore, using fine-grained scales (such as allowing 1/2 star ratings) makes sense as users prefer it and it does not harm recommender accuracy. There is little research into the particular impact of rating interfaces in other contexts. Intuitively, simple like/dislike scales seem useful in systems such as real-time music recommenders where

the user actually experiences the item rather than receiving a list of suggestions. It is important to include both like and dislike, however; in a study of user rating behavior for movie tags, Sen et al. [134] found that users with both "like" and "dislike" buttons provided more ratings than users with only one rating option, and users who had a "dislike" button but no "like" button provided far more ratings than users whose only option was "like".

When implicit ratings are collected, the observed user events must be translated into computationally useful estimators of preference [106]. Purchases, page views, and data such as the mention data used by PHOAKS are frequently treated as unary ratings data: purchased items have a "rating" of 1 and other items have unknown ratings [71]. With unary ratings, all that is known is that the user purchased a particular set of products: the fact that the seller does not have any information on a user's interest in other products does not imply that the user does not like those products. They may be unaware of the existence of unpurchased products (a situation the recommender likely aims to change), or they may have purchased them from a different vendor. Therefore, while a purchase can be considered a vote for a product, a lack-of-purchase alone may not be considered a vote against.

Other forms of data, such as the time spent on a page, also provide useful data points for inferring user preference. In the Usenet domain, time spent reading an article correlates strongly with the user finding that article interesting [78, 104]. It can be beneficial to consider a variety of observable user activites, particularly to discern the user's interest in an item they have not purchased in a commerce domain.

4.3 Soliciting Ratings

Beyond the input method used for providing ratings, there are also questions to consider in selecting which items to ask or encourage users to rate. To optimize individual user experience, a recommender system should provide quality recommendations with as enjoyable a rating experience as possible; in some cases, that may mean minimizing the effort required from the user, but requiring work users perceive as meaningful can result in higher user loyalty [96]. It should also provide an

enjoyable experience for the user. Therefore, when a new user joins the system, conducting the initial "interview" strategically, aiming to maximize the information gained about the user's tastes for the items presented for rating, can provide a smooth initial user experience. This is not the only consideration, however — recommender systems have multiple users, and collaborative filtering depends on knowing as many users' preferences for items as possible. Therefore, interview strategies can also attempt to improve the system's ability to serve the user base as a whole by asking users to rate items for which few ratings have been collected. Further, new users are not the only entities about which information needs to be collected. New items also present a problem, and the system has the opportunity to strategically present them to existing users to gain an understanding of their appeal profile.

Information and decision theory provide some insight into how this problem can be approached. Value-of-information (VOI) calculations can be used, based on estimated improvement in prediction accuracy, to ask the user to rate items that provide the system with the highest expected improvement in quality [112]. Entropy-based measures can also be used, determining which items will likely provide the system with the most information about the user's taste [117]. The user's preference for a movie with high variance in ratings tells the system more about that user's taste than their preference for a movie with more uniform ratings. Entropy alone, however, is not sufficient, as there may be many items with high information value that the user is unable to rate. Therefore, a probability that the user will be able to rate the item must also be factored in to the decision process; Rashid et al. [117] used the popularity of items, asking the user to rate items that were both popular and had high information value. The structure of the resulting formula is similar to value-of-information, maximizing the expected information gain.

The cost of consuming items is also a factor in the ability of a system to solicit user ratings and fill in user taste profiles. Movies, for example, have a higher cost of consumption than songs — even if the media is free, it takes around 2 hours to watch a movie while many songs last only 3–4 minutes. Therefore, a movie recommender is likely to depend on finding movies users have seen in order to collect ratings

(thus relying predominantly on memory ratings). A music recommender can have more flexibility — systems like Pandora or Launch.com can actually play the song for the user to get their opinion of it rather than just giving them a title and artist. Similarly, Jester has users rate jokes based on reading the joke; the small cost of reading a joke allows Jester to present the same starting set of jokes to each user to build an initial version of their preference profile [50]. These sytems are therefore able to make greater use of consumption ratings. Some domains, such as house or car purchases, have potentially very high cost, where items are expensive and their true utility is not fully known until they have been used for an extended period of time. Systems in such domains are therefore either unable to use ratings or depend primarily on expectation ratings.

A further consideration in soliciting ratings is whether or not to show the user's predicted preference in the rating interface. Cosley et al. [34] found that predicted preferences influence users' ratings, and that some expert users are aware of this influence. System designers should be aware of this and careful to consider what impact it will have on their users.

There has also been work looking more directly at user motivation and willingness to provide ratings. Harper et al. [56] provide an economic model for user behavior in providing ratings. This incorporates some issues already discussed, such as the time and effort needed to provide a rating, and provides a framework for reasoning about how users are motivated to interact with recommender systems. Avery and Zeckhauser [11] argue that external incentives are needed to provide an optimal set of recommendations and that market-based systems or social norms can provide a framework for promoting user contribution to the rating data.

4.4 Dealing with Noise

The process of expressing preference as ratings is not without error — the ratings provided by users will contain noise. This noise can result from normal human error, confounding factors in the rating process (such as the order in which items are presented for rating), and other

factors in user experience (e.g., a noisy theater detracting from the enjoyment of an otherwise-good movie). Detecting and compensating for noise in the user's input ratings therefore has potential to result in better recommender systems.

Natural noise in ratings can be detected by asking users to re-rate items [5, 34, 62]. These studies all found that users' ratings are fairly stable over time. Amatriain et al. [5] further found that moderate preferences — 2 or 3 on a 1–5 scale — are less stable than extreme like or dislike, and performed a more thorough analysis of the re-rating data to determine that ratings are a reliable method of measuring user preferences by the standards of classical test theory. Once noisy or unstable preferences have been detected, they can be discarded to improve the recommender's prediction accuracy [6].

O'Mahony et al. [109] proposed detecting and ignoring noisy ratings by comparing each rating to the user's predicted preference for that item and discarding ratings whose differences exceed some threshold from the prediction and recommendation process. This approach, however, makes the assumption that all high-deviance ratings are a result of user error rather than aspects of user preference for which the system cannot yet account, and discarding such ratings may inhibit (at least temporarily) the system's ability to learn that aspect of the user's taste.

Some recommendation algorithms have noise built into their user ratings model. Personality diagnosis explicitly assumes that user-provided ratings have Gaussian noise, being drawn from a normal distribution whose mean is the user's true preference [112]. Analyzing ratings with this model in a naïve Bayesian framework allows data from other users and the smoothing effect of probabilistic inference to compensate somewhat for noise in individual ratings.

5

User Information Needs

Recommender systems do not exist in a vacuum. Treating recommendation abstractly as mathematical problem, aiming primarily to improve offline evaluations with prediction accuracy metrics such as RMSE, ignores the broader context of use and is not necessarily measuring the impact these systems have on their users. Re-grounding recommender systems in user needs can have a profound impact on how we approach the field. Anchoring both the design of the recommender system itself and the evaluation strategy used to measure its effectiveness to a detailed understanding of user goals, tasks, and context can enable us to build systems that better serve their users [74, 99].

Users use a recommender system for some purpose. With systems like GroupLens and PHOAKS, that purpose can be to more easily filter through high volumes of articles and find interesting new resources. Movie recommenders can help users find new movies and choose films to watch. A joke recommender can provide entertainment. In each case, we can consider the recommendations to have some utility to the user. User needs can also extend beyond the recommendation list — some users interact with recommender systems for the purpose of self-expression, with the rating process rather than the resulting recommendations being their desired end [56, 61]. Evaluation metrics are useful, therefore, to the extent that

they map to (or at least correlate with) the user's utility derived from the system's output (recommendations and predictions) and the overall experience it provides. Task- and need-driven user-based studies are needed to determine what factors actually do affect the system's ability to meet user needs and improve the user experience.

5.1 User Tasks

The classical recommender tasks of *predict* and *recommend* can be re-cast in terms of user needs as "estimate how much I will like an item" and "find items I will like", respectively [61]. User needs can also take more nuanced manifestations. For example, users can use a recommender system to find new items the may like (*introduction*) or to recall previous items they enjoyed (*reuse*); systems like the Pandora music recommender are built to meet these two needs in balance (users want to discover new music while also listening to music they know they like). Users can also be interested in merely exploring the item space (*explore*, the "Just Browsing" task of Herlocker et al. [61]) rather than supporting a particular decision (*make decision*). Some users have the goal of determining the recommender's credibility (*evaluate recommender*) — they may wish to see how the recommender does at estimating their preference for items they know well in order to determine how much they trust its unfamiliar recommendations (or how much time they want to invest in training the system). By supporting this last task, a balance of familiar and unfamiliar items can be important in a developing long-term relationships between users and recommenders [99].

The type of interaction the user has with the recommender also impacts how it should perform. Does the user treat the recommender as an information source to be searched, or as a decision support tool? Even when using a recommender for decision support, users still have differing goals. In some cases they may be primarily interested in exploring the space of options. They may turn to the recommender to provide them with a candidate set from which they will choose. In other cases, they may want the recommender to actually make the selection. A movie recommender or an e-commerce site is likely to be used to explore or determine a candidate set; the user generally makes the

final decision on which movie to see. Real-time music recommenders such as Pandora, however, select a song for the user to hear next, providing the user with a means for critiquing the decision but not often do not allow the user directly select from a set of items.

Even recommenders with equivalent numerical performance can have qualitative differences in their result lists [146]. McNee et al. [99] call these *personalities*, and choosing an algorithm whose personality matches the user's needs can provide greater user satisfaction.

5.2 Needs for Individual Items

Thinking about prediction from the perspective of user perception has led to the adoption of some common evaluation metrics: RMSE's increased penalty for high error and other variants on MAE designed to distinguish high error both stem from the assumption that users are likely to forgive a small error in preference estimation (such as mispredicting by half a star), while gross errors (predicting a 3-star movie as a 5-star) will more significantly hamper user experience. It is also frequently more important to be accurate at the high end of the scale: if the system correctly predicts that the user will dislike an item, does it matter how much they will dislike it?

Another need users may have in their use of the recommender system is *serendipity* [98, 116]. Serendipity is the ability to recommend items that the user likes but does not expect; a movie, perhaps, that they never would have thought of liking but, once recommended, turned out to be enjoyable. Since users frequently use recommenders to learn about and try things that they would not otherwise know about, serendipity is generally a desirable characteristic of a recommender.

Related to serendipity is the factor of *risk* — what are the odds the user will like the item? If the system is recommending items that the user does not expect, it could be wrong sometimes and recommend a bad item. In some contexts, users may be better served by a recommender which is more aggressive in exploring the boundaries of the user's preference space in hopes of achieving serendipity, while other contexts may merit a more conservative system that prefers to avoid the chance of a bad recommendation.

Risk can be modelled by the system's confidence in its prediction, but that still leaves open the question of how much risk to take. Is the user more interested in finding potentially great items, or will the user accept merely good items in order to avoid bad ones, even if that involves missing potentially valuable recommendations? This is a function of both the user's desires and the characteristics of the domain. In low-cost domains, particularly interactive ones such as music recommenders, the cost of following a bad recommendation is low, so a recommender with riskier behavior may be more appropriate than in a higher-cost domain.

The issue of new versus old items arises here as well. If the user's task is "find a movie to go see this weekend", they are likely not interested in a movie they have seen before. But if it is "play some music for my commute", a blend of old and new may well be appropriate. Careful analysis of who the users are and what they expect from a system will provide insight into how to design the recommendation strategy.

5.3 Needs for Sets of Items

Many recommenders do not just recommend a single item to the user. They present a list, usually ranked, of items for the user to consider. Even recommenders which, at first glance, deal with individual items at a time may have a sense of the set or the whole: a music recommender plays invidiual songs in turn for a user, but the user perceives a sequence of songs as well as the individual selections [40].

At a basic level, the rank-accuracy metrics fit in line with viewing recommendation from a user perspective, as the exact preference prediction frequently does not matter when the user is only presented with a ranked list of choices. Half-life utility [22] adds the additional factor of modeling whether the user will even see the item in question, making it useful for assessing utility to the user in search-like contexts.

When users receive a list of recommended items, they do not experience it as a list of individual items but rather experience the list as an entity itself. It can well be that the best 5 items do not necessarily make the best 5-item recommendation list.

Research on research paper recommendations has shown that different algorithms produce lists that are better-suited for different goals: some algorithms produce lists better-suited for finding novel or authoritative work on a topic, while others produce lists that are better introductory or survey reading lists [146]. Users could discern differences between lists produced by different algorithms, including varying applicability to different tasks.

Perhaps the most prominent potential need users have when it comes to sets of items is that of diversity. Ali and van Stam [4] noted that, with standard collaborative filtering technology, a TV viewer who enjoys both science fiction shows and boating programs, but who has seen and rated more science fiction, may never receive boating recommendations. This domination of a profile by a single genre can reduce the recommender's capacity to satisfy the user needs across their multifacted interests.

Diversity in recommendations can desirable from two standpoints. First, users may have a variety of tastes, some expressed more strongly than others, and benefit from the recommender being able to recommend items across the breadth of their preferences. In many cases, the recommender should not only span their interests but also be able to push at the edges of the user's known interests with the hopes of introducing them to material that they enjoy but would not have found otherwise. Immediately, it needs to avoid pigeonholing users — the science-fiction and boating TV viewer of Ali and van Stam's portfolio effect may want to watch a good boating show in a Saturday afternoon, and the recommender may have an insufficient sense of diversity to provide one.

Second, excessively narrow recommendations have the potential to fragment communities into small pockets with little in common, with the rate-recommend cycle reinforcing an increasingly focused taste at the expense of breadth. Societally, this balkanization can decrease communication and interactions between groups of people [147].

Diversity is not always a desirable aim, however. Some user needs are better met by a more focused set of recommendations, and the need for diversity can change over time. Consider buying a house: in

the early stages, recommending a wide variety of styles, prices, and neighborhoods could be a good strategy. As the user looks at houses and sees what they do and do not like, narrowing the field of recommendations to a more focused set of options can help them to make their final decision. In the latest stages, it can be desirable to recommend several effectively substitutable options in case one is unavailable. Again, understanding the user's needs and relationship to the system is key to determining the role diversity should play in recommendation.

To add diversity to a recommendation list, Ziegler et al. [156] proposed "topic diversification", a method which discards some items of a recommendation list in favor of others to increase the diversity of items within a recommendation list; users found this to more comprehensively capture their reading interests in a study using the BookCrossing data set.

Another recent and promising approach for optimizing the variety of choices presented to the user in a recommendation set is regret-based recommendation [148], a recommendation algorithm based on utility theory. This method uses a model of the user's utility to recommend a set of items with minimal expected loss. The basic idea behind its operation is that a recommendation set that contains only comedies will have greater loss if the user is in the mood for an action movie than a set containing a mix of comedies and action movies. This framework is not just applicable to diversity, but can be adapted to address a host of whole-list concerns.

Key to both of these algorithms is that they consider the set of recommended items as a whole rather than simply picking the best individual recommendations. Seeking to maximize the expected utility of a set of recommended items rather than focusing on maximizing the user's interest on a per-item basis is a step forward in improving user experience across diverse tastes.

In order to understand and reason about diversity in recommendations, it is useful to understand more generally what diversity is and how to measure it. Harrison and Klein [57] identify three types of diversity studied in operations research: *separation*, where the members of a set are at different positions on a linear continuum; *variety*, where members of a set are members of different classes; and *disparity*, where

some members of a set have a higher status or position or are otherwise superior to other members. Separation and variety seem to be particularly relevant to assessing recommendation and preference: movies can be more or less violent (separation), and they can be of differing genres or from different directors (variety). Harrison and Klein's review also provides metrics for measuring the various types of diversity: standard deviation and Euclidian distance can be used to measure separation, and Blau's index and Teachman's entropy index are both useful for measuring variety among classes. Ziegler et al. [156] used the average pairwise similarity of items in a recommendation set, with proximity in a hierarchical taxonomy of the item space as the similarity metric, to measure the similarity (lack of diversity) within recommendation lists.

Measuring diversity is useful in understanding the behavior of a particular system, but the extent to which the diversity of a recommendation list impacts that list's ability to meet a user's need is harder to measure. It is also not always obvious what forms of diversity relate to users' needs. Ziegler et al. [156] found that increasing the diversity across Amazon.com book categories improved user's assessment of the value of book recommendation lists they received up to a point, after which further diversification decreased perceived value. The nature and impact of diversity will vary across information domains and user tasks.

Another major concern in set recommendation is that of interaction effects. Some items, while good individually, are either particularly bad or particularly good together [55]. Accounting for these is a difficult problem, and package recommendation under constraints frequently becomes NP-hard [152]. However, recommendations often do not need to be perfect — approximation algorithms are often sufficient — and some relationships or properties that are difficult or impossible to compute can, in some cases, be determined by humans and fed back into the system [150].

5.4 Systemic Needs

Not only do users have particular needs for individual items and recommendation lists, they have needs from the system as well.

5.4.1 Coverage

While many user needs are satisfied by finding some good items, and missing some items does not cause a problem, there are other cases where the penalty for missing items is high — high recall is critical [61].[1] This can be the case in locating research literature, where it is undesirable to overlook prior work, but is even more critical in legal settings: missing prior work in a patent application or a relevant precedent while handling litigation can have drastic consequences.

The concept of *coverage* provides a useful framework for analyzing how a recommender algorithm does or does not exhibit this problem. Simply put, the coverage with respect to a user is the fraction of the total item domain which the algorithm could possibly recommend to the user (in typical collaborative filtering algorithms, this is the set of items for which predictions can be generated for the user). In a user–user recommender, the ability to recommend an item i is dependent on the user having potential neighbors which have rated i; in an item–item recommender, it is dependent on the user having ratings for items in i's list of neighbors. Other algorithms can have similar limitations on their ability to recommend particular items based on the information they possess.

One way to alleviate coverage problems is by falling back to a baseline recommender when the collaborative filter cannot generate a prediction. Hybridizing algorithms with different coverage profiles can also help improve coverage. If the composite recommenders are of significantly different structure, this method can increase the system's ability to recommend some items. Latent factor models, such as SVD and PLSA, can also provide good coverage because the latent factors can connect users to items that they would not be connected to just through item or user neighborhoods.

McLaughlin and Herlocker [95] proposed a modified precision metric to measure the ability of a recommender to recommend items which takes coverage into account, penalizing algorithms with low coverage.

[1] Recall can rarely be directly measured. Data sets only contain rated items, while recall pertains to fining all relevant items regardless of whether they are already known to the user.

They then showed that a modified user–user recommender that operates by propagating belief distributions over rating offsets between similar users is able to achieve much better coverage at the expense of some accuracy compared to standard user–user and item–item CF. The balance with regards to coverage, accuracy, and computational requirements will vary by domain and application.

In addition to the impact of coverage on user needs, the system provider may also have an interest in coverage — if there are items which cannot be recommended, then it may be more difficult to sell those products.

5.4.2 Robustness

In order for recommendations to be useful to users, they need to be robust and fair. Recommendations should reflect the true opinions of legitimate users of the site, with minimal degradation due to natural noise and free from bias introduced by malicious users attempting to manipulate the system's ratings.

Having considered naturally-occurring noise in *Dealing with Noise*, we turn here to considerations for malicious users. An unscrupulous manufacturer desiring to sell more of their product may attempt to create accounts on e-commerce sites using recommender systems and strategically rating their product highly in order to cause it to be recommended to more users. Similarly, other groups may want to destroy a particular product's recommendation level or otherwise manipulate the recommendation system's output. Examples of this are so-called "Google Bombs", where link profiles are set up to cause a particular page to rate highly for specific searches, such as "miserable failure" returning the White House page for Bush's administration as the first result. Similar attacks have been carried out against Amazon.com, manipulating the "products you might also like" lists — a group of attackers succeeded in getting a book on gay sex listed as a related item for a Pat Robertson book.

Attacks based on false profiles, sometimes called *shilling* or *sybil* attacks, have received a fair amount of treatment in the literature. O'Mahony et al. [108] have examined the robustness of user-user

collaborative filtering to such attacks. Lam and Riedl [82] found that profile-based attacks can influence predictions and recommendation, but that item–item CF is less vulnerable than user–user to such attacks. Matrix factorization approaches (specifically PLSA) are also fairly robust to attack [103]. Further, it is difficult to reliably detect the presence of malicious user profiles using standard recommender accuracy metrics, although classifer-based methods can detect some errant profiles [102]. Resnick and Sami [120] proposed a reputation-based system, the *influence limiter*, which can be added to any collaborative filtering system and uses reputation scores, computed by determining which users contribute to accurate prediction of a target user's ratings, to limit the influence of individual users on the target user's recommendations.

Many currently known attacks exploit knowledge of the recommender algorithm in use, and sometimes additional information such as the average rating for a given item. Sometimes specific weaknesses can cause a marked increase in attack success. For example, recommenders based on Pearson correlation without similarity damping are susceptible to shilling attacks with small attack profiles, as two users with few ratings in common have abnormally high Pearson correlations [108].

So far, most research on the robustness of collaborative filtering to manipulation has focused on a few specific attack methods, most notably the creation of false profiles with the goal of artificially boosting or depressing an item in recommendation lists. Little is known about other attack methods or attacks aimed at different goals (such as an attack attempting to discount the credibility of the system by causing it to generate spurious or bogus recommendations). Resnick and Sami [121] have shown that there is an inherent tradeoff between manipulation-resistance and taking advantage of the information in user ratings; in order to resist manipulation by an attacker capable of creating an unbounded number of shill profiles, a system must discard all user ratings.

5.4.3 Privacy

Recommendation is not the only need users have with respect to their relationship with a recommender system. Among other things, they

may also desire privacy: ratings data can communicate a fair amount about a user's preferences and habits, and users may want to restrict the extent to which that data can be discovered by other users in a system (or even the system itself). Further, privacy is a significant consideration when releasing data sets to the public. There has been notable success in taking anonymized data sets, such as the Netflix Prize set, and de-anonymizing them (determining the identities of particular users) [41, 105].

An immediate privacy concern, at least within the e-commerce domain, is the ability for users of a system to gain information about other users. Ramakrishnan et al. [116] analyzed this problem, finding that users who tastes straddle seemingly-disparate regions of the item space can be susceptible to identification by other users of the system with access to aggregate sales data. They advised refraining from using connections involving a small number of people in producing recommendations so that the observable behavior of the recommender is based on sufficiently large samples that it is infeasible to identify individual users. The requirements for this attack are fairly sophisticated, but feasible with vendors sharing data and utilizing third-party personalization services.

Research has also considered recommendation schemes that allow users to receive recommendations and/or predictions without any central party knowing all users' ratings. Canny [26, 27] proposed factor analysis with this goal in mind, using homomorphic encryption and zero-knowledge protocols to allow users to collaboratively compute a recommendation model without disclosing their individual ratings. Other matrix factorization techniques such as singular value decomposition have also been deployed to implement decentralized recommenders: the PocketLens algorithm used SVD with encryption to perform decentralized collaborative filtering on hand-held devices [101] and Polat and Du [113] proposed using SVD with randomly-perturbed data to preserve privacy.

Hybrid methods, involving central servers that are partially trusted, have also been considered. Notable among these is Alambic, an architecture which separates purchasing and recommendation [8]. It calls for recommendation providers to be distinct from e-commerce providers,

so that the e-commerce vendor does not know a user's history and the recommendation provider does not know their current activity. There have also been proposals to have users share and aggregate their ratings before submitting them to the central server [138].

Kobsa [75] provide a thorough analysis of the privacy concerns in personalization systems in general, including legally imposed conditions. We refer you to that work for further reading on this aspect of the user–recommender relationship.

5.5 Summary

Within results of equivalent accuracy or numerical quality, there is still sufficient variation that algorithms with equivalent accuracy performance can produce recommendations that are qualititatively different from each other in terms of user response. This provides opportunity to tailor a recommender system deployment to meet the particular needs of its users. This can be done by tuning the algorithm to the expected needs [99], providing (or automatically adjusting) parameters to influence its behavior [156], or switching algorithms based on the user's current task [156].

As with user interface design (and, indeed, design for any system that interacts with humans), clearly identifying the user tasks the system is intended support and what users expect from the system provides a useful framework for the design and evaluation of a recommender system. At the end of the day, users are more concerned with having their needs met or otherwise getting value from a system than numeric measures of its accuracy.

6

User Experience

In order to move from predicting preference to meeting user needs, recommender systems must interact with their users in some fashion. They need to collect ratings, frequently requiring user interaction (although this interaction is occasionally indirect), and they must present the recommendations or predictions back to the user.

Recommenders also have the opportunity to integrate more broadly with user experience, particularly the social context in which recommendations are received and used. There have also been projects to use recommenders to affect user behavior in ways beyond purchasing products or seeking entertainment.

This section surveys a variety of work that has been done on user interaction with recommenders and on the ways in which recommenders interact with user experience and social contexts. Better understanding of human–recommender interaction and how recommenders affect and are affected by the people who use them is also an important direction for further recommender systems research.

6.1 Soliciting Ratings

As discussed in *Rating Scales*, a variety of methods have been tried for soliciting ratings from users of recommender systems. GroupLens

augmented standard newsreaders with buttons or commands for applying ratings of 1–5 stars [119]. Jester provided a continuous bar that recorded the position at which users clicked it as their rating (using an HTML image map) [50]. MovieLens uses drop-down menus or images for users to select 1–5 star (with half-star) ratings; some early recommenders interacted with users via e-mail.

The most common interface construct for allowing users to provide ratings is a rating widget of some kind displayed with the item either on an item info details or in its entry in a list. In general, the rating interface will need to integrate well with the rest of the application or site's user experience, and standard human–computer interaction design principles apply (including using interface devices familiar to users from other systems). There is currently a good deal of inconsistency, even within a single system, with regards to how ratings are collected; Karvonen et al. [70] provide a heuristic evaluation of several popular web sites with rating or reputation interfaces and demonstrates a number of consistency issues. Live deployments are not all bad, however; Netflix star ratings and Pandora's thumbs up/down are good examples of current best practices in rating interfaces.

Users prefer more granularity in their ratings interfaces — on a 5-star scale, they like to be able to give half-star ratings — so it is beneficial for the user experience to allow a relatively fine-grained rating process, but the increased granularity of ratings will not necessarily translate into more accurate recommendations [34].

6.2 Presenting Recommendations

Once the system has collected user ratings and computed recommendations, it must communicate those recommendations to the user so that they can evaluate and potentially act on the recommended items. Communicating predictions is fairly straightforward — in an item listing or details view, the predicted preference can be shown as a feature of the item. MovieLens does this, replacing the predicted preference with the user's actual rating when it is an item they have rated.

Methods of presenting recommendations have changed as communications technology evolved. In the early 1990s, when the Web was

young and had not yet achieved widespread usage, recommenders such as BellCore's video recommender interacted with the user via e-mail: users e-mailed ratings of movies to `videos@bellcore.com` and received recommendations in an e-mail reply [62]. The majority of recommender systems are currently built into web sites. TV set-top boxes are also used as a delivery mechanism for recommendations [4], soliciting ratings via the remote control and presenting the user with a list of recommended shows with their air dates. Increasing usage of sophisticated cell phones have made mobile recommendations increasingly a reality [100]. Delivering recommendations via devices other than traditional PCs has the potential to bring recommendation much closer to the actual context of use for a wider variety of items — TV shows or movies can be immediately watched, and users can receive product recommendations while in the store or at the theater.

There is more diversity in approaches to showing recommendations. Some systems, such as MovieLens, merely have a view where items are displayed in a search results fashion ordered by predicted preference. Others display various forms of recommendations in-line with the browsing experience: many pages on Amazon.com, such as product information pages and shopping baskets, show recommendations based on the user's past history or the actions taken by other users with respect to the currently displayed item.

In environments where the cost of consumption is low and recommendation process is ongoing, the system can simply provide the user with the recommended item. This is done by Jester, where the system shows the user jokes and asks for ratings until the user decides to leave [50]. Music recommenders such as Pandora also frequently do this — Pandora starts playing the next song, and users can skip or thumb-down the song if they don't like it.

One specific aspect of communicating recommendations to users is that of explaining why particular items were recommended. Recommender systems are frequently "black boxes", presenting recommendations to the user without any explanation of why the user might like the recommended item. Explanations seem to be useful, however, for helping the user be confident in the provided recommendations and to make better decisions based on them [60].

Explanation methods can stem from a variety of goals. Tintarev [144] identifies seven potential goals for explanations of recommendations: transparency of the recommender algorithm, scrutability of the results, trustworthiness of the recommendations, effectiveness for supporting decisions, persuasiveness, efficiency of decision-making, and satisfaction with the recommendation process.

User modelling work by Cook and Kay [32] demonstrated a means for showing users the system's model of them and showed that many users were interested in this information. The earliest systematic evaluation of recommender explanations was by Herlocker et al. [60]. While it did not show a change in users' enjoyment of recommended items based on explanations, that study did document a significant user interest in explanations of the recommendations they received. Bilgic and Mooney [17] argue that for many domains, user satisfaction with the item is more important than whether they act on the recommendation: satisfaction and effectiveness are more critical to evaluate than persuasiveness. A later focus group study suggested that explanations can help with user satisfaction by helping them better know what to expect [145].

For other stakeholders the balance of explanatory goals can be different. An e-commerce site typically uses recommendations to increase sales volume, increasing the importance of persuasion as a goal. This has limits, however — if the system has a reputation for recommending bad items it can lose trust with users and suffer in the long run.

Some work has been done on assessing the impact of the recommender interface on user reception of recommendations [54, 60]. Ultimately, however, this will likely have a good number of domain- and system-specific attributes. Further, it is an example of one of the aspects of recommender evaluation that cannot be done off-line: to measure user response to recommendation, it is necessary to test a system with actual users.

6.3 Recommending in Conversation

Many recommender systems simply provide the user with a set of recommendations or predictions, and the only feedback mechanism provided to the user for refining future recommendations is the ability

to rate (or purchase) more items. *Conversational recommenders* seek to expand that, soliciting the user's input to more accurately meet their needs through an ongoing cycle of recommendation and feedback (a *conversation*). In some ways, the resulting user experience is similar to that of Grundy [123], in which the user could reject the suggested book and receive a new suggestion. Conversational recommendation has predominantly been explored in the context of case-based reasoning systems [3, 46, 94] where recommendation is of a more content-based than collaborative nature.

McGinty and Smyth [94] divide the kinds of feedback which are solicited by various recommenders into four types: *value elicitation*, where the user provides a specific value for some feature as a hard constraint (e.g., "built by Sony"); *ratings*, where the user provides ratings for items; *critique*, where the user provides a specific correction to the system's recommendation (e.g., "like this, but with more megapixels"); and *preference*, where the user selects the most desirable of the recommended items and the system replies with more items similar to the selected item.[1]

6.4 Recommending in Social Context

So far, the recommendation systems considered have all treated the user as a single, independent entity who will consume the recommendation without interaction with others. There has been work, however, on making recommender systems aware of social contexts and connections between users.

6.4.1 Group Recommenders

Group recommenders have the goal of recommending items to be experienced by a group of people, aiming to find an item (such as a movie or travel destination) which all members will enjoy [92]. PolyLens provided movie recommendations to groups by using MoveLens's user–user collaborative filter to generate predictions for each user, producing the

[1] Under this taxonomy, traditional ratings-based recommender systems become a special case of conversational recommenders without an explicitly articulated recommend-refine cycle.

group's recommendations by merging the individual predictions using the "principle of least misery": the predicted preference for a movie was the lowest preference predicted for any group member [107]. Users enjoyed having the group recommendations, although there are privacy tradeoffs as some information about users' preferences can be inferred by their fellow group members.

Jameson [66] describes a number of challenges in building group recommenders. First is preference collection: while PolyLens simply used the ratings each user had previously provided, in some domains it can be desirable to allow users to see each other's stated preferences and/or interact with each other around the preference-stating process. Second, the preferences need to be aggregated and the recommendations produced. With user–user CF algorithms, this can be done either by blending individual recommendation lists for each user or by creating a composite pseudo-user for the group and generating recommendations for that user [107]. In either of these cases, it is necessary to have some aggregation function for combining preferences. O'Connor considered several methods for PolyLens, including minimum preference, maximum preference, and average preference; Jameson points out that it is important for this preference function to be *non-manipulable* so that no user can manipulate the group's recommendations by falsely stating their preferences. For his Travel Decision Forum system, he selected median as the aggregation function. This requirement is important, as manipulability has been a problem for deployed group recommenders. The initial trial of the MusicFX system for selecting fitness center background music allowed an individual to force a music change by giving the current music the minimum rating [92].

After the system has produced a set of recommendations for the group, it needs to provide the group with those recommendations and, in some cases, support the group as they finalize their decision. Jameson spends some time on the nuances of facilitating group decision around the resulting recommendation lists.

6.4.2 Incorporating Trust in Recommenders

Even when recommendations are intended for individual users, users may have relationships with other users that can be used as an

additional source of information in the recommendation process. This is particularly the case with the rise of social networking, with extensive data available about peoples' connections to each other.

Various authors have proposed methods for integrating information about user's connections into recommender systems in the form of "trust-based recommenders". Massa and Avesani proposed a method for using users' statements of their trust in other users' opinions to weight user ratings by estimated trust rather than similarity in user–user CF when producing predictions and recommendations and built a ski mountaineering site around it [12, 91]. Golbeck [47] used a similar integration method, based on a different trust estimation algorithm, for movie recommendation.

Trust is a complicated concept, and representing and estimating it computationally is difficult. While many systems have a good deal of information about peoples' connections, how those connections correspond to trust in recommendation is not obvious. Even annotating the network with trust ratings is problematic, with confidence and notions such as active distrust making a useful single "estimated trust" value an elusive goal. There is continued work, however, on various methods for estimating and propagating trust through social networks [12, 47, 48, 114, 155]. Ziegler and Golbeck [154] found that trust and conventional rating similarities are correlated. Guy et al. also found that users tended to find recommendations of web sites, discussion forums, and other social software artifacts more interesting when they were recommended from the user's social connections rather than users with similar preference histories [54].

6.4.3 Recommending Social Connections

Some recommender projects have gone beyond recognizing that recommendations are used in social contexts by using recommenders to alter or enhance the structure of that social context — recommending social connections or friends.

Kautz et al. [72] did early work in this direction, building the Referral Web system that allowed users to query their social network;

McDonald [93] extended these ideas by building a recommender for individuals with particular expertise from the user's social network or organization. These systems enabled users to find people in their social network or organization that could provide them with particular expertise or otherwise meet a need they had.

Terveen and McDonald [143] provide a survey and overview of social connection recommendation, covering information-need-based recommendation such as that of Kautz et al. and McDonald as well as other applications of social connection recommendation. Recent work has focused on recommending connections in the contexts of online social networks [28].

6.5 Shaping User Experience with Recommendations

The interaction between recommender systems and user experience goes beyond providing a useful user experience around recommendations. Recommendations can also be used to alter user experience and behavior. Cosley et al. [34] found that predictions influence user rating behavior, but there is interest in using recommender systems to influence user behavior in broader contexts.

SuggestBot, an implementation of *intelligent task routing*, uses recommender algorithms to suggest pages that Wikipedia editors may want to edit [33]. Intelligent task routing aims to help people do work on community-maintained projects by guiding them to work that needs to be done and which matches with their interests as inferred from prior work history.

In online social networking tools, recommenders have been successfully deployed to improve user experience and engagement. Recommending connections to existing users of IBM's internal social networking site SocialBlue (formerly known as BeeHive) resulted in users expanding their connections. Users' social network information was more useful for finding people they already knew on the system, but recommender algorithms based on similarity in user profile contents were more effective at helping users discover new friends [28]. Recommending

social connections and profile content to new users resulted in the users being more active on the site [42].

Recommenders therefore have the potential not just to meet information needs, but to shape and guide user behavior both on-line and in the real world.

7

Conclusion and Resources

Recommender systems have become ubiquitous. People use them to find books, music, news, smart phones, vacation trips, and romantic partners. Nearly every product, service, or type of information has recommenders to help people select from among the myriad alternatives the few they would most appreciate. Sustaining these commercial applications is a vibrant research community, with creative interaction ideas, powerful new algorithms, and careful experiments. Still, there are many challenges for the field, especially at the interaction between research and commercial practice. We have framed these challenges here in a five part taxonomy.

Algorithms: Recommender systems researchers have developed a suite of highly effective algorithms for the basic problem of recommending a set of substitutable goods from a large population of similar goods to individual users. There are however many remaining algorithmic challenges, most involving richer sets of data about the users, the items, the interactions between the users and the items, or the relationships among groups of users or groups of items. For instance, how can such sources

as Amazon reviews and Twitter posts about items be incorporated into recommendations. We expect to see a wide variety of approaches, but are particularly optimistic about algorithms that generalize to process multiple parallel matrices of user and item data simultaneously.

Data: The data used by recommender systems are sometimes biased in unexpected ways that can have a dramatic effect on outcomes. For instance, in designing algorithms for selecting items for new users to rate in MovieLens we found that the original MovieLens algorithm, which was performing poorly in practice, nearly always scored higher in offline tests than the newer algorithms we were designing. The reason is that since new users had been using the original algorithm for years, the dataset we were using for offline testing was much more likely to have a rating for items chosen by that algorithm than by any new algorithm. Similarly, nearly all rating data sets are strongly biased toward high ratings, because users are careful to only choose to consume items they suspect they will like. This class of problems is challenging to solve in general, but there are some elegant approaches emerging for specific instances. For instance, researchers have shown that datasets can be effectively statistically unbiased for some popularity biases [88].

User experience: The goal of recommender systems is to improve user experience. In both research and practice, crafting the user experience to fit the application and the user lifecycle remains a substantial challenge. One challenge is to adapt the nature of recommendations as the user gains more experience with the recommender — a new user may need more verifiable recommendations and may lack the trust needed for high-risk recommendations. A similar challenge is how to balance serving the individual now (with the best available recommendation) vs. serving the individual and community long-term (with recommendations that help the recommender system learn). These challenges vary with different domains, and particularly with different usage patterns. Part of the research challenge is to design interfaces that give users control over

the recommendation process without overwhelming the user or rendering the tool too complicated for novice users.

Evaluation and metrics: Evaluation of recommender systems has advanced significantly over the past decade, but many challenges remain. Even with a better understanding of the relationships among different algorithm performance metrics, the field struggles to standardize evaluation. As a result, too often research results are incomparable with prior published results, even when based on the same data sets. Examples of this problem include the lack of standard treatment of items for which the recommender is unable to make a prediction. The broader goal of user-centered holistic evaluation, including A/B testing of the short- and long-term effects of recommendation differences, is still met by only a few research groups and companies that have the live systems and resources for such evaluation. Deploying innovative recommenders is still too hard, and there is a substantial need for research platforms where innovations can be tested without first building up a community of thousands of users.

Social impact: Because collaborative filtering recommender systems must by their nature collect substantial personalized data about their users, they face important privacy and security challenges. Researchers have been approaching these challenges head-on by attempting to develop ways to collect and store data in such a way that extracting personalized data is provably difficult. Recommender systems also have some unique community challenges. One such challenge is about the relationship between the recommender system and its community of users. How do the users know that the recommender system is providing honest recommendations? Furthermore, the existence of the recommender may influence the structure of the community over time. If users are choosing items to read based on personalized recommendations, over time they may cluster into groups of like-minded individuals, *balkanizing* the community into groups who seldom interact with people with whom they do not agree. We hope in the future to see recommender systems

that explicitly react to combat this type of damage to the community structure, ensuring that not only individuals, but also the surrounding community benefit from the existence of the recommender.

7.1 Resources

Given the broad application of recommenders and the substantial set of open problems, the field of recommender systems research promises to remain active for years to come. We conclude this survey with a brief review of resources we feel would be useful for both academic and industrial explorations of recommender systems.

Publication venues: Recommender systems research continues to progress rapidly, with many papers appearing each year in venues including: ACM Recommender Systems (RecSys) User Modeling, Adaptation and Personalization (UMAP) ACM Conference on Human Factors in Computing Systems (CHI) ACM Conference on Computer-Supported Cooperative Work (CSCW) ACM Conference on Information Retrieval (SIGIR) ACM Conference on Knowledge Discovery and Data Mining (KDD) ACM Transactions on Information Systems User Modeling and User-Adapted Interaction and many other workshops, conferences, and journals.

Software implementations: There are a variety of collaborative filtering implementations available and ready for use in projects, many under open source licenses, including the following:

- LensKit by GroupLens Resesarch, Java. `http://lenskit.grouplens.org`
- MyMediaLite by MyMedia at University of Hildesheim, C#/.NET. `http://www.ismll.uni-hildesheim.de/mymedialite/`
- easyrec, Java with RESTful HTTP API. `http://easyrec.org`

- Mahout Taste by the Apache project, Java. http://mahout.apache.org/
- Cofi by Daniel Lemire et al., Java. http://www.nongnu.org/cofi/
- SUGGEST by George Karypis, binary-only, C API. http://glaros.dtc.umn.edu/gkhome/suggest/download

Data sets: For evaluating and tuning recommender performance, commonly-used, readily-available data sets include the following:

- MovieLens: movie ratings (100 K, 1 M, and 10 M ratings sets, the latter includes 100 K tag applications). With the availability of larger data sets, we do not recommend using the 100 K for other than development and preliminary testing; larger data sets will provide more robust validation of algorithms. http://www.grouplens.org/node/73
- Jester: ratings of jokes. http://eigentaste.berkeley.edu/dataset/
- BookCrossing: book ratings with user demographic information. http://www.informatik.uni-freiburg.de/~cziegler/BX/
- WebScope — Yahoo! provides a number of web user behavior data sets from various Yahoo! services through their WebScope program. http://webscope.sandbox.yahoo.com
- RecLab — RichRelevance provides a recommendation evaluation framework and simulated data set for evaluating recommender performance for e-commerce. http://code.richrelevance.com/reclab.

Work on this survey was funded by the National Science Foundation under grant IIS 05-34939.

References

[1] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[2] E. Agichtein, E. Brill, S. Dumais, and R. Ragno, "Learning user interaction models for predicting web search result preferences," in *ACM SIGIR '06*, pp. 3–10, ACM, 2006.

[3] D. Aha and L. Breslow, "Refining conversational case libraries," in *Case-Based Reasoning Research and Development*, vol. 1266 of *Lecture Notes in Computer Science*, pp. 267–278, Springer, 1997.

[4] K. Ali and W. van Stam, "TiVo: Making show recommendations using a distributed collaborative filtering architecture," in *ACM KDD '04*, pp. 394–401, ACM, 2004.

[5] X. Amatriain, J. Pujol, and N. Oliver, "I like it... I like it not: Evaluating user ratings noise in recommender systems," in *UMAP 2009*, vol. 5535 of *Lecture Notes in Computer Science*, pp. 247–258, Springer, 2009.

[6] X. Amatriain, J. M. Pujol, N. Tintarev, and N. Oliver, "Rate it again: Increasing recommendation accuracy by user re-rating," in *ACM RecSys '09*, pp. 173–180, ACM, 2009.

[7] Amazon.com, "Q4 2009 Financial Results," Earnings Report Q4-2009, January 2010.

[8] E. Aïmeur, G. Brassard, J. Fernandez, and F. M. Onana, "Alambic: A privacy-preserving recommender system for electronic commerce," *International Journal of Information Security*, vol. 7, no. 5, pp. 307–334, October 2008.

[9] T. Amoo and H. H. Friedman, "Do numeric values influence subjects' responses to rating scales?," *Journal of International Marketing and Marketing Research*, vol. 26, pp. 41–46, February 2001.
[10] A. Ansari, S. Essegaier, and R. Kohli, "Internet recommendation systems," *Journal of Marketing Research*, vol. 37, no. 3, pp. 363–375, August 2000.
[11] C. Avery and R. Zeckhauser, "Recommender systems for evaluating computer messages," *Communications of the ACM*, vol. 40, no. 3, pp. 88–89, ACM ID: 245127, March 1997.
[12] P. Avesani, P. Massa, and R. Tiella, "A trust-enhanced recommender system application: Moleskiing," in *ACM SAC '05*, pp. 1589–1593, ACM, 2005.
[13] M. Balabanović and Y. Shoham, "Fab: Content-based, collaborative recommendation," *Communications of the ACM*, vol. 40, no. 3, pp. 66–72, 1997.
[14] R. M. Bell and Y. Koren, "Scalable collaborative filtering with jointly derived neighborhood interpolation weights," in *IEEE ICDM 2007*, pp. 43–52, 2007.
[15] J. Bennett and S. Lanning, "The netflix prize," in *KDD Cup and Workshop '07*, 2007.
[16] M. W. Berry, S. T. Dumais, and G. W. O'Brien, "Using linear algebra for intelligent information retrieval," *SIAM Review*, vol. 37, no. 4, pp. 573–595, December 1995.
[17] M. Bilgic and R. J. Mooney, "Explaining recommendations: Satisfaction vs. promotion," in *Beyond Personalization 2005: A Workshop on the Next Stage of Recommender Systems Research*, pp. 13–18, 2005.
[18] D. Billsus and M. J. Pazzani, "Learning collaborative information filters," in *AAAI 2008 Workshop on Recommender Systems*, 1998.
[19] D. Billsus and M. J. Pazzani, "A personal news agent that talks, learns and explains," in *AGENTS '99*, pp. 268–275, ACM, 1999.
[20] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, March 2003.
[21] M. Brand, "Fast online svd revisions for lightweight recommender systems," in *SIAM International Conference on Data Mining*, pp. 37–46, SIAM, 2003.
[22] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *UAI 1998*, pp. 43–52, AAAI, 1998.
[23] P. Brusilovsky, "Methods and techniques of adaptive hypermedia," *User Modeling and User-Adapted Interaction*, vol. 6, no. 2, pp. 87–129, July 1996.
[24] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370, November 2002.
[25] R. Burke, "Evaluating the dynamic properties of recommendation algorithms," in *ACM RecSys '10*, pp. 225–228, ACM, 2010.
[26] J. Canny, "Collaborative filtering with privacy," in *IEEE Symposium on Security and Privacy 2002*, pp. 45–57, IEEE Computer Society, 2002.
[27] J. Canny, "Collaborative filtering with privacy via factor analysis," in *ACM SIGIR '02*, pp. 238–245, ACM, 2002.
[28] J. Chen, W. Geyer, C. Dugan, M. Muller, and I. Guy, "Make new friends, but keep the old: Recommending people on social networking sites," in *ACM CHI '09*, pp. 201–210, ACM, 2009.

[29] L. Chen and P. Pu, "Evaluating critiquing-based recommender agents," in *AAAI 2006*, vol. 21, pp. 157–162, AAAI, 2006.

[30] Y. H. Chien and E. I. George, "A bayesian model for collaborative filtering," in *7th International Workshop on Artificial Intelligence and Statistics*, 1999.

[31] B. H. Clark, "Marketing performance measures: History and interrelationships," *Journal of Marketing Management*, vol. 15, no. 8, pp. 711–732, November 1999.

[32] R. Cook and J. Kay, "The justified user model: A viewable, explained user model," in *User Modelling 1994*, 1994.

[33] D. Cosley, D. Frankowski, L. Terveen, and J. Riedl, "SuggestBot: using intelligent task routing to help people find work in wikipedia," in *ACM IUI '07*, pp. 32–41, ACM, 2007.

[34] D. Cosley, S. K. Lam, I. Albert, J. A. Konstan, and J. Riedl, "Is seeing believing?: How recommender system interfaces affect users' opinions," in *ACM CHI '03*, pp. 585–592, ACM, 2003.

[35] B. Dahlen, J. Konstan, J. Herlocker, N. Good, A. Borchers, and J. Riedl, "Jump-starting MovieLens: User benefits of starting a collaborative filtering system with 'dead data," Technical Report 98-017, University of Minnesota, Minneapolis, MN, March, 1998.

[36] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[37] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, ArticleType: research-article/Full publication date: 1977/Copyright © 1977 Royal Statistical Society, Janaury 1977.

[38] M. Deshpande and G. Karypis, "Item-based top-N recommendation algorithms," *ACM Transactions on Information Systems*, vol. 22, no. 1, pp. 143–177, 2004.

[39] M. D. Ekstrand, P. Kannan, J. A. Stemper, J. T. Butler, J. A. Konstan, and J. T. Riedl, "Automatically building research reading lists," in *ACM RecSys '10*, pp. 159–166, ACM, 2010.

[40] B. Fields, C. Rhodes, and M. d'Inverno, "Using song social tags and topic models to describe and compare playlists," in *Workshop on Music Recommendation and Discovery 2010*, 633, CEUR, September 2010.

[41] D. Frankowski, D. Cosley, S. Sen, L. Terveen, and J. Riedl, "You are what you say: Privacy risks of public mentions," in *ACM SIGIR '06*, pp. 565–572, ACM, 2006.

[42] J. Freyne, M. Jacovi, I. Guy, and W. Geyer, "Increasing engagement through early recommender intervention," in *ACM RecSys '09*, pp. 85–92, ACM, 2009.

[43] H. H. Friedman and T. Amoo, "Rating the rating scales," *Journal of Marketing Management*, vol. 9, no. 3, pp. 114–123, 1999.

[44] S. Funk, "Netflix update: Try this at home," http://sifter.org/~simon/journal/20061211.html, Archived by WebCite at http://www.webcitation.org/5pVQphxrD, December 2006.

[45] R. Garland, "The mid-point on a rating scale: Is it desirable?," *Marketing Bulletin*, vol. 2, pp. 66–70, May 1991.

[46] M. Göker and C. Thompson, "Personalized conversational case-based recommendation," in *Advances in Case-Based Reasoning*, vol. 1898 of *Leceture Notes in Computer Science*, pp. 29–82, Springer, 2000.

[47] J. Golbeck, "Generating predictive movie recommendations from trust in social networks," in *International Conference on Trust Management*, vol. 3986 of *Lecture Notes in Computer Science*, pp. 93–104, Springer, 2006.

[48] J. Golbeck, "Trust on the World Wide Web: A survey," *Foundations and Trends® in Web Science*, vol. 1, no. 2, pp. 131–197, 2006.

[49] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Communications of the ACM*, vol. 35, no. 12, pp. 61–70, 1992.

[50] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A constant time collaborative filtering algorithm," *Information Retrieval*, vol. 4, no. 2, pp. 133–151, July 2001.

[51] G. Gorrell, "Generalized Hebbian algorithm for incremental singular value decomposition in natural language processing," in *EACL 2006*, pp. 97–104, ACL, 2006.

[52] M. Grigoriev, "Intelligent multimedia management system," 2003.

[53] A. Gunawardana and G. Shani, "A survey of accuracy evaluation metrics of recommendation tasks," *Journal of Machine Learning Research*, vol. 10, pp. 2935–2962, 2009.

[54] I. Guy, N. Zwerdling, D. Carmel, I. Ronen, E. Uziel, S. Yogev, and S. Ofek-Koifman, "Personalized recommendation of social software items based on social relations," in *ACM RecSys '09*, pp. 53–60, ACM, 2009.

[55] D. L. Hansen and J. Golbeck, "Mixing it up: Recommending collections of items," in *ACM CHI '09*, pp. 1217–1226, ACM, 2009.

[56] F. M. Harper, X. Li, Y. Chen, and J. A. Konstan, "An economic model of user rating in an online recommender system," in *User Modeling 2005*, vol. 3538 of *Lecture Notes in Computer Science*, pp. 307–316, Springer, August 2005.

[57] D. A. Harrison and K. J. Klein, "What's the difference? Diversity constructs as separation, variety, or disparity in organizations," *Academy of Management Review*, vol. 32, no. 4, pp. 1199–1228, Oct 2007.

[58] J. Herlocker, J. A. Konstan, and J. Riedl, "An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms," *Information Retrieval*, vol. 5, no. 4, pp. 287–310, 2002.

[59] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," in *ACM SIGIR '99*, pp. 230–237, ACM, 1999.

[60] J. L. Herlocker, J. A. Konstan, and J. Riedl, "Explaining collaborative filtering recommendations," in *ACM CSCW '00*, pp. 241–250, ACM, 2000.

[61] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems*, vol. 22, no. 1, pp. 5–53, 2004.

[62] W. Hill, L. Stead, M. Rosenstein, and G. Furnas, "Recommending and evaluating choices in a virtual community of use," in *ACM CHI '95*, pp. 194–201, ACM Press/Addison-Wesley Publishing Co., 1995.

[63] W. Hill and L. Terveen, "Using frequency-of-mention in public conversations for social filtering," in *ACM CSCW '96*, pp. 106–112, ACM, 1996.

[64] T. Hofmann, "Probabilistic latent semantic indexing," in *ACM SIGIR '99*, pp. 50–57, ACM, 1999.

[65] T. Hofmann, "Latent semantic models for collaborative filtering," *ACM Transactions on Information Systems*, vol. 22, no. 1, pp. 89–115, 2004.

[66] A. Jameson, "More than the sum of its members: Challenges for group recommender systems," in *Working Conference on Advanced Visual Interfaces*, pp. 48–54, ACM, 2004.

[67] X. Jin, Y. Zhou, and B. Mobasher, "Web usage mining based on probabilistic latent semantic analysis," in *ACM KDD '04*, pp. 197–205, ACM, 2004.

[68] D. Kahneman, P. P. Wakker, and R. Sarin, "Back to bentham? Explorations of experienced utility," *The Quarterly Journal of Economics*, vol. 112, no. 2, pp. 375–405, ArticleType: research-article/Issue Title: In Memory of Amos Tversky (1937–1996)/Full publication date: May, 1997/Copyright© 1997 The MIT Press, May 1997.

[69] J. Karlgren, "Newsgroup clustering based on user behavior — a recommendation algebra," Technical Report, European Research Consortium for Informatics and Mathematics at SICS, 1994.

[70] K. Karvonen, S. Shibasaki, S. Nunes, P. Kaur, and O. Immonen, "Visual nudges for enhancing the use and produce of reputation information," in *Workshop on User-Centric Evaluation of Recommender Systems and Their Interfaces*, September 2010.

[71] G. Karypis, "Evaluation of item-based top-N recommendation algorithms," in *ACM CIKM '01*, pp. 247–254, ACM, 2001.

[72] H. Kautz, B. Selman, and M. Shah, "Referral Web: Combining social networks and collaborative filtering," *Communications of the ACM*, vol. 40, no. 3, pp. 63–65, 1997.

[73] B. Kitts, D. Freed, and M. Vrieze, "Cross-sell: A fast promotion-tunable customer-item recommendation method based on conditionally independent probabilities," in *ACM KDD '00*, pp. 437–446, ACM, 2000.

[74] B. P. Knijnenburg, L. Schmidt-Thieme, and D. G. Bollen, "Workshop on user-centric evaluation of recommender systems and their interfaces," in *ACM RecSys '10*, p. 383, ACM, 2010.

[75] A. Kobsa, "Privacy-enhanced web personalization," in *The Adaptive Web*, pp. 628–670, Springer, 2007.

[76] R. Kohavi, R. M. Henne, and D. Sommerfield, "Practical guide to controlled experiments on the web: Listen to your customers not to the HiPPO," in *ACM KDD '07*, pp. 959–967, ACM, ACM ID: 1281295, 2007.

[77] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, "Controlled experiments on the web: Survey and practical guide," *Data Mining and Knowledge Discovery*, vol. 18, no. 1, pp. 140–181, 2008.

[78] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl, "GroupLens: applying collaborative filtering to Usenet news," *Communications of the ACM*, vol. 40, no. 3, pp. 77–87, 1997.

[79] Y. Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," in *ACM KDD '08*, pp. 426–434, ACM, 2008.

[80] Y. Koren, "Collaborative filtering with temporal dynamics," *Communications of the ACM*, vol. 53, no. 4, pp. 89–97, 2010.

[81] M. Kurucz, A. A. Benczúr, and K. Csalogány, "Methods for large scale SVD with missing values," in *KDD Cup and Workshop 2007*, August 2007.

[82] S. K. Lam and J. Riedl, "Shilling recommender systems for fun and profit," in *ACM WWW '04*, pp. 393–402, ACM, 2004.

[83] T. Landgrebe, P. Paclik, R. Duin, and A. Bradley, "Precision-recall operating characteristic (P-ROC) curves in imprecise environments," in *ICPR 2006*, pp. 123–127, IEEE Computer Society, 2006.

[84] N. Lathia, "Evaluating Collaborative Filtering Over Time," PhD thesis, University College London, London, UK, June, 2010.

[85] N. Lathia, S. Hailes, and L. Capra, "Evaluating collaborative filtering over time," in *SIGIR 09 Workshop on the Future of IR Evaluation*, July 2009.

[86] N. Lathia, S. Hailes, and L. Capra, "Temporal collaborative filtering with adaptive neighbourhoods," in *ACM SIGIR '09*, pp. 796–797, ACM, 2009.

[87] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.

[88] B. M. Marlin and R. S. Zemel, "Collaborative prediction and ranking with non-random missing data," in *ACM RecSys '09*, pp. 5–12, ACM, 2009.

[89] B. M. Marlin, R. S. Zemel, S. Roweis, and M. Slaney, "Collaborative filtering and the missing at random assumption," in *UAI '07*, pp. 50–54, AUAI, 2007.

[90] F. J. Martin, "RecSys '09 industrial keynote: Top 10 lessons learned developing deploying and operating real-world recommender systems," in *ACM RecSys '09*, pp. 1–2, ACM, 2009.

[91] P. Massa and P. Avesani, "Trust-Aware collaborative filtering for recommender systems," in *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, vol. 3290 of *Lecture Notes in Computer Science*, pp. 275–301, Springer, 2004.

[92] J. F. McCarthy and T. D. Anagnost, "MusicFX: an arbiter of group preferences for computer supported collaborative workouts," in *ACM CSCW '98*, pp. 363–372, ACM, 1998.

[93] D. W. McDonald, "Evaluating expertise recommendations," in *ACM GROUP '01*, pp. 214–223, ACM, ACM ID: 500319, 2001.

[94] L. McGinty and B. Smyth, "Adaptive selection: An analysis of critiquing and preference-based feedback in conversational recommender systems," *International Journal of Electronic Commerce*, vol. 11, no. 2, pp. 35–57, 2006.

[95] M. R. McLaughlin and J. L. Herlocker, "A collaborative filtering algorithm and evaluation metric that accurately model the user experience," in *ACM SIGIR '04*, pp. 329–336, Sheffield, United Kingdom: ACM, 2004.

[96] S. McNee, S. Lam, J. Konstan, and J. Riedl, "Interfaces for eliciting new user preferences in recommender systems," in *User Modeling 2003*, vol. 2702, pp. 178–187, Springer, 2003.

[97] S. M. McNee, I. Albert, D. Cosley, P. Gopalkrishnan, S. K. Lam, A. M. Rashid, J. A. Konstan, and J. Riedl, "On the recommending of citations for research papers," in *ACM CSCW '02*, pp. 116–125, ACM, 2002.

[98] S. M. McNee, J. Riedl, and J. A. Konstan, "Being accurate is not enough: How accuracy metrics have hurt recommender systems," in *ACM CHI '06 Extended Abstracts*, pp. 1097–1101, ACM, 2006.

[99] S. M. McNee, J. Riedl, and J. A. Konstan, "Making recommendations better: An analytic model for human-recommender interaction," in *ACM CHI '06 Extended Abstracts*, pp. 1103–1108, ACM, 2006.

[100] B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl, "MovieLens unplugged: Experiences with an occasionally connected recommender system," in *ACM IUI '03*, pp. 263–266, ACM, 2003.

[101] B. N. Miller, J. A. Konstan, and J. Riedl, "PocketLens: Toward a personal recommender system," *ACM Transactions on Information Systems*, vol. 22, no. 3, pp. 437–476, 2004.

[102] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams, "Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness," *ACM Transactions on Internet Technology*, vol. 7, no. 4, p. 23, 2007.

[103] B. Mobasher, R. Burke, and J. Sandvig, "Model-based collaborative filtering as a defense against profile injection attacks," in *AAAI 2006*, pp. 1388–1393, AAAI, 2006.

[104] M. Morita and Y. Shinoda, "Information filtering based on user behavior analysis and best match text retrieval," in *ACM SIGIR '94*, pp. 272–281, Springer-Verlag, 1994.

[105] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *IEEE Symposium on Security and Privacy 2008*, pp. 111–125, IEEE Computer Society, 2008.

[106] D. Oard and J. Kim, "Implicit feedback for recommender systems," in *AAAI Workshop on Recommender Systems*, Madison, Wisconsin, 1998.

[107] M. O'Connor, D. Cosley, J. A. Konstan, and J. Riedl, "PolyLens: a recommender system for groups of users," in *ECSCW 2001*, pp. 199–218, Kluwer Academic Publishers, 2001.

[108] M. O'Mahony, N. Hurley, N. Kushmerick, and G. Silvestre, "Collaborative recommendation: A robustness analysis," *ACM Transactions on Internet Technology*, vol. 4, no. 4, pp. 344–377, 2004.

[109] M. P. O'Mahony, N. J. Hurley, and G. C. Silvestre, "Detecting noise in recommender system databases," in *ACM IUI '06*, pp. 109–115, ACM, 2006.

[110] A. Paterek, "Improving regularized singular value decomposition for collaborative filtering," in *KDD Cup and Workshop 2007*, August 2007.

[111] D. M. Pennock, E. Horvits, and C. L. Giles, "Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering," in *AAAI 2000*, AAAI, 2000.

[112] D. M. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles, "Collaborative filtering by personality diagnosis: A hybrid memory-and model-based approach," in *UAI 2000*, pp. 473–480, AUAI, 2000.

[113] H. Polat and W. Du, "SVD-based collaborative filtering with privacy," in *ACM SAC '05*, pp. 791–795, ACM, 2005.

[114] A. Popescul, L. H. Ungar, D. M. Pennock, and S. Lawrence, "Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments," in *UAI 2001*, pp. 437–444, Morgan Kaufmann Publishers Inc., 2001.

[115] G. Potter, "Putting the collaborator back into collaborative filtering," in *KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, pp. 1–4, ACM, 2008.

[116] N. Ramakrishnan, B. Keller, B. Mirza, A. Grama, and G. Karypis, "Privacy risks in recommender systems," *IEEE Internet Computing*, vol. 5, no. 6, pp. 54–63, 2001.

[117] A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl, "Getting to know you: Learning new user preferences in recommender systems," in *ACM IUI '02*, pp. 127–134, ACM, 2002.

[118] J. Reilly, J. Zhang, L. McGinty, P. Pu, and B. Smyth, "Evaluating compound critiquing recommenders: A real-user study," in *ACM EC '07*, pp. 114–123, ACM, ACM ID: 1250929, 2007.

[119] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: an open architecture for collaborative filtering of netnews," in *ACM CSCW '94*, pp. 175–186, ACM, 1994.

[120] P. Resnick and R. Sami, "The influence limiter: Provably manipulation-resistant recommender systems," in *ACM RecSys '07*, pp. 25–32, ACM, 2007.

[121] P. Resnick and R. Sami, "The information cost of manipulation-resistance in recommender systems," in *ACM RecSys '08*, pp. 147–154, ACM, 2008.

[122] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, eds., *Recommender Systems Handbook*. Springer, 2010.

[123] E. Rich, "User modeling via stereotypes," *Cognitive Science*, vol. 3, no. 4, pp. 329–354, October 1979.

[124] C. J. V. Rijsbergen, *Information Retrieval*. Butterworth-Heinemann, 1979.

[125] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted Boltzmann machines for collaborative filtering," in *ACM ICML '07*, pp. 791–798, ACM, 2007.

[126] G. Salton, "The state of retrieval system evaluation," *Information Processing and Management*, vol. 28, no. 4, pp. 441–449, July 1992.

[127] T. D. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network," *Neural Networks*, vol. 2, no. 6, pp. 459–473, 1989.

[128] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Analysis of recommendation algorithms for e-commerce," in *ACM EC '00*, pp. 158–167, ACM, ACM ID: 352887, 2000.

[129] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Incremental SVD-based algorithms for highly scaleable recommender systems," in *ICCIT 2002*, 2002.

[130] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Reidl, "Item-based collaborative filtering recommendation algorithms," in *ACM WWW '01*, pp. 285–295, ACM, 2001.

[131] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl, "Application of dimensionality reduction in recommender system — a case study," in *WebKDD 2000*, 2000.

[132] J. B. Schafer, J. A. Konstan, and J. Riedl, "E-Commerce recommendation applications," *Data Mining and Knowledge Discovery*, vol. 5, no. 1, pp. 115–153, Janaury 2001.

[133] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, "Methods and metrics for cold-start recommendations," in *ACM SIGIR '02*, pp. 253–260, ACM, 2002.

[134] S. Sen, F. M. Harper, A. LaPitz, and J. Riedl, "The quest for quality tags," in *ACM GROUP '07*, pp. 361–370, ACM, 2007.

[135] G. Shani and A. Gunawardana, "Evaluating recommendation systems," in *Recommender Systems Handbook*, (F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, eds.), pp. 257–297, Springer, 2010.

[136] G. Shani, D. Heckerman, and R. I. Brafman, "An MDP-based recommender system," *Journal of Machine Learning Research*, vol. 6, pp. 1265–1295, 2005.

[137] U. Shardanand and P. Maes, "Social information filtering: Algorithms for automating "word of mouth"," in *ACM CHI '95*, pp. 210–217, ACM Press/Addison-Wesley Publishing Co., 1995.

[138] R. Shokri, P. Pedarsani, G. Theodorakopoulos, and J. Hubaux, "Preserving privacy in collaborative filtering through distributed aggregation of offline profiles," in *ACM RecSys '09*, pp. 157–164, ACM, 2009.

[139] B. Smyth, "Case-based recommendation," in *The Adaptive Web*, vol. 4321 of *Lecture Notes in Computer Science*, (P. Brusilovsky, A. Kobsa, and W. Nejdl, eds.), pp. 342–376, Springer, 2007.

[140] X. Su and T. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in Artificial Intelligence*, vol. 2009, p. 19, August 2009.

[141] K. Swearingen and R. Sinha, "Interaction design for recommender systems," in *DIS 2002*, ACM, 2002.

[142] J. A. Swets, "Information retrieval systems," *Science*, vol. 141, no. 3577, pp. 245–250, July 1963.

[143] L. Terveen and D. W. McDonald, "Social matching: A framework and research agenda," *ACM Transactions on Computer-Human Interaction*, vol. 12, no. 3, pp. 401–434, ACM ID: 1096740, September 2005.

[144] N. Tintarev, "Explanations of recommendations," in *ACM RecSys '07*, pp. 203–206, ACM, 2007.

[145] N. Tintarev and J. Masthoff, "Effective explanations of recommendations: User-centered design," in *ACM RecSys '07*, pp. 153–156, ACM, 2007.

[146] R. Torres, S. M. McNee, M. Abel, J. A. Konstan, and J. Riedl, "Enhancing digital libraries with TechLens+," in *ACM/IEEE JCDL '04*, pp. 228–236, ACM, 2004.

[147] M. van Alstyne and E. Brynjolfsson, "Global village or cyber-balkans? Modeling and measuring the integration of electronic communities," *Management Science*, vol. 51, no. 6, pp. 851–868, June 2005.

[148] P. Viappiani and C. Boutilier, "Regret-based optimal recommendation sets in conversational recommender systems," in *ACM RecSys '09*, pp. 101–108, ACM, 2009.

[149] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*. Princeton, NJ: Princeton University Press, 1944.

[150] G. Walsh and J. Golbeck, "Curator: A game with a purpose for collection recommendation," in *ACM CHI '10*, pp. 2079–2082, ACM, ACM ID: 1753643, 2010.

[151] P. M. West, D. Ariely, S. Bellman, E. Bradlow, J. Huber, E. Johnson, B. Kahn, J. Little, and D. Schkade, "Agents to the Rescue?," *Marketing Letters*, vol. 10, no. 3, pp. 285–300, 1999.

[152] M. Xie, L. V. S. Lakshmanan, and P. T. Wood, "Breaking out of the box of recommendations: From items to packages," in *ACM RecSys '10*, pp. 151–158, ACM, ACM ID: 1864739, 2010.

[153] Y. Yang and X. Liu, "A re-examination of text categorization methods," in *ACM SIGIR '99*, pp. 42–49, ACM, 1999.

[154] C. Ziegler and J. Golbeck, "Investigating interactions of trust and interest similarity," *Decision Support Systems*, vol. 43, no. 2, pp. 460–475, March 2007.

[155] C. Ziegler and G. Lausen, "Propagation models for trust and distrust in social networks," *Information Systems Frontiers*, vol. 7, no. 4, pp. 337–358, December 2005.

[156] C. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, "Improving recommendation lists through topic diversification," in *ACM WWW '05*, pp. 22–32, ACM, 2005.

[157] P. Zigoris and Y. Zhang, "Bayesian adaptive user profiling with explicit & implicit feedback," in *ACM CIKM '06*, pp. 397–404, ACM, 2006.