Chapter II
Computing Recommendations with Collaborative Filtering

Neal Lathia
University College London, UK

ABSTRACT

Recommender systems generate personalized content for each of its users, by relying on an assumption reflected in the interaction between people: those who have had similar opinions in the past will continue sharing the same tastes in the future. Collaborative filtering, the dominant algorithm underlying recommender systems, uses a model of its users, contained within profiles, in order to guide what interactions should be allowed, and how these interactions translate first into predicted ratings, and then into recommendations. In this chapter, the authors introduce the various approaches that have been adopted when designing collaborative filtering algorithms, and how they differ from one another in the way they make use of the available user information. They then explore how these systems are evaluated, and highlight a number of problems that prevent recommendations from being suitably computed, before looking at the how current trends in recommender system research are projecting towards future developments.

INTRODUCTION

Recommender systems are experiencing a growing presence on the Internet; they have evolved from being interesting additions of e-commerce web sites into essential components and, in some cases, the core of online businesses. The success of these systems stems from the underlying algorithm, based on collaborative filtering, which re-enacts the way humans exchange recommendations in a way that can be scaled to communities of millions of online users. Users of these systems will thus see personalized, unique, and interest-based recommendations presented to them

computed according to the opinions of the other users in the system, and can actively contribute to other's recommendations by inputting their own ratings.

This chapter introduces recommender systems and the algorithms, based on collaborative filtering, that fuel the success these systems are experiencing in current online applications. There are a number of methods that have been applied when designing filtering algorithms, but they all share a common assumption: the users, and the interactions between them, can be modeled in such a way that it is possible to filter content based on the responses they input.

In particular, the objectives of this chapter can be decomposed into a number of questions:

- Why do we need recommender systems; what problem do they address?
- How are recommendations generated? This question explores collaborative filtering: what it is, how it works, and how different fields of research have led collaborative filtering to be categorized into memory- and model-based approaches.
- How are recommender systems evaluated? In particular, what problems do these systems face, and how does research address these problems? Lastly,
- What are the current future directions of recommender system research?

We explore these questions by considering the participants of a recommender system as members of a community of users. This method highlights the importance of user models within recommender systems, both as a means of reasoning about the underlying operations on the data and building a system that end-users will respond positively to. However, we begin by looking at the motivating problems and history of these systems.

BACKGROUND

As the Internet grows, forever broadening both the range and diversity of information that it makes accessible to its users, a new problem arises: the amount of information available, and the rate at which new information is produced, becomes too great for individuals to sift through it all and find relevant resources. Resources may include, but are not limited to, movies, music, products of e-commerce catalogues, blogs, news articles and documents. Users, unable to dedicate the time to browse all that is available, are thus confronted with the problem of *information overload*, and the sheer abundance of information diminishes users' ability to identify what would be most useful and valuable to each of their needs.

Recommender systems, based on the principles of collaborative filtering, have been developed in response to information overload, by acting as a decision-aiding tool. However, recommender systems break away from merely helping users search for content towards providing interest-based, personalized content without requiring any search query. Recommender systems diverge from traditional information retrieval by building long term models of each user's preferences, and selectively combining different users' opinions in order to provide each user with unique recommendations.

Research into the field of collaborative filtering began in the early 1990s, with the first filtering system, Tapestry, being developed at the Xerox Palo Alto Research Center (Goldberg et al, 1992). This system, recognizing that simply using mailing lists would not ensure that all users interested in an e-mail's content would receive the message, allowed users to annotate e-mail messages so that others could filter them by building complex queries. This was the first system to capture the power of combining human judgments, expressed in message annotations, with automated filtering, in order to benefit all of the system's users. Similar concepts were later applied to Usenet

news by the GroupLens research project, which extended previous work by applying the same principles to the Internet discussion forum, which had become too big for any single user to manage (Konstan et al, 1997). In doing so, they created the first *virtual community of recommenders*, which we will explore further below. The GroupLens project continues contributing to recommender system research, and has also implemented the MovieLens movie recommender system, providing the research community with valuable data of user ratings.

The initial success that recommender systems experienced reflected the surge of web sites dedicated to e-commerce; Schafer et al (2001) review and describe a number of mainstream examples that implement these systems. The cited sites, like Amazon.com and CDNow.com, implement recommenders to build customer loyalty, increase profits, and boost item-cross selling. In fact, it has been reported that 35% of Amazon.com's product sales come from recommendations, recommendations generate 38% more click-through on Google news, and over two thirds of movies rented by online movie-renting site Netflix were recommended (Celma & Lamere, 2007). The same technologies can also be used to address a wide range of different needs. These include ad targeting and one-to-one marketing. However, Schafer et al also describe the relationship between recommender systems and users rating buyers and sellers on sites like eBay.com; in fact, they touch upon the overlap between recommendation and reputation systems. More recently, web sites like Last.fm have reaped the benefits of collecting user-music listening habits, in order to provide customized radio stations and music recommendations to their subscribers. The influence, presence, and importance of the recommender system is not only well established, but also grows over time, as we address the evermore important problem of filtering never ending content.

Before introducing the underlying algorithms of recommender systems, it is useful to define the terms that will be used throughout this chapter.

- **User:** the end-user of the system, or the person we wish to provide with recommendations. This is often referred to as the *active user*; however, in this chapter we differentiate between the current users we are generating recommendations for and the users contributing to the recommendation by referring to the latter users as *recommenders*. The entire set of users is referred to as the *community*.
- **Rating:** The problem of generating recommendations is often described as a problem of *predicting* how much a user will like, or the exact rating that the user will give to, a particular item. Ratings can be explicit or implicit, as detailed in the next section.
- **Profile:** Users in a recommender system can be modeled according to a wide variety of information, but the most important information is the set of ratings that users have provided the system with, which corresponds to each user's profile. These are considered in more depth below.

RATINGS AND USER PROFILES

The focal point of recommender systems is the set of user profiles; by containing a collection of judgments, or ratings, of the available content, this set provides an invaluable source of information that can be used to provide each user with recommendations.

Human judgments, however, can come from two separate sources. These are related to the broader category of *relevance feedback* from the information retrieval community (Ruthven & Lalmas, 2003); a comprehensive review of information retrieval techniques can be found Faloutsos & Oar, 1995. On the one hand, the judgments could be in the form of *explicit* ratings. For example, a user who liked a movie could give it

a 4-star rating, or can give a faulty product a 1-star rating; the judgment is a numeric value that is input directly by the user. On the other hand, judgments can be extracted from the *implicit* behavior of the user. These include time spent reading a web page, number of times a particular song or artist was listened to, or the items viewed when browsing an online catalogue. Measuring these qualities is an attempt to capture taste by measuring how users interact with the content, and thus will often depend on the specific context that the recommender system is operating upon. For example, movie-recommender systems often prefer to let users explicitly rate movies, since it might often be the case that users disliked a particular movie. Music recommender systems, on the other hand, tend to construct user profiles based on listening habits, by collecting meta-data of the songs each user has listened to; these systems favor implicit ratings by assuming that users will only listen to music they like. Implicit ratings can be converted to a numeric value with an appropriate transpose function, and therefore the algorithms we describe below are equally applicable to both types of data. They also both share a common characteristic: the set of available judgments for each user, compared to the total number of items that can be rated, will be very small. This stems from the very nature of the information overload problem, and without it, recommender systems would no longer be needed. The lack of information is often referred to the problem of data *sparsity*, and has a strong effect on the predictive power of any algorithms that base their recommendations on this data. A small example of a set of user profiles, often called a user-rating matrix, for a movie recommender system, is shown in Table 1.

The problem of data sparsity reveals itself in this example; not all users have rated all the content. It also paves the way for the algorithms we describe in the following sections, which aim at predicting ratings for each user. It is important to note, however, that the techniques described here can be equally applied to both user profiles, which contain a vector of content (or item) ratings, and item profiles, which contain a vector of user ratings (Sarwar et al, 2001; Linden et al, 2003). For example, a user-centered approach would refer to "Alice's" profile as containing "Citizen Kane" and "Hannibal," with 4 and 3 star ratings, respectively. An item-centered approach, instead, would consider "The Matrix's" profile as "Bob" and "David," who assigned 5 and 2 stars to the item. Both methods produce comparable results, and differ only in their perspective of the system; one considers the rows of the user-rating matrix, and the other uses the columns. In this chapter, we focus on the user-centered approach. Furthermore, a rating give by user u for item i will be referred to as r_{ui}, and the set of ratings that correspond to user u's profile is R_u.

Although above we have differentiated between the explicit and implicit collection of user preferences, the two methods need not be separate. In fact, Basu et al (2001) discuss how technical papers can be recommended to reviewers by combining information from multiple sources; not limiting the sources of information can improve recommendation by increasing the knowledge we have of each user's profile. However, Herlocker et al (2004) identified that user profiles are created

Table 1.

	The Matrix	Citizen Kane	Hannibal	Snow White	…
Alice		4	3		…
Bob	5		4	1	…
David	2	4		4	…
…	…	…	…	…	…

for different reasons, including self-expression, and helping or influencing others' decisions. Similarly, the tasks that are requested of recommender systems can vary, from finding good items, finding all items, recommending a sequence of items, or as a browsing aide. However, the main goal of recommender systems remains the same: we aim at filtering content in order to provide relevant and useful suggestions to each user of the system. The particular task, or context, will influence the approaches that can be used, which we discuss below. Filters are often classified into one of two categories; *content*-based filters, or *collaborative*-filters.

CONTENT-BASED FILTERS

Content-based recommender system algorithms disregard the *collaborative* component, which we will explore further below, and base their recommendation generative power on matching descriptions of the content in the system to individual user profiles (Pazzani & Billsus, 2007). The key to these recommendations lies in decomposing the content in the system into a number of attributes, which may be based on enumerable, well-defined descriptive variables, such as those found in an explicit taxonomy. The attributes can also be extracted features, such as word frequency in news articles, or user-input tags. The user profile, on the other hand, contains a model of the items that are of interest to that user. These may include a history of purchases, or explicitly defined areas of interest; for example, a user may input the sort of qualities desired when looking for a product (e.g. "price is less than," "album artist is," and so on).

Recommendations can then be generated by applying one of a wide variety of methods to the user model of preferences. These include rule induction methods and decision trees; a comprehensive review can be found in Pazzani & Billsus (2007). However, an interesting consequence of building recommender systems this way is that they can quickly adapt to and change recommendations based on the user's immediate feedback. This leads to the idea of *conversational* recommenders, which allows users to revise the preferences they input by critiquing the obtained results (Viappiani et al, 2007). In doing so, user models themselves are highly dynamic and specific to the current recommendation that is sought, and allow users to understand the effect of their preferences on the recommendations they are given.

Content-based systems, however, are not appropriately or readily applied to the entire range of scenarios where users may benefit from recommendations. On the one hand, these systems require content that favors analysis, and can be described in terms of a number of attributes, which may not always be the case. Eliciting preferences is a valid data collection technique in a limited number of contexts and more suitable for environments where content attributes play a significant role in each user's ultimate decision, such as selecting an apartment, a restaurant, or a laptop computer. In other cases, it may be too much work to impose on the user, and the collaborative filtering alternative is a more appropriate solution.

COLLABORATIVE FILTERING

Unlike content-based systems, collaborative filtering algorithms take a "black-box" approach to content that is being filtered (Herlocker et al, 1999). In other words, they completely disregard any descriptions or attributes of the data, or what the data actually is, in favor of human judgments, and focuses on generating recommendations based on the opinions that have been expressed by a community of users. In doing so, they augment the power of filtering algorithms towards pure quality-based filtering, and have been widely applied to a variety of Internet web sites, such as the ones explored above.

The problem of generating recommendations, and the use of the data that is available to tackle this task, has been approached from a very wide range of perspectives. Each perspective applies different heuristics and methodologies in order to create recommendations. In the following sections, we review the two broadest categories of filters: memory- and model-based collaborative filtering followed by a brief look at other methods and hybrid approaches.

Memory-Based Collaborative Filtering

Memory-based collaborative filtering is often referred to as the dominant method of generating recommendations; its clear structure, paired with the successful results it produces, makes it an easy choice for system developers. It is called memory-based filtering since it relies on the assumption that users who have been historically like-minded in the past will continue sharing their interests in the future (Herlocker et al, 1999). Therefore, recommendations can be produced for a user by generating predicted ratings of unrated content, based on an aggregate of the ratings given by the most similar (or "nearest") users from within the community. This is why the process is often referred to as *k*NN, or *k* nearest-neighbor filtering, and can be decomposed into three stages; neighborhood formation, opinion aggregation, and recommendation.

Neighborhood Formation

This first step aims at finding a unique subset of the community for each user, by identifying others with similar interests to act as recommenders. To do so, every pair of user profiles is compared, in order to measure the degree of similarity $w_{a,b}$ shared between all user pairs *a* and *b*. In general, similarity values range from 1 (perfect similarity) to -1 (perfect dissimilarity), although different measures may only return values on a limited amount of this range. If a pair of users has no profile overlap, there is no means of comparing how similar they are, and thus the similarity is set to 0.

Similarity can be measured in a number of ways, but the main goal of this measure remains that of modeling the potential relationship between users with a numeric value. The simplest means of measuring the strength of this relationship is to count the proportion of co-rated items shared by the pair of users (Charikar 2002):

$$w_{a,b} = \frac{|R_a \cap R_b|}{|R_a \cup R_b|} \quad (1)$$

This similarity measure disregards the values of the ratings input by each user, and instead opts to only consider *what* each user has rated; it is the size of the intersection of the two users' profiles over the size of the union. The underlying assumption is that two users who continuously rate the same items share a common characteristic: their choice to rate those items.

However, the most cited method of measuring similarity is the Pearson Correlation Coefficient, which aims at measuring the degree of linearity that exists on the intersection of the pair of users' profiles (Breese et al, 1998; Herlocker et al, 1999): this is a measure of linearity between two user's profiles.

$$w_{a,b} = \frac{\sum_{i=1}^{N} \left(r_{a,i} - \overline{r_a}\right)\left(r_{b,i} - \overline{r_b}\right)}{\sqrt{\sum_{i=1}^{N} \left(r_{a,i} - \overline{r_a}\right)^2 \sum_{i=1}^{N} \left(r_{b,i} - \overline{r_b}\right)^2}} \quad (2)$$

Each rating above is normalized by subtracting the user's mean rating; this value is the average of all the ratings in the user profile. The Pearson Correlation similarity measure has been subject to a number of improvements. For example, if the intersection between the pair of user's profiles

is very small, the resulting similarity measure is highly unreliable, as it may indicate a very strong relationship between the two users (who, on the other hand, have only co-rated very few items). To address this, Herlocker et al (1999) introduced *significance weighting*: if the number of co-rated items n is less than a threshold value x, the similarity measure is multiplied by n/x. This modification reflects the fact that similarity measures become more reliable as the number of co-rated items increases, and has positive effects on the predictive power of the filtering algorithm. The same researchers also cite the *constrained* Pearson correlation coefficient, which replaces the user means in the above equation with the rating scale midpoint.

There are a number of other ways of measuring similarity that have been applied in the past. These include the Spearman Rank correlation, the Vector Similarity (or cosine angle between the two user profiles), Euclidean and Manhattan distance, and other methods aimed at capturing the proportion of agreement between users, such as the methods explored by Agresti and Winner (1997). Each method differs in the operations it applies in order to derive similarity, and may have a strong effect on the power the algorithm has to generate predicted ratings.

Similarity measures are also often coupled with other heuristics that aim at improving the reliability and power of the derived measures. For example, Yu et al (2001) introduced *variance weighting*; when comparing user profiles, items that have been rated by the community with greater variance receive a higher weight. The aim here is to capture the content that, by being a measurably high point of disagreement amongst community members, is a better descriptor of taste. Measuring similarity, however, remains an open issue; to date, there is little that can be done other than comparing prediction accuracy in order to demonstrate that one similarity measure outperforms another on a particular dataset.

Opinion Aggregation

Once comparisons between the user and the rest of the community of recommenders (regardless of the method applied) are complete, we have a set of recommender weights, and predicted ratings of unrated content can be computed. As above, there are a number of means of computing these predictions. Here we present two (Herlocker et al, 1999; Bell & Koren, 2007):

$$p_{a,i} = \overline{r_a} + \frac{\sum \left(r_{b,i} - \overline{r_b}\right) w_{a,b}}{\sum w_{a,b}} \tag{3}$$

$$p_{a,i} = \frac{\sum \left(r_{b,i} \times w_{a,b}\right)}{\sum w_{a,b}} \tag{4}$$

Both equations share a common characteristic: a predicted rating $p_{a,i}$ of item i for user a is computed as a weighted average of neighbor ratings $r_{b,i}$. The weights $w_{a,b}$ are the similarity measures we found in the first step, and therefore neighbors who are more similar will have greater influence on the prediction. The main difference between the two methods is that Equation 3 subtracts each recommender's mean from the relative rating. The aim of this method is to minimize the differences between different recommender's rating style, by considering how much ratings deviate from each recommender's mean rather than the rating itself.

The natural question to ask at this step is: which recommender ratings are chosen to contribute to the predicted rating? A variety of choices is once again available, and has a direct impact on the performance that can be achieved. In some cases, only the top-k most similar neighbors are allowed to contribute ratings, thus guaranteeing that only the closest ratings create the prediction. However, it is often the case that none of the top-k neighbors have rated the item in question, and thus the prediction *coverage*, or the number of items

that can be successfully predicted, is negatively impacted. A straightforward alternative, therefore, is to consider the top-*k* recommenders who *can* give rating information about the item in question. On the one hand, this method guarantees that all predictions will be made; on the other hand, predictions may now be made according to ratings provided by only modestly-similar users, and may thus be less accurate.

A last alternative is to only select users above a pre-determined similarity threshold. Given that different similarity measures will produce different similarity values, generating predictions this way may also prevent predictions from being covered. All methods, however, share a common decision: what should the threshold value, or value of *k*, be? This question remains unanswered and dependent on the available dataset; however, research in the area tends to publish results for a wide range of values.

Recommendation

Once predicted ratings have been generated for the items, and sorted according to predicted value, the top-*n* items can be proposed to the end user as recommendations. This step completes the process followed by recommender systems, which can now elicit feedback from the user. User profiles will grow, and the recommender system can begin cycling through the above process: re-computing user similarity measures, predicting ratings, and offering recommendations.

It is important to note that the user interface of the system plays a vital role in this last step. The interface does not only determine the ability the system has to present generated recommendations to the end user in a clear, transparent way, but will also have an effect on the response that the user gives to received recommendations. Wu & Huberman (2007) conducted a study investigating the temporal evolution of opinions of products posted on the web. They concluded that if the aggregate rating of an item is visible to users and the cost of expressing opinions for users is low (e.g. one click of a mouse), users will tend to express either neutral ratings or reinforce the view set by previous ratings. On the other hand, if the cost is high (such as requiring users to write a full review), users tended to offer opinions when they felt they could offset the current trend. Changing the visibility of information and the cost imposed on users to express their opinions, both determined by the interface provided to end users, will thus change the rating trend of the content, and the data that feeds into the filtering algorithm.

Up to this point, we have considered the process of generating recommendations strictly from the memory-based, nearest-neighbor approach. However, tackling the problem of information overload has been approached from a wide range of research fields and backgrounds. In the following section we review some of the contributions made by the field of machine learning, often referred to as model-based collaborative filtering.

Model-Based Collaborative Filtering

Model-based approaches to collaborative filtering, stemming from the field of machine learning, aim to apply the broad set of solutions developed by that field of research to the problem of information filtering. A complete introduction to machine learning is beyond the scope of this chapter, although there are many sources available for background reading, such as Alpaydin (2004).

The applicability of machine-learning techniques is founded in our original description of the aim of filtering: we would like to predict how much users will like, or rate, the content they have not rated already, and rank these items in order to provide the top-*n* as recommendations. In other words, collaborative filtering falls between the broader categories of *classification*, or deciding what rating group unrated items belong to, and *regression*, the process of modeling the relationship a variable (such as a user rating) has with other variables (the set of user profiles).

An example that highlights the applicability of these techniques to recommender systems is the use of a p-rank algorithm (Crammer & Singer, 2001). The items that a user has rated, in this case, are considered as a set of training *instances*. Each instance can be described by a vector of features *x*; in our case, the features correspond to the ratings given to the item by the community of users. The goal of the algorithm is to learn a *ranking rule*, or mapping from an instance to the correct rank (or, equivalently, a mapping from a user-item to the correct rating). To do so, the algorithm needs to learn how to weight the individual features, and will attempt to do so by iterating over the training instances. It begins with a vector of weights *w* (set to an initial value), and a set of *b thresholds*, one for each rank possible. Therefore, for example, if a 5-star rating scale is implemented, $b = 5$. At each step, it will make a prediction based on the current set of weights, by multiplying the feature vector *x* with the weight vector *w*. The predicted rank is then computed as the index *r* of the smallest threshold such that $w \times x < b_r$. When the user inputs the actual rating, the algorithm can check to see if it made a mistake, and, if it did, it will update its weights *w* and thresholds *b*. Over time, this algorithm aims to minimize the loss between predicted and actual ranks, by learning how to make accurate predictions using a set of instance features. This algorithm approaches the problem of filtering as an instance of a linear classification problem, and thus its inception is based on perceptron classifiers.

The p-rank algorithm is just one of the solutions proposed by the machine learning community. Other quoted examples include the use of singular value decomposition, neural net classifiers, Bayesian networks, support vector machines, induction rule learning, and latent semantic analysis (Breese et al, 1998; Yu et al, 2004). Each differs in the method applied to learn how to generate recommendations, but they all share a similar high-level solution: they are based on inferring rules and patterns from the available rating data.

Model-based approaches are attractive solutions since, once trained, they compute predicted ratings extremely efficiently. However, they have had limited success, since (the simpler) memory-based approaches have been shown to be just as accurate (Grcar et al, 2005). The two categories of solutions also differ in their interpretation of the users operating within the system. Memory-based methods model all user interactions based on measurable similarity-values, and thus leads to the notion of a community of recommenders. Model-based approaches, instead, train a *separate* model for each user in the system, and are thus characterized by a stronger subjective view of the recommender system's end users.

Hybrid Methods

As we have seen, filtering algorithms have been designed from a number of different backgrounds, leading to the categorization of these algorithms into memory- and model-based groups. Each method provides a number of advantages, and faces a number of weaknesses. Hybrid methods, combining a series of techniques from both groups, aim at achieving the best of both worlds: the advantages of each method stripped of the weaknesses that it faces when operating alone.

For example, Rashid et al (2006) proposed a filtering algorithm suitable for extremely large datasets that combines a clustering algorithm with the nearest-neighbor prediction method. The aim was to cluster similar users together first, in order to overcome the incredibly costly operation of measuring the similarity between all of the community user pairs in the system, and then apply a nearest-neighbor technique to make predictions in order to reap the high accuracy it tends to achieve. Much like the work presented by Li & Kim (2003), clustering methods can be implemented to replace the "neighborhood formation" step of memory-based approach described above. The Yoda system, designed by Shahabi et al (2001), is an example system that performs

similar functions: clustering is implemented to address the scalability issues that arise as the community of users and available items grows. A full overview of the performance of memory- and model-based approaches is available in Breese et al (1998).

Other means of modeling a community of users in order to successfully filter information for each member have been proposed; for example, Cayzer & Aickelin (2002) drew parallels between information filtering and the operation of the human immune system, in order to construct a novel means of filtering. Another example moves into the domain of recommending a *coherent* ordering of songs by applying case-based reasoning (Baccigalupo & Plaza, 2007). Case-based reasoning looks at a set of previous experiences in order to derive information that can be applied to a new problem. Solving the new problem follows similar steps to that described for general machine learning procedures, and entails retrieving the correct sub-set of experiences, applying them to the current problem, and then revising the solution based on any received feedback. This technique was applied successfully to a domain where simply predicting good songs was not enough, but predicting a good sequence of songs was desired.

Up to now, we have had a high-level overview of the multiple approaches applied to recommender systems. However, as we will discuss in the next section, none of the above methods is perfect; moreover, they all share common weaknesses and problems that hinder the generation of useful recommendations.

RECOMMENDER SYSTEMS: PROBLEMS AND EVALUATIONS

The issues that recommender systems face can be grouped into three generic categories: problems arising from within the algorithm, user issues, and system vulnerabilities. A good part of research into collaborative filtering has thus centered on solving these problems, or minimizing the effect that they have on the system and the end user experience. In doing so, the primary metrics used to evaluate these systems emerge, and further questions regarding the suitability of these evaluation methods arise. In this section we will take a look at how experiments on filtering algorithms are conducted, what error measures can be extracted, and the problems that these measures highlight in the operation of recommender systems.

Algorithm

The first set of issues stem from the filtering algorithms applied to generate recommendations. As we have seen above, the common goal of the many algorithms is to *predict* how much users will like different items on offer to them.

Missing Data

Predictions are based on the rating information that has been input by the community of users, and the breadth and number of ratings available is generally much smaller than the full possible set of ratings. In other words, as we have seen, the user-rating matrix is very *sparse*. This characteristic of the data prevents user profiles from being compared to one another, as there will often not be an overlap between the two profiles, and therefore the incomparable pair of users will never be able to contribute to each other's predictions. In other words, the amount of information that can be propagated around the community by means of similarity becomes limited. Solutions to this problem have been proposed; these include dimensionality reduction techniques, such as singular value decomposition (Paterek, 2007), and missing data prediction algorithms (Ma et al, 2007).

Accuracy Error Metrics

Regardless of whether a method is applied to tackle data sparsity, the main task remains that of predicting items users will like. To evaluate how well an algorithm is accomplishing this task, experiments are performed on one of the available user-rating datasets. The dataset is first partitioned into two subsets; the first acts as a *training* set, and will be used to set any values required by the algorithm. These may include, in the case of nearest-neighbor filtering, user-similarity values and determining each user's top-*k* recommenders. In the case of model-based approaches, the training set would determine what instances are available for the algorithm to learn from. The second subset is the *test* set, and remains hidden to the algorithm. An evaluation will feed the training set into the algorithm, and then ask the algorithm to make predictions on all the items in the test set. Predictions can thus be compared to the actual, hidden values held in the test set, and measures of accuracy and coverage can be extracted.

Accuracy metrics aim to evaluate how well the system is making predictions. Available measures of statistical accuracy include the mean absolute error (MAE) and the root mean squared error (RMSE):

$$MAE = \frac{\sum_{N} \left| r_{a,i} - p_{a,i} \right|}{N} \quad (5)$$

$$RMSE = \sqrt{\frac{\sum_{N} \left(r_{a,i} - p_{a,i} \right)^2}{N}} \quad (6)$$

Both of the above measures focus on the difference between a rating of item i by user a, $r_{a,i}$, and the prediction for the same user and item, $p_{a,i}$. In general, both metrics measure the same thing and will thus behave similarly; if an experiment outputs a reduced MAE, the RMSE will also reduce. The difference lies in the degree to which different mistakes are penalized.

The traditional focus on accuracy in recommender research continues to be disputed. On the one hand, the above accuracy metrics focus on the predictions that have been output by the algorithm, regardless of whether the prediction was at all possible or not. Massa & Avesani (2007) have shown that, in terms of prediction accuracy, these systems seem to perform well when pre-defined values are returned. For example, if each prediction simply returned the current user mean (thus not allowing content to be ranked and converted into recommendations), accuracy metrics would still not reflect such poor behavior. McLaughlin & Herlocker (2004) further this argument, by arguing that striving for low mean errors biases recommender systems towards good predictors rather than recommenders. In other words, a error in a prediction affects the mean error the same way, regardless of whether the prediction enabled the entry to qualify as a top-*n* recommendation or not. Furthermore, as shown in the work by Yu et al (2001), many items will have a low rating variance. A natural consequence of this is that an evaluation method that only makes predictions on items in the test set, items that the user has rated, will tend to show good performance. Real systems, that have to provide recommendations based on making predictions on *all* unrated items may have much worse performance. Mean errors will therefore not tend to reflect the end-user experience. Concerns over accuracy-centric research continues; McNee et al (2006) even argued that striving for accuracy is detrimental to recommender system research, and propose that evaluations should revert to user-centric methods.

Accuracy metrics persist, however, due to the need for empirical evaluations of filtering algorithms which can compare the relative performance of different techniques without including the subjective views of a limited (and, more often than not, inaccessible) group of test subjects. Some fixes have been proposed; for example, Lathia et al (2008) limit the measurement of error to predictions that were possible. In this case, it is

imperative to report both accuracy and coverage metrics (as described below) to provide a clear picture of an algorithm's performance; however, this fix does not address the issue of whether a prediction excludes an item from a top-*n* list, and the effect this will have on the end user.

Coverage metrics aim to explore the breadth of predictions that were possible using the given method. Looking back on Equations 3 and 4, it is possible to conceive of a scenario where no neighbor rating information can be found, and thus no prediction can be made. In these cases a default value is returned instead; often this value is the user's rating mean, and these predictions will be labeled uncovered. Coverage metrics compare the proportion of the dataset that is uncovered to the size of the test set, in order to measure the extent that predictions were made possible using the current algorithm and parameters.

Other error measures have been applied when analyzing the accuracy of a filtering algorithm, including receiver-operating characteristic (ROC) sensitivity (Herlocker et al, 1999). This measure draws from work done in Information Retrieval, and aims at measuring how effectively predicted ratings helped a user select high-quality items. Recommendations are therefore reduced to a binary decision: either the user "consumed" the content (i.e. watched the movie, listened to the song, read the article) and rated it, or did not. By comparing the number of false-positives, or items that should have been recommended that were not, and false-negatives, or not recommending an item that should have been, this metric aims at measuring the extent to which the recommender system is helping users making good decisions. However, this method relies on a prediction score threshold that determines whether the item was recommended or not, which is often not translate to the way that users are presented with recommendations.

User-Related Problems

Although poor accuracy and coverage will have a great influence on the user response to the recommender system, the second set of problems is tied much closer to the immediate user experience with the system.

The first of these issues is referred to as the *cold-start* problem; this problem can affect users, items, and new recommender systems equally. On the one hand, users with no historical ratings, which include any new-entrants into the system, will not be able to receive any personalized recommendations. No historical profile implies that no neighbors can be computed, recommender weights will all be zero, and no predictions will be possible. On the other hand, items that have not been rated by any member of the community can not be recommended to any users. This highlights the dependence of filtering algorithms on the altruism of the community of users making use of the recommender system; if users do not rate items, or contribute to their profile then the cycling process of generating recommendations can not be completed.

A number of solutions have been proposed to confront the cold-start problem. In the case of recommender systems based on explicit ratings, the system could require users to rate a number of items as part of the sign-up procedure (Rashid et al, 2002). This method imposes an additional burden on users, but guarantees that there will be a limited amount of profile information to generate the first recommendations. Other researchers proposed to counter the cold-start problem by making inferences from non-profile information which may be included in the sign-up process, such as simple demographic values like age, gender, and location (Nguyen et al, 2007). In this case, a rule-based induction process is applied in order to identify, for each user, a sub-set of the community that will most likely include good recommenders. However, not all recommender

systems require users to input demographic data; this solution is dependent on the details of the sign-up procedure. Other solutions to the user cold-start problem diverge away from similarity, and lean towards the broader notion of trust (Massa & Avesani, 2007). Trust is defined as a user-input measure of how relevant and interesting other recommender's ratings (or reviews) seem to be; it thus has a strong overlap with similarity, but is received from the end-users rather than being computed, and can successfully be propagated over an entire community. However, as described above, the cold-start problem does not only affect users: it can also plague a new system, or prevent new items from being recommended. In this case, Park et al (2006) proposed the use of *filter-bots*, or automated surrogate users who rate items purely based on their content or attributes. Although these bots do not equal the ability a community of users has to find high quality content, they ensure that new items will not be excluded from the recommendation process.

The second set of issues regards the effect that recommendations, when they can be generated, have on the user. On the one hand, there is an issue of *transparency*; in other words, do users understand how their recommendations were generated? This issue will be of primary concern to those developing the user interface with the system, who will aim to present recommendations in a clear, understandable way (Tintarev & Masthoff, 2007). On the other hand, users look to recommender systems for new, interesting, and surprising (or *serendipitous*) information. If a user rates an item (for example, rating an album by The Beatles), loading the user's recommendations with extremely similar items (i.e. all of the other albums by The Beatles) is often not helpful at all; the user has not been pointed towards new information, and is only inundated with recommendations towards content that is probably known already. The question therefore becomes: to what extent do filtering algorithms generate serendipitous recommendations? This is a very difficult characteristic to measure, and remains an open research question.

The last user-issue that we consider here is also tied with the algorithm chosen to generate recommendations. Whether the algorithms are learning over training instances or computing relationships between the user pairs of the community, these algorithms suffer from very high latency. Computing user similarity or feeding data into a learning algorithm is a very expensive operation, often requiring exponential space or time, and can not be continuously updated. Recommender systems therefore tend to perform iterative, regular updates. Users will not be continuously offered new recommendations, and will have to wait for a system update to see their recommendations change. Constant time algorithms have been proposed (Goldberg et al, 2000), but have yet to be widely applied.

System Vulnerabilities

The last set of problems faced by recommender systems are system vulnerabilities; these are the set of problems that are caused by malicious users attempting to game or modify the system. Why would users want to exploit a recommender system? Attempting to modify or control the recommendations that are output by a system aims at harvesting the success of recommender systems for the attacker's selfish purposes. This may be done in order to artificially promote a piece of content, to demote content (perhaps since it competes with the attacker's content), or to target a specific target audience of users. These attacks are aided by the near-anonymity of users in the recommender system community. In some cases, signing up to an e-service that uses recommender system technology only requires an email address. Creating a number of fake accounts is thus not beyond the realms of possibility; furthermore, if each of these fake accounts has an equal contribu-

tion to predicted ratings that honest user profiles do, it becomes possible to direct the output of recommender systems at will.

Malicious users can therefore build a number of fake profiles in order to influence the underlying collaborative filtering algorithm. These attacks have often been referred to as shilling, profile-injection, or Sybil attacks (Mobasher et al, 2007). All of these share the common method of inserting multiple entities into the system in order to change the resulting predicted ratings for the target item, user, or set of users. In other words, they take advantage of the way that users are modeled by the algorithm in order to achieve a desired outcome. The fake profiles that are being inserted can be engineered to be highly correlated to the target user (or item), with the small exception of the rating for the item under attack. This way, nearest-neighbor methods, as described in previous sections, will select the fake profile when generating a predicted rating, and thus will return a prediction that will deviate a lot from the experience the user will have. For example, a movie may be predicted to have a five star rating, while the user would in fact input only two stars; the injected profiles have managed to change the outcome of the recommendations and favor the disliked movie.

Profile-injection attacks are often classified according to the amount of information attackers require in order to successfully construct fake profiles. Lam & Riedl (2004) explored the effectiveness of introducing profiles based on random-valued ratings against profiles based on ratings centered on the global mean of the dataset. Attacks are more effective if they are based on full knowledge of the underlying dataset distribution; however, many filtering datasets share similar distributions, and thus most malicious users will be able to perform more than a simple, naïve attack.

Research in the field of recommender system vulnerabilities can be broken into two categories. On the one hand, system administrators require a means of *identifying* attacks, by being able to recognize when an attack is occurring and which users are malicious profiles. To do so, Chirita et al (2005) propose to identify attackers by measuring characteristics of the injected profiles. A malicious profile can be identified if it shares high similarity with a large subset of users, has a strong effect on the predictive accuracy of the system, and includes ratings that have a strong deviation from the mean agreement amongst the community members. Although this technique can be used to successfully eliminate injected profiles, the difficulty of the problem is also highlighted: what if an honest user's profile is identified as a malicious one?

On the other hand, the vulnerabilities themselves are addressed; how can these attacks be prevented? How can the cost or effect be minimized? General solutions often involve minimizing the number of recommenders that users can interact with, mimicking the social behavior of not trusting unknown people. Model-based approaches have also been shown to be more resilient to manipulation. Resnick & Sami (2007) propose a manipulation-resistant recommender system that protects its user community by applying a reputation score to recommenders. A more comprehensive review of the vulnerabilities of collaborative recommender systems and their robustness to attack can be found in Mobasher et al (2007).

FUTURE TRENDS

Recommender system research is by no means waning. In fact, it has recently been encouraged by the announcement of the Netflix competition, which has contributed to research by releasing one of the largest user-rating datasets available to date[1]. The competition aims at reducing the error in the Netflix movie recommender. It has therefore given way to a surge in research aiming at mere accuracy, and, as we discussed above,

there is much more to recommender systems than solving the prediction problem.

Research to date has widely ignored the temporal characteristic of recommender systems. The only exception lies in the definition of the cold-start problem, which recognizes that new entrants into the system will not receive suitable recommendations until their profile has grown. Exploring the temporal characteristic of these systems would shed light on how they grow, evolve over time and the influence that varying amounts of available rating information has on the accuracy of a system. In other words, a temporal view of the system would shed light on the effect of filtering algorithms applied to a community of users. One method of approaching this problem is to consider a recommender system as a graph. Nodes in this graph correspond to users, and a link between a pair of users is weighted with the similarity shared between the two (Lathia et al, 2008). This paves the way for the techniques described by graph-theory research to be applied to recommender systems.

As we have seen, both the user and community model is essential to collaborative filtering algorithms. These models are based on measurable similarity in order to allow opinion information to be propagated around the community. Understanding how similarity evolves over time, therefore, would also highlight how interactions between recommenders can be designed, and what properties of the system emerge when different filtering algorithms are applied.

Majority of the focus of recommender system research has been context-specific. The datasets available reinforce this focus; the publicly available datasets do not cross between different types of content. However, performing cross-context recommendations remains an open question. Given a user profile of movie preferences, can the user be recommended music successfully? If a user's music has been profiled, can the user be recommended live music events or concerts of interest? This issue is of particular interest to e-commerce portals, which tend to provide a wide range of items and are not limited to a specific type. Many services also only profile a user in a given context, and thus users tend to build multiple profiles over a wide range of locations. Finding means of porting profiles from one place to another, and successfully using them for cross-contextual recommendations has yet to be explored.

Recommender systems also hold the potential to be applied in non-centralized domains: including peer to peer file sharing networks and mobile telephones. There has been a wide range of work done addressing peer to peer network recommendations (Ziegler, 2005), but little work addressing how collaborative filtering would operate in a mobile environment. Mobile collaborative filtering would allow users to benefit as they have when using online services. It would allow them to, for example, share content when on the move, and receiving recommendations relating to their immediate surroundings. Porting collaborative filtering to distributed environments brings to light a new set of obstacles. How will recommendations be computed? Where will profiles be stored, and who will they be shared with? Data *privacy* and *security* gain renewed importance (Lathia et al, 2007).

CONCLUSION

In this chapter, we have introduced the underlying algorithms of recommender systems, based on collaborative filtering. Recommender systems were conceived in response to information overload, a natural consequence of the ever-expanding breadth of online content; it has become impossible to sift through or browse online content without recommendations. Collaborative filtering automates the process of generating recommendations by building on the common assumption of like-mindedness. In other words, people who have displayed a degree of similarity in the past

will continue sharing the same tastes in the future. The model of users held by these systems therefore focuses on the set of preferences that each individual has expressed, and interactions between users can be determined according to values derived by operating on the information available in user's profiles.

The approaches themselves, however, originate from a wide variety of backgrounds, and thus have been separated into content-based methods, which infer recommendations from item attributes, model-based solutions, which draw on the success of machine learning techniques, and the dominant memory-based, nearest neighbor technique. Nearest neighbor algorithms follow a three-stage process: finding a set of recommenders for each user, based on a pre-defined measure of similarity, computing predicted ratings based on the input of these recommenders, and serving recommendations to the user, hoping that they will be accurate and useful suggestions. The choice of what method to implement relies on a fine balance between accuracy, performance, and is also dependent on specific context that recommendations need to be made for. Each method has its own strengths and weaknesses, and hybrid methods attempt to reap the best of both worlds by combining a variety of methods.

The most general problems faced by recommender systems remain the same, regardless of the approach used to build the filtering algorithm. These problems were grouped into three categories: problems originating from the algorithm, including accuracy, coverage, and whether they actually help the user's decision making process, problems centered on the users, including the cold-start problem and displaying serendipitous, transparent recommendations, and lastly, system-wide vulnerabilities and their susceptibility to attack. However, the exciting on-going research promises to not only solve, but clarify the effects of these algorithms on end-users and boost their potential to help users in a wide range of contexts.

REFERENCES

Agresti, A., & Winner, L. (1997). Evaluating Agreement and Disagreement Among Movie Reviewers. *Chance*, *10*, 10-14.

Alpaydin, E. (2004). *Introduction to Machine Learning. Massachusetts*, USA. MIT Press.

Basu, C., Hirsh, H., & Cohen, W. (2001). Technical Paper Recommendation: A Study in Combining Multiple Information Sources. *Journal of Artificial Intelligence Research, 14*, 213-252.

Baccigalupo, C., & Plaza, E. (2007). A Case-Based Song Scheduler For Group Customized Radio. In *Proceedings of the International Conference on Case Based Reasoning (ICCBR)*. Belfast, Ireland: Springer.

Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering (Tech Rep. No. MSR-TR-98-12). Redmond, WA: Microsoft Research.

Borchers A., Herlocker J., Konstan J., & Riedl J. (1998). Ganging up on Information Overload. *IEEE Computer*, *31*, 106-108.

Cayzer, S., & Aickelin, U. (2002). A Recommender System based on the Immune Network. In *Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002)*, Honolulu, USA: IEEE.

Celma O., & Lamere, P. (2007, September). Music Recommendation Tutorial. *Presented at the 8th International Conference on Music Information Retrieval*, Vienna, Austria.

Charikar, M. (2002). Similarity Estimation Techniques From Rounding Algorithms. In *Annual ACM Symposium on Theory of Computing*, Montreal, Canada: ACM Press.

Chirita, P-A., Nejdl, W., & Zamfir, C. Preventing Shilling Attacks in Online Recommender Systems. In *Proceedings of the 7th Annual ACM*

International Workshop on Web Information and Data Management. Bremen, Germany: ACM Press.

Faloutsos, C., & Oard, D. (1995). *A Survey of Information Retrieval and Filtering Methods* (Tech. Rep. No. CS-TR-3514). Maryland, USA: University of Maryland, Department of Computer Science.

Goldberg, D., Nichols, D., Oki, B. M., & Terry, D. (1992). Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM*, *35*, 61-70. ACM Press.

Goldberg, K., Roeder T., Gupta, D., & Perkins, C. (2000). *Eigentaste: A Constant Time Collaborative Filtering* Algorithm (Tech Rep. No. UCB/ERL M00/41). Berkeley, California: University of California, EECS Department.

Grcar, M., Fortuna, B., & Mladenic, D. (2005, August). KNN versus SVM in the Collaborative Filtering Framework. In *Workshop on Knowledge Discovery on the Web.*

Herlocker, J. L., Konstan, J. A., Borchers, A., & Riedl, J. (1999). An Algorithmic Framework for Performing Collaborative Filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Berkley, CA: ACM Press.

Herlocker J., Konstan J., Terveen, L., & Riedl, J. (2004). Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems, 22*, 5-53.

Konstan, J., Miller, B., Maltz, D., Herlocker, J., Gordon, L., & Riedl, J. (1997) GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM, 40*, 77-87. ACM Press.

Lam, S. K., & Riedl, J. (2004). Shilling recommender systems for fun and profit. In *Proceedings of the 13th international conference on World Wide Web*, New York, NY, USA: ACM Press.

Lathia, N., Hailes, S., & Capra, L (2007). Private Distributed Collaborative Filtering Using Estimated Concordance Measures. In *Proceedings of the 2007ACM Conference on Recommender Systems (RecSys)*. Minneapolis, USA: ACM Press.

Lathia, N., Hailes, S., & Capra, L. (2008). The Effect of Correlation Coefficients on Communities of Recommenders. In *23rd Annual ACM Symposium on Applied Computing, Trust, Recommendations, Evidence and other Collaboration Know-how (TRECK) Track*. Fortaleza, Ceara, Brazil: ACM Press.

Li, Q., & Kim, B. M. (2003). Clustering Approach for Hybrid Recommender System. In *Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence*. Beijing, China: IEEE Press.

Linden, G., Smith, B., & York, J. (2003). Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEE Internet Computing*, *7*, 76-80.

Ma, H., King, I., Lyu, M. R. (2007). Effective Missing Data Prediction for Collaborative Filtering. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Amsterdam, Holland: ACM Press.

Massa, P., & Avesani, P. (2007). Trust-aware Recommender Systems. In *Proceedings of the 2007ACM Conference on Recommender Systems (RecSys)*. Minneapolis, USA: ACM Press.

McLaughlin, M. R., & Herlocker, J. L. (2004). A Collaborative Filtering Algorithm and Evaluation Metric that Accurately Model the User Experience. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Sheffield, United Kingdom: ACM Press.

McNee, S. M., Riedl, J., & Konstan, J. A. (2006, April). Being Accurate is Not Enough: How Accu-

racy Metrics have hurt Recommender Systems. In *Extended Abstracts of the 2006 ACM Conference on Human Factors in Computing Systems (CHI 2006)*. Montreal, Canada: ACM Press.

Mobasher, B., Burke, R., Bhaumik, R., & Williams, C. (2007). Towards Trustworthy Recommender Systems: An Analysis of Attack Models and Algorithm Robustness. *Transations on Internet Technology (TOIT)*. 7, 4.

Nguyen, A., Denos, N., & Berrut, C. (2007). Improving new user recommendations

with rule-based induction on cold user data. In *Proceedings of the 2007ACM Conference on Recommender Systems (RecSys)*. Minneapolis, USA: ACM Press.

Park, S., Pennock, D., Madani, O., Good, N., & DeCoste, D. (2006). Naïve filterbots for Robust Cold-start Recommendations. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*. Philadelphia, USA: ACM Press.

Paterek, A. (2007). Improving Regularized Singular Value Decomposition For Collaborative Filtering. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*. Philadelphia, USA: ACM Press.

Pazzani, M. J., & Billsus, D. (2007) Content-Based Recommendation Systems. *The Adaptive Web*, *4321*, 325-341.

Rashid, A. M., Albert, I., Cosley, D., Lam, S. K., McNee, S. M., Konstan, J. A., & Riedl, J. (2002). Getting to Know You: Learning New User Preferences in Recommender Systems. In *International Conference on Intelligent User Interfaces (IUI 2002)*. Miami, Florida: ACM Press.

Rashid, A. M., Lam, S. K., Karypis G., & Riedl, J. (2006, August). ClustKNN: A Highly Scalable Hybrid Model- & Memory-Based CF Algorithm. In *The 12th ACM Conference on Knowledge Discovery and Data Mining* Philadelphia, Pennsylvania, USA: ACM Press.

Resnick, P., & Sami, R. The Influence Limiter: Provably Manipulation Resistant Recommender Systems. In *Proceedings of the 2007ACM Conference on Recommender Systems (RecSys)*. Minneapolis, USA: ACM Press.

Ruthven, I., & Lalmas, M. (2003). A Survey on the Use of Relevance Feedback for Information Access Systems. *The Knowledge Engineering Review*, *18*, 95-145.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International World Wide Web Conference (WWW10),* Hong Kong, China: ACM Press.

Sarwar, B., Konstan, J., Borchers, A., Herlocker, J., Miller, B., & Riedl, J. (1998). Using Filtering Agents to Improve Prediction Quality in the GroupLens Research Collaborative Filtering System. *Proceedings of the 1998 Conference on Computer Supported Cooperative Work*. New Orleans, USA: ACM Press.

Schafer, J., Konstan, J., & Riedl, J. (2001) E-Commerce Recommendation Applications. In *Data Mining and Knowledge Discovery, 5*(1), 115-153.

Shahabi, C., Banaei-Kashani, F., Chen Y.-S., & McLeod, D. (2001). Yoda: An Accurate and ScalableWeb-based Recommendation System. In *Proceedings of Sixth International*

Conference on Cooperative Information Systems. Trento, Italy: Springer**.**

Tintarev, N., & Masthoff, J. (2007). Effective Explanantions of Recommendations: User-Centered Design. In *Proceedings of the 2007 ACM Conference on Recommender Systems (RecSys).* Minneapolis, USA: ACM Press.

Viappiani, P., Pu, P., & Faltings, B. (2007). Conversational Recommenders with Adaptive Suggestions. In *Proceedings of Recommender Systems (RecSys)*. Minneapolis, USA: ACM Press

Yu, K., Schwaighofer, A., Tresp, V., Xu, X., & Kriegel, H. (2004). Probabilistic memory-based collaborative filtering. *IEEE Transactions on Knowledge and Data Engineering, 16*, 56–69.

Yu, K., Wen, Z., Xu, X., & Ester, M. (2001). Feature Weighting and Instance Selection for Collaborative Filtering. In *Proceedings of the 12th International Workshop on Database and Expert Systems Applications*. Munich, Germany: IEEE Press.

Wu, F., & Huberman, B. A. (2007). *Public Discourse in the Web Does Not Exhibit Group Polarization* (Technical Report). Palo Alto, CA: HP Labs Research.

Ziegler, C. (2005). *Towards Decentralised Recommender Systems*. (PhD Thesis), Freiburg, Germany: Freiburg University, Department of Computer Science.

ENDNOTE

[1] http://www.netflixprize.com