CCS360

RECOMMENDER SYSTEMS

CCS360 RECOMMENDER SYSTEMS

UNIT 1 INTRODUCTION

INTRODUCTION

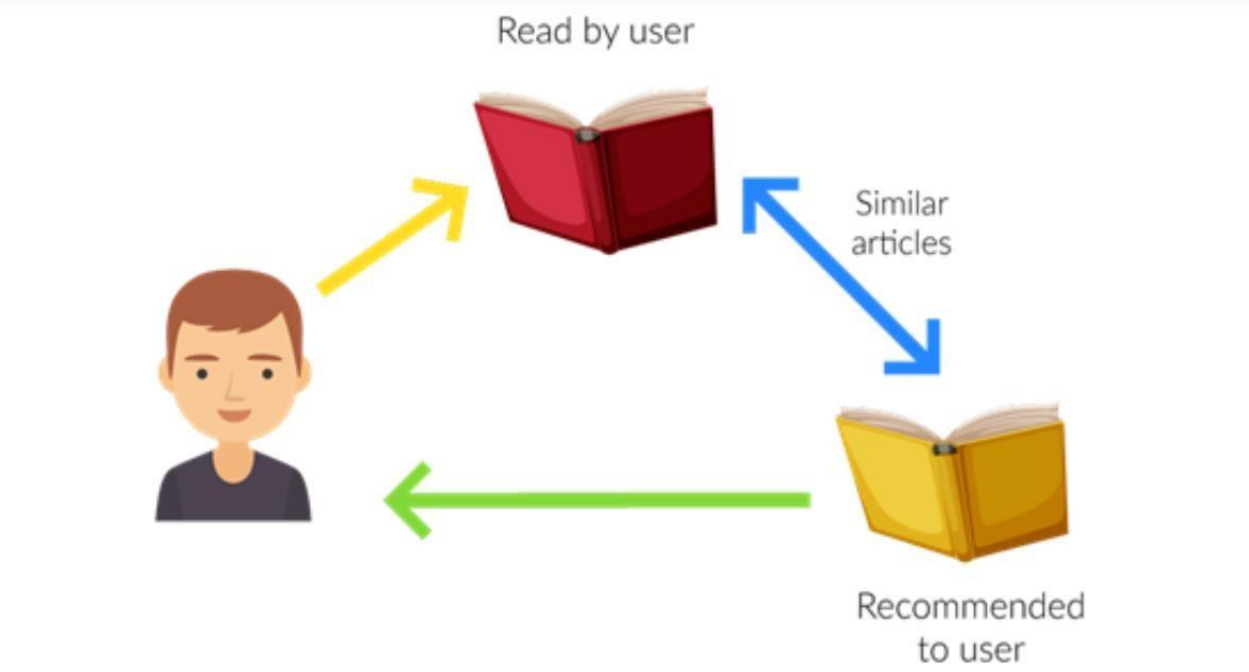
The increasing importance of the Web as a medium for electronic and business transactions has served as a driving force for the development of recommender systems technology. An important catalyst in this regard is the ease with which the Web enables users to provide feedback about their likes or dislikes. For example, consider a scenario of a content provider such as Netflix. In such cases, users are

as an endorsement for that item. Such forms of feedback are commonly used by online merchants such as Amazon.com, and the collection of this type of data is completely effortless in terms of the work required of a customer. The basic idea of recommender systems is to utilize these various sources of data to infer customer interests. The entity to which the recommendation is provided is referred to as the user, and the

able to easily provide feedback with a simple click of a mouse. A typical methodology to provide feedback is in the form of ratings, in which users select numerical values from a specific evaluation system (e.g., five-star rating system) that specify their likes and dislikes of various items.

Definition for Recommender Systems

A recommendation system is a subclass of Information filtering Systems that seeks to predict the rating or the preference a user might give to an item. In simple words, it is an algorithm that suggests relevant items to users.



Other forms of feedback are not quite as explicit but are even easier to collect in the Web-centric paradigm. For example, the simple act of a user buying or browsing an item may be viewed

product being recommended is also referred to as an item. Therefore, recommendation analysis is often based on the previous interaction between users and items, because past interests and proclivities are often good indicators of future choices.

The basic principle of recommendations is that significant dependencies exist between user and itemcentric activity. For example, a user who is interested in a historical documentary is more likely to be interested in another historical documentary or an educational program, rather than in an action movie. In many cases, various categories of items may show significant correlations, which can be leveraged to make more accurate recommendations. Alternatively, the dependencies may be present at the finer granularity of individual items rather than categories. These dependencies can be learned in a data-driven manner from the ratings matrix, and the resulting model is used to make predictions for target users. The larger the number of rated items

that are available for a user, the easier it is to make robust predictions about the future behavior of the user. Many different learning models can be used to accomplish this task. For example, the collective buying or rating behavior of various users can be leveraged to create cohorts of similar users that are interested in similar products. The interests and actions of these cohorts can be leveraged to make recommendations to individual members of these cohorts.

The aforementioned description is based on a very simple family of recommendation algorithms, referred to as neighborhood models. This family belongs to a broader class of models, referred to as collaborative filtering. The term "collaborative filtering" refers to the use of ratings from multiple users in a collaborative way to predict missing ratings. In practice, recommender systems can be more complex and data-rich, with a wide variety of auxiliary data types. For example, in content-based recommender systems, the content plays a primary role in the recommendation process, in which the ratings of users and the attribute descriptions of items are leveraged in order to make predictions. The basic idea is that user interests can be modeled on the basis of properties (or attributes) of the items they have rated or accessed in the past. A different framework is that of knowledge-based systems, in which users interactively specify their interests, and the user specification is combined with domain knowledge to provide recommendations. In advanced models, contextual data, such as temporal information, external knowledge, location information, social information, or network information, may be used.

Goals of Recommender Systems

Prediction version of problem:

The first approach is to predict the rating value for a user-item combination. It is assumed that training data is available, indicating user preferences for items. For m users and n items, this corresponds to an incomplete m × n matrix, where the specified (or observed) values are used for training. The missing (or unobserved) values are predicted using this training model. This problem is also referred to as the matrix completion problem because we have an incompletely specified matrix of values, and the remaining values are predicted by the learning algorithm.

Ranking version of problem:

In practice, it is not necessary to predict the ratings of users for specific items in order to make recommendations to users. Rather, a merchant may wish to recommend the top-k items for a particular user, or determine the top-k users to target for a particular item. The determination of the top-k items is more common than the determination of top-k users, although the methods in the two cases are exactly analogous. Throughout this book, we will discuss only the determination of the top-k items, because it is the more common setting. This problem is also referred to as the top-k recommendation problem, and it is the ranking formulation of the recommendation problem.

Increasing product sales is the primary goal of a recommender system. Recommender systems are, after all, utilized by merchants to increase their profit. By recommending carefully selected items to users, recommender systems bring relevant items to the attention of users. This increases the sales volume and profits for the merchant. Although the primary goal of a recommendation system is to increase revenue for the merchant, this is often achieved in ways that are less obvious than might seem at first sight. In order to

achieve the broader business-centric goal of increasing revenue, the common operational and technical goals of recommender systems are as follows:

Relevance:

The most obvious operational goal of a recommender system is to recommend items that are relevant to the user at hand. Users are more likely to consume items they find interesting. Although relevance is the primary operational goal of a recommender system, it is not sufficient in isolation. Therefore, we discuss several secondary goals below, which are not quite as important as relevance but are nevertheless important enough to have a significant impact.

Novelty:

Recommender systems are truly helpful when the recommended item is something that the user has not seen in the past. For example, popular movies of a preferred genre would rarely be novel to the user. Repeated recommendation of popular items can also lead to reduction in sales diversity

Serendipity:

A related notion is that of serendipity wherein the items recommended are somewhat unexpected, and therefore there is a modest element of lucky discovery, as opposed to obvious recommendations. Serendipity is different from novelty in that the recommendations are truly surprising to the user, rather than simply something they did not know about before. It may often be the case that a particular user may only be consuming items of a specific type, although a latent interest in items of other types may exist which the user might themselves find surprising. Unlike novelty, serendipitous methods focus on discovering such recommendations. For example, if a new Indian restaurant opens in a neighborhood, then the recommendation of that restaurant to a user who normally eats Indian food is novel but not necessarily serendipitous. On the other hand, when the same user is recommended Ethiopian food, and it was unknown to the user that such food might appeal to her, then the recommendation is serendipitous. Serendipity has the beneficial side effect of increasing sales diversity or beginning a new trend of interest in the user. Increasing serendipity often has long-term and strategic benefits to the merchant because of the possibility of discovering entirely new areas of interest. On the other hand, algorithms that provide serendipitous recommendations often tend to recommend irrelevant items. In many cases, the longer term and strategic benefits of serendipitous methods outweigh these short-term disadvantages.

Increasing recommendation diversity:

Recommender systems typically suggest a list of top-k items. When all these recommended items are very similar, it increases the risk that the user might not like any of these items. On the other hand, when the recommended list contains items of different types, there is a greater chance that the user might like at least one of these items. Diversity has the benefit of ensuring that the user does not get bored by repeated recommendation of similar items

GroupLens Recommender System

GroupLens was a pioneering recommender system, which was built as a research prototype for recommendation of Usenet news. The system collected ratings from Usenet readers and used them to predict whether or not other readers would like an article before they read it Some of the earliest automated collaborative filtering algorithms were developed in the GroupLens1 setting. The general ideas developed by this group were also extended to other product settings such as books and movies. The corresponding recommender systems were referred to as BookLens and MovieLens, respectively. Aside from its pioneering contributions to collaborative filtering research, the GroupLens research team was notable for releasing several data sets during the early years of this field, when data sets were not easily available for benchmarking. Prominent examples include three data sets [688] from the MovieLens recommender system. These data sets are of successively increasing size, and they contain 105, 106, and 107 ratings, respectively

Amazon.com Recommender System

Amazon.com was also one of the pioneers in recommender systems, especially in the commercial setting. During the early years, it was one of the few retailers that had the foresight to realize the usefulness of this technology. Originally founded as a book e-retailer, the business expanded to virtually all forms of products. Consequently, Amazon.com now sells virtually all categories of products such as books, CDs, software, electronics, and so on. The recommendations in Amazon.com are provided on the basis of explicitly provided ratings, buying behavior, and browsing behavior. The ratings in Amazon.com are specified on a 5-point scale, with lowest rating being 1-star, and the highest rating being 5-star. The customer-specific buying and browsing data can be easily collected when users are logged in with an account authentication mechanism supported by Amazon. Recommendations are also provided to users on the main Web page of the site, whenever they log into their accounts. In many cases, explanations for recommendations are provided. For example, the relationship of a recommended item to previously purchased items may be included in the recommender system interface. The purchase or browsing behavior of a user can be viewed as a type of implicit rating, as opposed to an explicit rating, which is specified by the user. Many commercial systems allow the flexibility of providing recommendations both on the basis of explicit and implicit feedback. In fact, several models have been designed (cf. section 3.6.4.6 of Chapter 3) to jointly account for explicit and implicit feedback in the recommendation process. Some of the algorithms used by early versions of the Amazon.com recommender system are discussed

Netflix Movie Recommender System

Netflix was founded as a mail-order digital video disc (DVD) rental company [of movies and television shows, which was eventually expanded to streaming delivery. At the present time, the primary business of Netflix is that of providing streaming delivery of movies and television shows on a subscription basis. Netflix provides users the ability to rate the movies and television shows on a 5-point scale. Furthermore, the user actions in terms of watching various items are also stored by Netflix. These ratings and actions are then used by Netflix to make recommendations. Netflix does an excellent job of providing explanations for the recommended items. It explicitly provides examples of recommendations based on specific items that were watched by the user. Such information provides the user with additional information to decide whether or not to watch a specific movie. Presenting meaningful explanations is important to provide the user with an understanding of why they might find a particular movie interesting. This approach also makes it more likely for the user to act on the recommendation and truly improves the user experience. This type of interesting approach can also help improve customer loyalty and retention.

Facebook Friend Recommendations

Social networking sites often recommend potential friends to users in order to increase the number of social connections at the site. Facebook is one such example of a social networking Web site. This kind of recommendation has slightly different goals than a product recommendation. While a product recommendation directly increases the profit of the merchant by facilitating product sales, an increase in the number of social connections improves the experience of a user at a social network. This, in turn, encourages the growth of the social network. Social networks are heavily dependent on the growth of the network to increase their advertising revenues. Therefore, the recommendation of potential friends (or links) enables better growth and connectivity of the network. This problem is also referred to as link prediction in the field of social network analysis. Such forms of recommendations are based on structural relationships rather than ratings data. Therefore, the nature of the underlying algorithms is completely different

Collaborative Filtering Models

Collaborative filtering models use the collaborative power of the ratings provided by multiple users to make recommendations. The main challenge in designing collaborative filtering methods is that the underlying ratings matrices are sparse. Consider an example of a movie application in which users specify ratings indicating their like or dislike of specific movies. Most users would have viewed only a small fraction of the large universe of available movies. As a result, most of the ratings are unspecified. The specified ratings are also referred to as observed ratings. Throughout this book, the terms

"specified" and "observed" will be used in an interchangeable way. The unspecified ratings will be referred to as "unobserved" or "missing."

The basic idea of collaborative filtering methods is that these unspecified ratings can be imputed because the observed ratings are often highly correlated across various users and items. For example, consider two users named Alice and Bob, who have very similar tastes. If the ratings, which both have specified, are very similar, then their similarity can be identified by the underlying algorithm. In such cases, it is very likely that the ratings in which only one of them has specified a value, are also likely to be similar. This similarity can be used to make inferences about incompletely specified values. Most of the models for collaborative filtering focus on leveraging either inter-item correlations or interuser correlations for the prediction process. Some models use both types of correlations. Furthermore, some models use carefully designed optimization techniques to create a training model in much the same way a classifier creates a training model from the labeled data. This model is then used to impute the missing values in the matrix, in the same way that a classifier imputes the missing test labels. There are two types of methods that are commonly used in collaborative filtering, which are referred to as memorybased methods and model-based methods

1. **Memory-based methods:** Memory-based methods are also referred to as neighborhood based collaborative filtering algorithms. These were among the earliest collaborative filtering algorithms, in which the ratings of user-item combinations are predicted on the basis of their neighborhoods. These neighborhoods can be defined in one of two ways:
   - **User-based collaborative filtering:** In this case, the ratings provided by like-minded users of a target user A are used in order to make the recommendations for A. Thus, the basic idea is to determine users, who are similar to the target user A, and recommend ratings for the unobserved

ratings of A by computing weighted averages of the ratings of this peer group. Therefore, if Alice and Bob have rated movies in a similar way in the past, then one can use Alice's observed ratings on the movie Terminator to predict Bob's unobserved ratings on this movie. In general, the k most similar users to Bob can be used to make rating predictions for Bob. Similarity functions are computed between the rows of the ratings matrix to discover similar users.

- **Item-based collaborative filtering:** In order to make the rating predictions for target item B by user A, the first step is to determine a set S of items that are most similar to target item B. The ratings in item set S, which are specified by A, are used to predict whether the user A will like item B. Therefore, Bob's ratings on similar science fiction movies like Alien and Predator can be used to predict his rating on Terminator. Similarity functions are computed between the columns of the ratings matrix to discover similar items. The advantages of memory-based techniques are that they are simple to implement and the resulting recommendations are often easy to explain. On the other hand, memory-based algorithms do not work very well with sparse ratings matrices. For example, it might be difficult to find sufficiently similar users to Bob, who have rated Gladiator. In such cases, it is difficult to robustly predict Bob's rating of Gladiator. In other words, such methods might lack full coverage of rating predictions. Nevertheless, the lack of coverage is often not an issue, when only the top-k items are required. Memory-based methods are discussed in detail in

2. **Model-based methods:** In model-based methods, machine learning and data mining methods are used in the context of predictive models. In cases where the model is parameterized, the parameters of this model are learned within the context of an optimization framework. Some examples of such model-based methods include decision trees, rule-based models, Bayesian methods and latent factor models. Many of these methods, such as latent factor models, have a high level of coverage even for sparse ratings matrices. Model-based collaborative filtering algorithms are discussed Even though memory-based collaborative filtering algorithms are valued for their simplicity, they tend to be heuristic in nature, and they do not work well in all settings. However, the distinction between memory-based and model-based methods is somewhat artificial,

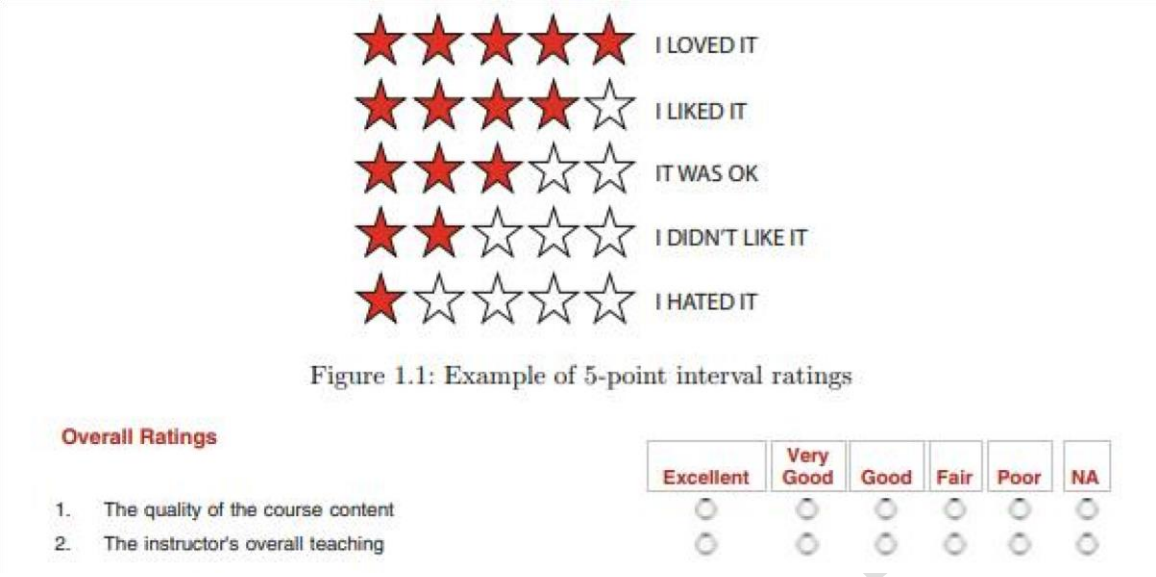


Figure 1.1: Example of 5-point interval ratings

Types of Ratings

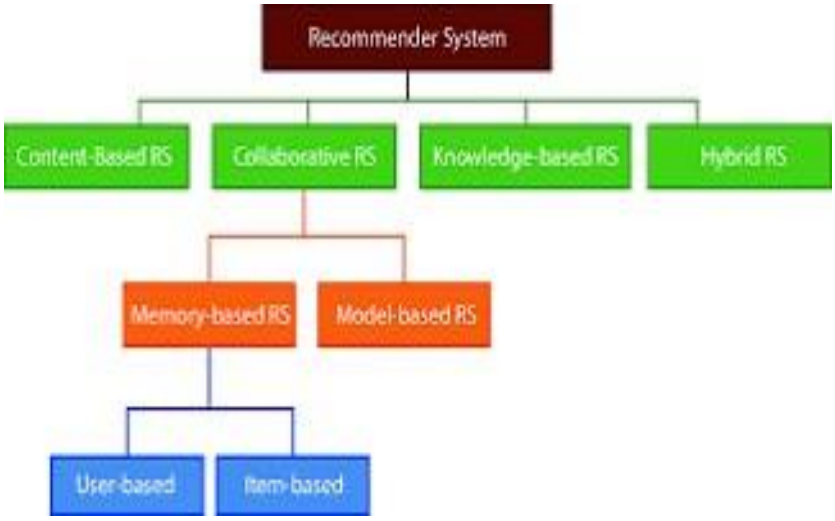
The design of recommendation algorithms is influenced by the system used for tracking ratings. The ratings are often specified on a scale that indicates the specific level of like or dislike of the item at hand. It is possible for ratings to be continuous values, such as in the case of the Jester joke recommendation engine in which the ratings can take on any value between -10 and 10. This is, however, relatively rare. Usually, the ratings are interval-based, where a discrete set of ordered numbers are used to quantify like or dislike. Such ratings are referred to as interval-based ratings. For example, a 5-point rating scale might be drawn from the set $\{-2, -1, 0, 1, 2\}$, in which a rating of -2 indicates an extreme dislike, and a rating of 2 indicates a strong affinity to the item. Other systems might draw the ratings from the set $\{1, 2, 3, 4, 5\}$.

The number of possible ratings might vary with the system at hand. The use of 5 -point, 7-point, and 10-point ratings is particularly common. The 5-star ratings system, illustrated in Figure 1.1, is an example of interval ratings. Along each of the possible ratings, we have indicated a semantic interpretation of the user's level of interest. This interpretation might vary slightly across different merchants, such as Amazon or Netflix. For example, Netflix uses a 5-star ratings system in which the 4star point corresponds to "really liked it," and the central 3-star point corresponds to "liked it." Therefore, there are three favorable ratings and two unfavorable ratings in Netflix, which leads to an unbalanced rating scale.

BASIC MODELS OF RECOMMENDER SYSTEMS

There may be an even number of possible ratings, and the neutral rating might be missing. This approach is referred to as a forced choice rating system. One can also use ordered categorical values such as {Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree} in order to achieve the same goals. In general, such ratings are referred to as ordinal ratings, and the term is derived from the concept of ordinal attributes. An example of ordinal ratings, used in Stanford University course evaluation forms, is illustrated in Figure 1.2. In binary ratings, the user may represent only a like or

dislike for the item and nothing else. For example, the ratings may be 0, 1, or unspecified values. The unspecified values need to be predicted to 0 -1 values. A special case of ratings is that of unary ratings, in which there is a mechanism for a user to specify a liking for a n item but no mechanism to specify a dislike. Unary ratings are particularly common, especially in the case of implicit feedback data sets In these cases, customer preferences are derived from their activities rather than their explicitly specified ratings. For example, the buying behavior of a customer can be converted to unary ratings. When a customer buys an item, it can be viewed as a preference for the item. However, the act of not buying an item from a large universe of possibilities does not always indicate a dislike. Similarly, many social networks, such as Facebook, use "like" buttons, which provide the ability to express liking for an item. However, there is no mechanism to specify dislike for an item. The implicit feedback setting can be viewed as the matrix completion analog of the positive-unlabeled (PU) learning problem in data classification



Relationship with Missing Value Analysis

Collaborative filtering models are closely related to missing value analysis. The traditional literature on missing value analysis studies the problem of imputation of entries in an incompletely specified data matrix. Collaborative filtering can be viewed as a (difficult)special case of this problem in which the underlying data matrix is very large and sparse. Many of these methods can also be used for recommender systems, although some of them might require specialized adaptations for very large and sparse matrices. In fact,some of the recent classes of models for recommender systems, such as latent factor models

Content-Based Recommender Systems

In content-based recommender systems, the descriptive attributes of items are used to make recommendations. The term "content" refers to these descriptions. In content-based methods, the ratings and buying behavior of users are combined with the content information available in the items. For example,

consider a situation where John has rated the movie Terminator highly, but we do not have access to the ratings of other users. Therefore, collaborative filtering methods are ruled out. However, the item description of Terminator contains similar genre keywords as other science fiction movies, such as Alien and Predator. In such cases, these movies can be recommended to John. In content-based methods, the item descriptions, which are labeled with ratings, are used as training data to create a user-specific classification or regression modeling problem. For each user, the training documents correspond to the descriptions of the items she has bought or rated. The class (or dependent) variable corresponds to the specified ratings or buying behavior. These training documents are used to create a classification or regression model, which is specific to the user at hand (or active user). This user-specific model is used to predict whether the corresponding individual will like an item for which her rating or buying behavior is unknown.

Content-based methods have some advantages in making recommendations for new items, when sufficient rating data are not available for that item. This is because other items with similar attributes might have been rated by the active user. Therefore, the supervised model will be able to leverage these ratings in conjunction with the item attributes to make recommendations even when there is no history of ratings for that item.

Content-based methods do have several disadvantages as well:

1. In many cases, content-based methods provide obvious recommendations because of the use of keywords or content. For example, if a user has never consumed an item with a particular set of keywords, such an item has no chance of being recommended. This is because the constructed model is specific to the user at hand, and the community knowledge from similar users is not leveraged. This phenomenon tends to reduce the diversity of the recommended items, which is undesirable.
2. Even though content-based methods are effective at providing recommendations for new items, they are not effective at providing recommendations for new users. This is because the training model for the target user needs to use the history of her ratings. In fact, it is usually important to have a large number of ratings available for the target user in order to make robust predictions without overfitting.

Time-Sensitive Recommender Systems

In many settings, the recommendations for an item might evolve with time. For example, the recommendations for a movie may be very different at the time of release from the recommendations

received several years later. In such cases, it is extremely important to incorporate temporal knowledge in the recommendation process. The temporal aspect in such recommender systems can be reflected in several ways:

1. The rating of an item might evolve with time, as community attitudes evolve and the interests of users change over time. User interests, likes, dislikes, and fashions inevitably evolve with time.

2. The rating of an item might be dependent on the specific time of day, day of week, month, or season. For example, it makes little sense to recommend winter clothing during the summer, or raincoats during the dry season.

The first type of recommender system is created by incorporating time as an explicit parameter in collaborative filtering systems. The second type can be viewed as a special case of context-based recommender systems. Temporal recommender systems are challenging because of the fact that the matrix of ratings is sparse, and the use of specific temporal context aggravates the sparsity problem. Therefore, it is particularly important to have access to large data sets in these settings.

Similarity Measures:

Similarity in a recommender system is about finding items (or users, or user and item) that are similar. How to measure it depends on which type of recommender you use.

If you are doing collaborative filtering, then two items are similar if a a certain number of people like or hate it the same way. An example could be that if you have 5 users who have given two items the following ratings.

user1: item1:5 item2:4 user2:
item1:4 item2:4 user3:
item1: 3 item2: 3

user4: item1: 1 item2: 2 user5:
item1: 1 item2: 2

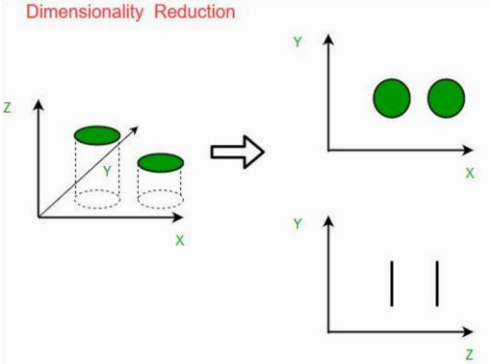
Without getting into how to calculate it, you can observe that there is a trend that people how like item1 also like item2 and people who dont like item1 also dont like item2. If you want to figure out how to calculate it, take a look at Cosine similarity

Dimensionality Reduction and Neighborhood Methods

Dimensionality reduction methods can be used to improve neighborhood-based methods both in terms of quality and in terms of efficiency. In particular, even though pair wise similarities are hard to robustly compute in sparse rating matrices, dimensionality reduction provides a dense lowdimensional representation in terms of latent factors. Therefore, such models are also referred to as latent factor models. Even when two users have very few items rated in common, a distance can be computed between their low-dimensional latent vectors. Furthermore, it is more efficient to determine the peer groups with low-dimensional latent vectors. Before discussing the details of dimensionality reduction methods, we make some comments about two distinct ways in which latent factor models are used in recommender systems:

1. A reduced representation of the data can be created in terms of either row-wise latent factors or in terms of column-wise latent factors. In other words, the reduced representation will either compress the item dimensionality or the user dimensionality into latent factors. This reduced representation can be used to alleviate the sparsity problem for neighborhood-based models. Depending on which dimension has been compressed into latent factors, the reduced representation can be used for either user-based neighborhood algorithms or item-based neighborhood algorithms.

2. The latent representations of both the row space and the column space are determined simultaneously. These latent representations are used to reconstruct the entire ratings matrix in one shot without the use of neighborhood-based methods.



SINGULAR VALUE DECOMPOSITION

Singular value decomposition (SVD) is a form of matrix factorization in which the columns of U and V are constrained to be mutually orthogonal. Mutual orthogonality has the advantage that the concepts can be completely independent of one another, and they can be geometrically interpreted in scatterplots. However, the semantic interpretation of such a decomposition is generally more difficult, because these latent vectors contain both positive and negative quantities, and are constrained by their orthogonality to other concepts. For a fully specified matrix, it is relatively easy to perform SVD with the use of eigen decomposition methods.

Consider the case in which the ratings matrix is fully specified. One can approximately factorize the ratings matrix R by using truncated SVD of rank k min{m, n}. Truncated SVD is computed as follows:

$$R \approx Q_k \Sigma_k P_k^T$$

Here, Q_k, Σ_k, and P_k are matrices of size m×k, k×k, and n×k, respectively. The matrices Q_k and P_k respectively contain the k largest eigenvectors of RRT and RT R, whereas the (diagonal) matrix Σk contains the (non-negative) square roots of the k largest eigen values of either matrix along its diagonal. It is noteworthy that the nonzero eigen values of RRT and RT R are the same, even though they will have a different number of zero eigen values when m not equal to n. The matrix P_k contains the top eigenvectors

of RT R, which is the reduced basis representation required for dimensionality reduction of the row space. These eigenvectors contain information about the directions of item -item correlations among ratings, a nd therefore they provide the ability to represent each user in a reduced number of dimensions in a rotated axis system.

It is easy to see from Equation 3.22 that SVD is inherently defined as a matrix factorization. Of course, the factorization here is into *three* matrices rather than *two*. However, the diagonal matrix Σ_k can be absorbed in either the user factors Q_k or the item factors P_k. By convention, the user factors and item factors are defined as follows:

$$U = Q_k \Sigma_k$$
$$V = P_k$$

As before, the factorization of the ratings matrix R is defined as $R = UV^T$. As long as the user and item factor matrices have orthogonal columns, it is easy to convert the resulting factorization into a form that is compliant with SVD (see Exercise 9). Therefore, the goal of the factorization process is to discover matrices U and V with orthogonal columns. Therefore, SVD can be formulated as the following optimization problem over the matrices U and V:

$$\text{Minimize } J = \frac{1}{2}||R - UV^T||^2$$
$$\text{subject to:}$$
$$\text{Columns of } U \text{ are mutually orthogonal}$$
$$\text{Columns of } V \text{ are mutually orthogonal}$$

It is easy to see that the only difference from the case of unconstrained factorization is the presence of orthogonality constraints. In other words, the same objective function is being optimized over a smaller space of solutions compared to unconstrained matrix factorization. Although one would expect that the presence of constraints would increase the error J of the approximation, it turns out that the optimal value of J is identical in the case of SVD and unconstrained matrix factorization, if the matrix R is fully specified and regularization is not used. Therefore, for fully specified matrices, the optimal solution to SVD is one of the alternate optima of unconstrained matrix factorization. This is not necessarily true in the cases in which R is not fully specified, and the objective function $J = \frac{1}{2}||R - UV^T||^2$ is computed *only over the observed entries*. In such cases, unconstrained matrix factorization will typically provide lower error on the observed entries. However, the relative performance on the unobserved entries can be unpredictable because of varying levels of *generalizability* of different models.

A Simple Iterative Approach to SVD

In this section, we discuss how to solve the optimization problem when the matrix R is incompletely specified. The first step is to mean-center each row of R by subtracting the average rating μi of the user i from it. These row-wise averages are stored because they will eventually be needed to reconstruct the raw ratings of the missing entries. Let the centered

The iterative steps 1 and 2 are executed to convergence. In this method, although the initialization step causes bias in the early SVD iterations, later iterations tend to provide more robust estimates. This is because the matrix $Q_k\Sigma_k P_k^T$ will differ from R to a greater degree in the biased entries. The final ratings matrix is then given by $Q_k\Sigma_k P_k^T$ at convergence.

The approach can become stuck in a local optimum when the number of missing entries is large. In particular, the local optimum at convergence can be sensitive to the choice of initialization. It is also possible to use the baseline predictor discussed in section 3.7.1 to perform more robust initialization. The idea is to compute an initial predicted value B_{ij} for user i and item j with the use of *learned* user and item biases. This approach is equivalent to applying the method in section 3.6.4.5 at $k = 0$, and then adding the bias of user i to that of item j to derive B_{ij}. The value of B_{ij} is subtracted from each observed entry (i, j) in the ratings matrix, and missing entries are set to 0 at initialization. The aforementioned iterative approach is applied to this adjusted matrix. The value of B_{ij} is added back to entry (i, j) at prediction time. Such an approach tends to be more robust because of better initialization.

Regularization can be used in conjunction with the aforementioned iterative method. The idea is to perform regularized SVD of R_f in each iteration rather than using only vanilla SVD. Because the matrix R_f is fully specified in each iteration, it is relatively easy to apply regularized SVD methods to these intermediate matrices. Regularized singular value decomposition methods for complete matrices are discussed in [541]. The optimal values of the regularization parameters λ_1 and λ_2 are chosen adaptively by using either the hold-out or the cross-validation methods.

matrix be denoted by R_c. Then, the missing entries of R_c are set to 0. This approach effectively sets the missing entries to the average rating of the corresponding user, because the missing entries of the centered matrix are set to 0. SVD is then applied to R_c to obtain the decomposition $R_c = Q_k\Sigma_k P_k^T$. The resulting user factors and item factors are given by $U = Q_k\Sigma_k$ and $V = P_k$. Let the ith row of U be the k-dimensional vector denoted by $\overline{u_i}$ and the jth row of V be the k-dimensional vector denoted by $\overline{v_j}$. Then, the rating $\hat{r}_{ij}$ of user i for item j is estimated as the following adjusted dot product of $\overline{u_i}$ and $\overline{v_j}$:

$$\hat{r}_{ij} = \overline{u_i} \cdot \overline{v_j} + \mu_i \tag{3.23}$$

Note that the mean μ_i of user i needs to added to the estimated rating to account for the mean-centering applied to R in the first step.

The main problem with this approach is that the substitution of missing entries with row-wise means can lead to considerable bias. A specific example of how column-wise mean substitution leads to bias is provided in section 2.5.1 of Chapter 2. The argument for row-wise substitution is exactly similar. There are several ways of reducing this bias. One of the methods is to use maximum-likelihood estimation [24, 472], which is discussed in section 2.5.1.1 of Chapter 2. Another approach is to use a method, which reduces the bias iteratively by improving the estimation of the missing entries. The approach uses the following steps:

1. **Initialization:** Initialize the missing entries in the ith row of R to be the mean μ_i of that row to create R_f.
2. **Iterative step 1:** Perform rank-k SVD of R_f in the form $Q_k\Sigma_k P_k^T$.
3. **Iterative step 2:** Readjust only the (originally) missing entries of R_f to the corresponding values in $Q_k\Sigma_k P_k^T$. Go to iterative step 1.

3.6.5.2 An Optimization-Based Approach

The iterative approach is quite expensive because it works with fully specified matrices. It is simple to implement for smaller matrices but does not scale well in large-scale settings. A more efficient approach is to add orthogonality constraints to the optimization model of the previous sections. A variety of gradient-descent methods can be used for solve the model. Let S be the set of specified entries in the ratings matrix. The optimization problem (with regularization) is stated as follows:

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j) \in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right)^2 + \frac{\lambda_1}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\lambda_2}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2$$

subject to:

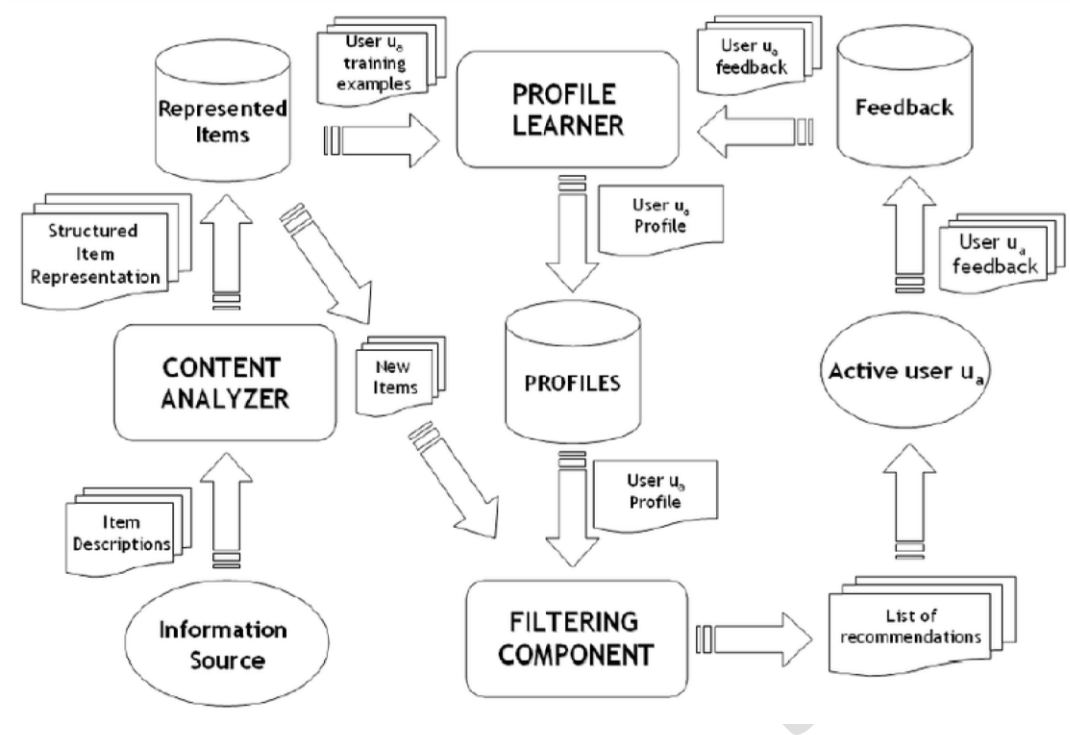
Columns of U are mutually orthogonal

Columns of V are mutually orthogonal

The primary difference of this model from unconstrained matrix factorization is the addition of orthogonality constraints, which makes the problem more difficult. For example, if one tries to directly use the update equations of the previous section on unconstrained matrix factorization, the orthogonality constraints will be violated. However, a variety of modified update methods exist to handle this case. For example, one can use a *projected gradient descent* [76] method, wherein all components of a particular column of U or V are updated at one time. In projected gradient descent, the descent direction for the pth column of U (or V), as indicated by the equations of the previous section, is projected in a direction that is orthogonal to the first $(p-1)$ columns of U (or V). For example, the implementation of section 3.6.4.3 can be adapted to learn orthogonal factors by projecting each factor in a direction orthogonal to those learned so far at each step. One can easily incorporate user and item biases by computing the baseline predictions B_{ij} (discussed in the previous section) and subtracting them from the observed entries in the ratings matrix before modeling. Subsequently, the baseline values can be added back to the predicted values as a postprocessing step.

UNIT II CONTENT-BASED RECOMMENDATION SYSTEMS

HIGH LEVEL ARCHITECTURE OF CONTENT-BASED RECOMMENDATION SYSTEMS



User-Based and Item-Based Collaborative Filtering on my last post, which uses the interactions of the users with the different items to make recommendations, we realized that these systems have some problems:

- Cold-start for new users.
- New-item problem.
- Sparcity.
- Transparency.

Content-Based Recommender Systems are born from the idea of using the content of each item for recommending purposes, and trying to solve the problems describes above. Here are some pros and cons from this new method:

Pros:

Unlike Collaborative Filtering, if the items have sufficient descriptions, we avoid the "new item problem".Content representations are varied and they open up the options to use different approaches like: text processing techniques, the use of semantic information, inferences, etc…It is easy to make a more transparent system: we use the same content to explain the recommendations.

Cons:

Content-Based RecSys tend to over-specialization: they will recommend items similar to those already consumed, with a tendecy of creating a "filter bubble".The methods based on Collaborative Filtering have shown to be, empirically, more precise when generating recommendations.

Content-based filtering has many benefits compared to collaborative filtering, including:

No data from other users is required to start making recommendations. Unlike collaborative filtering, content-based filtering doesn't need data from other users to create recommendations. Once a user has searched on and browsed a few items and/or completed some purchases, a content-based filtering system can begin making relevant recommendations. This makes it ideal for businesses that don't have an enormous pool of users to sample. It also works well for sellers that have many users but a small number of user interactions in specific categories or niches.

Recommendations are highly relevant to the user. Content-based recommenders can be highly tailored to the user's interests, including recommendations for niche items, because the method relies on matching the characteristics or attributes of a database object with the user's profile. For instance, content-based filtering will recognize a specific user's preferences and tastes, such as hot sauces made in Texas with organic Scotch bonnet peppers, and recommend products with the same attributes. Content-based filtering is also valuable for businesses with extensive libraries containing a single type of product, such as smartphones, where recommendations need to be based on many discrete features.

Recommendations are transparent to the user. Highly relevant recommendations project a sense of openness to the user, bolstering their trust level in offered recommendations. Comparatively, with collaborative filtering, instances are more likely to occur where users don't understand why they see specific recommendations. For example, let's say a group of users who purchased an umbrella also happen to buy down puffer coats. A collaborative system may recommend down puffer coats to other users who bought umbrellas but are uninterested in and have never browsed or purchased that product.

You avoid the "cold start" problem. Collaborative filtering creates a potential cold start scenario when a new website or community has few new users and lacks user connections. Although content-based filtering needs some initial inputs from users to start making recommendations, the quality of early recommendations is generally better than a collaborative system that requires the addition and correlation of millions of data points before becoming optimized.

Content-based filtering systems are generally easier to create. The data science behind a content-based filtering system is relatively straightforward compared to collaborative filtering systems intended to mimic user-to-user recommendations. The real work in content-based filtering is assigning the attributes.

Content representation and content similarity

The simplest way to describe catalog items is to maintain an explicit list of features for each item (also often called attributes, characteristics, or item profiles). For a book recommender, one could, for instance, use the genre, the author's name, the publisher, or anything else that describes the item and store. this information in a relational database system. When the user's preferences are described in terms of his or her interests using exactly this set of features, the recommendation task consists of matching item characteristics and user preferences.

A book recommender system can construct Alice's profile in different ways. The straightforward way is to explicitly ask Alice, for instance, for a desired price range or a set of preferred genres. The other way is to ask Alice to rate a set of items, either as a whole or along different dimensions. In the example, the set of preferred genres could be defined manually by Alice, whereas the system may automatically derive a set of keywords from those books that Alice liked, along with their average price. In the simplest case, the set of keywords corresponds to the set of all terms that appear in the document.

To make recommendations, content-based systems typically work by evaluating how strongly a not-yet-seen item is "similar" to items the active user has liked in the past. Similarity can be measured in different ways in the example. Given an unseen book B, the system could simply check whether the genre of the book at hand is in the list of Alice's preferred genres. Similarity in this case is either 0 or 1. Another option is to calculate the similarity or overlap of the involved keywords. As a typical similarity metric that is suitable for multivalued characteristics, we could, for example, rely on the Dice coefficient2 as follows: If every book Bi is described by a set of keywords keywords(Bi), the Dice coefficient measures the similarity between books bi and bj as

The vector space model and TF-IDF

A vector space model is a model where each term of the query is considered a vector dimension. It assumes that each term is a dimension that is orthogonal to all other terms, which means terms are modeled as occurring in documents independently. When evaluating the query to a document considered, one vector represents the query, and another, the document. The cosine similarity of the two vectors can be used to represent the relevance of the document to the query. A cosine value of 0 means that the query and the document vector are orthogonal and have no match (ie. none of the query terms were in the document). Cosine similarity is advantageous over Euclidean distance because cosine similarity measures the angle between two vectors, which means it captures the direction of the two vectors - while Euclidean distance captures the magnitude. Two vectors can be far apart by Euclidean distance, ie. imagine a short vector and a long vector, but have a small angle between them.

The content of a document can be encoded in such a keyword list in different ways. In a first, and very na¨ıve, approach, one could set up a list of all words that appear in all documents and describe each document by a Boolean vector, where a 1 indicates that a word appears in a document and a 0 that the word does not appear. If the user profile is described by a similar list (1 denoting interest in a keyword), document matching can be done by measuring the overlap of interest and document content.

Term frequency describes how often a certain term appears in a document (assuming that important words appear more often). To take the document length into account and to prevent longer documents from getting a higher relevance weight, some normalization of the document length should be done We search for the normalized term frequency value

TF(i, j) of keyword i in document j . Let freq(i, j) be the absolute number of occurrences of I in j . Given a keyword i, let Other Keywords(i, j) denote the set of the other keywords appearing in j Compute the maximum frequency maxOthers(i, j) as max(freq(z, j)), z ∈ OtherKeywords(i, j). Finally, calculate TF(i, j) as in

$$TF(i, j) = \frac{freq(i, j)}{maxOthers(i, j)}$$

Inverse document frequency is the second measure that is combined with term frequency. It aims at reducing the weight of keywords that appear very often in all documents. The idea is that those generally frequent words are not very helpful to discriminate among documents, and more weight should therefore be given to words that appear in only a few documents. Let N be the number of all recommendable documents and n(i) be the number of documents from N in which keyword i appears. The inverse document frequency for i is typically calculated as

$$IDF(i) = log\frac{N}{n(i)}$$

$$TF\text{-}IDF(i, j) = TF(i, j) * IDF(i)$$

Improving the vector space model/limitations

Stop words and stemming.

A straightforward method is to remove so-called stop words. In the English language these are,

for instance, prepositions and articles such as “a”, “the”, or “on”, which can be removed from the document vectors because they will appear in nearly all documents. Another commonly used technique is called stemming or conflation, which aims to replace variants of the same word by their common stem (root word). The word “stemming” would, for instance, be replaced by “stem”, “went” by “go”, and so forth. These techniques further reduce the vector size and at the same time, help to improve the matching process in cases in which the word stems are also used in the user profile. Stemming procedures are commonly implemented as a combination of morphological analysis.

Size cutoffs.

Another straightforward method to reduce the size of the document representation and hopefully remove “noise” from the data is to use only the n most informative words. The optimal number of words to be used was determined experimentally for the Syskill & Webert system for several domains. The evaluation showed that if too few keywords (say, fewer than 50) were selected, some possibly important document features were not covered. On the other hand, when including too many features (e.g., more than 300), keywords are used in the document model that have only limited importance and therefore represent noise that actually worsens the recommendation accuracy. In principle, complex techniques for “feature selection” can also be applied for determining the most informative keywords.

Phrases:

A further possible improvement with respect to representation accuracy is to use

"phrases as terms", which are more descriptive for a text than single words alone. Phrases, or composed words such as "United Nations", can be encoded as additional dimensions in the vector space. Detection of phrases can be done by looking up manually defined lists or by applying statistical analysis techniques

Similarity-based retrieval

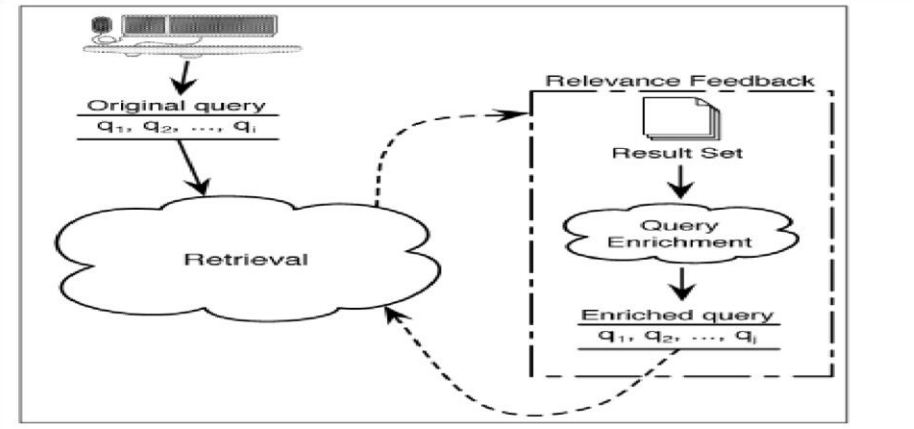
When the item selection problem in collaborative filtering can be described as "recommend items that similar users liked", content-based recommendation is commonly described as "recommend items that are similar to those the user liked in the past". Therefore, the task for a recommender system is again –based on a user profile – to predict, for the items that the current user has not seen, whether he or she will like them **Nearest neighbors**

A first, straightforward, method for estimating to what extent a certain document will be of interest to a user is simply to check whether the user liked similar documents in the past. To do this, two pieces of information are required. First, we need some history of "like/dislike" statements made by the user about previous items. Similar to collaborative approaches, these ratings can be provided either explicitly via the user interface or implicitly by monitoring the user's behavior. Second, a measure is needed that captures the similarity of two documents. The prediction for a not-yet-seen item d is based on letting the k most similar items for which a rating exists "vote" for n. If, for instance, four out of k = 5 of the most similar items were liked by the current user, the system may guess that the chance that d will also be liked is relatively high. Besides varying the neighborhood size k, several other variations are possible, such as binarization of ratings, using a minimum similarity threshold, or weighting of the votes based on the degree of similarity.

Relevance feedback – Rocchio's method

Relevance feedback is a query expansion and refinement technique with a long history. First proposed in the 1960s, it relies on user interaction to identify relevant documents in a ranking based on the initial query. Other semi-automatic techniques were discussed in the last section, but instead of choosing from lists of terms or alternative queries, in relevance feedback the user indicates which documents are interesting (i.e., relevant) and possibly which documents are completely off-topic (i.e., non-relevant). Based on this information, the system automatically reformulates the query by adding terms and reweighting the original terms, and a new ranking is generated using this modified query**.**

This process is a simple example of using machine learning in information retrieval, where training data (the identified relevant and non-relevant documents) is used to improve the system's performance. Modifying the query is in fact equivalent to learning a classifier that distinguishes between relevant and non-relevant documents.



For example, an initial query "find information surrounding the various conspiracy theories about the assassination of John F. Kennedy" has both useful keywords and noise. The most useful keywords are probably assassination. Like many queries (in terms of retrieval) there is some meaningless information. Terms such as various and information are probably not stop words (i.e., frequently used words that are typically ignored by an information retrieval system such as a, an, and, the), but they are more than likely not going to help retrieve relevant documents. The idea is to use all terms in the initial query and ask the user if the top ranked documents are relevant. The hope is that the terms in the top ranked documents that are said to be relevant will be "good" terms to use in a subsequent query.

Rocchio's approach used the vector space model to rank documents. The query is represented by a vector Q, each document is represented by a vectorDi, and a measure of relevance between the query and the document vector is computed as SC(Q, Di), where SC is the similarity coefficient. The SC is computed as an inner product of the document and query vector or the cosine of the angle between the two vectors. The basic assumption is that the user has issued a query Q and retrieved a set of documents. The user is then asked whether or not the documents are relevant. After the user responds, the set R contains the nl relevant document vectors, and the set S contains the n2 non-relevant document vectors. Rocchio builds the new query Q' from the old query Q using the equation given below:

Ri and Si are individual components of R and S, respectively. The document vectors from the relevant documents are added to the initial query vector, and the vectors from the non-relevant documents are subtracted .If all documents are relevant, the third term does not appear. To ensure that the new information does not completely override the original query, all vector modifications are normalized by the number of relevant and nonrelevant documents. The process can be repeated such that Qi+1 is derived from Qi for as many iterations as desired.

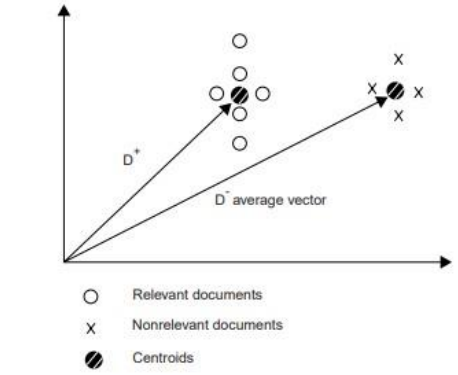
In addition to using values n1 and n2, it is possible to use arbitrary weights.The equation now becomes:

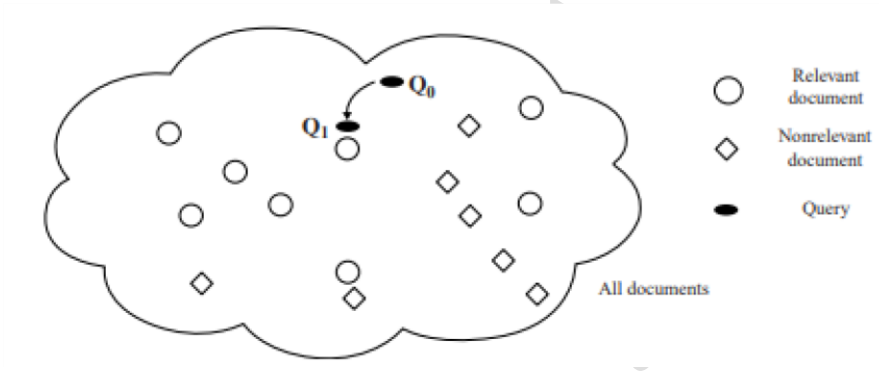
$$Q' = \alpha Q + \beta \sum_{i=1}^{n_1} \frac{R_i}{n_1} - \gamma \sum_{i=1}^{n_2} \frac{S_i}{n_2}$$

Not all of the relevant or non-relevant documents must be used. Addingthresholds na and nb to indicate the thresholds for relevant and non-relevantvectors results in:

$$Q' = \alpha Q + \beta \sum_{i=1}^{min(n_a,n_1)} \frac{R_i}{n_1} - \gamma \sum_{i=1}^{min(n_b,n_2)} \frac{S_i}{n_2}$$

$$Q' = \alpha Q + \beta \sum_{i=1}^{n_1} R_i - S_1$$





The probabilistic model

The probabilistic model computes the similarity coefficient (SC) between a query and a document as the probability that the document will be relevant to the query. This reduces the relevance ranking problem to an application of probability theory. Probability theory can be used to compute a measure of relevance between a query and a document. *Two fundamentally different approaches were proposed*. The first relies on usage patterns to predict relevance, the second uses each term in the query as clues as to whether or not a document is relevant.

Simple Term Weights:

The use of term weights is based on the Probability Ranking Principle (PRP), which assumes that optimal effectiveness occurs when documents are ranked based on an estimate of the probability of their relevance to a query. The key is to assign probabilities to components of the query and then use each of these as evidence in computing the final probability that a document is relevant to the query. The terms in the query are assigned weights which correspond to the probability that a particular term, in a match with a given query, will retrieve are relevant document. The weights for each term in the query are combined to obtain a final measure of relevance.

Suppose we are trying to predict whether or not a softball team called the Salamanders will win one of its games. We might observe, based on past experience, that they usually win on sunny days when their best shortstop plays. This means that two pieces of evidence, outdoor-conditions and presence of good-shortstop, might be used.

For any given game, there is a seventy five percent chance that the team will win if the weather is sunny and a sixty percent chance that the team will win if the shortstop plays. Therefore, we write:

P(win I sunny) = 0.75

P(win I good-shortstop) = 0.6

The conditional probability t hat the team will win given both situations is written as p(win I sunny, good shortstop). This is read "the probability that the team will win given that there is a sunny day and the good -shortstop plays."We have two pieces of evidence indicating that the Salamanders will win. Intuition says that together the two pieces should be stronger than either alone. This method of combining them is to "look at the odds." A seventy -five percent chance of winning is a twenty-five percent chance of losing, and a sixty percent chance of winning is a forty percent chance of losing. Let us assume the independence of the pieces of evidence.

P(win I sunny, good-$_{\text{shortstop}}$) = α

P(win I sunny) = β

P(win I good-$_{\text{shortstop}}$) = γ

By Bayes' Theorem:

$$\alpha = \frac{P(\text{win}, sunny, good\text{–}shortstop)}{P(sunny, good\text{–}shortstop)} = \frac{P(sunny, good\text{–}shortstop|\text{win})P(\text{win})}{P(sunny, good\text{–}shortstop)}$$

Therefore

$$\frac{\alpha}{1-\alpha} = \frac{P(sunny, good\text{–}shortstop|\text{win})P(\text{win})}{P(sunny, good\text{–}shortstop|\text{lose})P(\text{lose})}$$

Solving for the first term (because of the independence assumptions):

$$\frac{P(sunny, good\text{–}shortstop|\text{win})}{P(sunny, good\text{–}shortstop|\text{lose})} = \frac{P(\text{sunny}|\text{win})P(\text{good}\text{–}shortstop|\text{win})}{P(\text{sunny}|\text{lose})P(\text{good}\text{–}shortstop|\text{lose})}$$

UNIT III COLLABORATIVE FILTERING

Collaborative recommendation

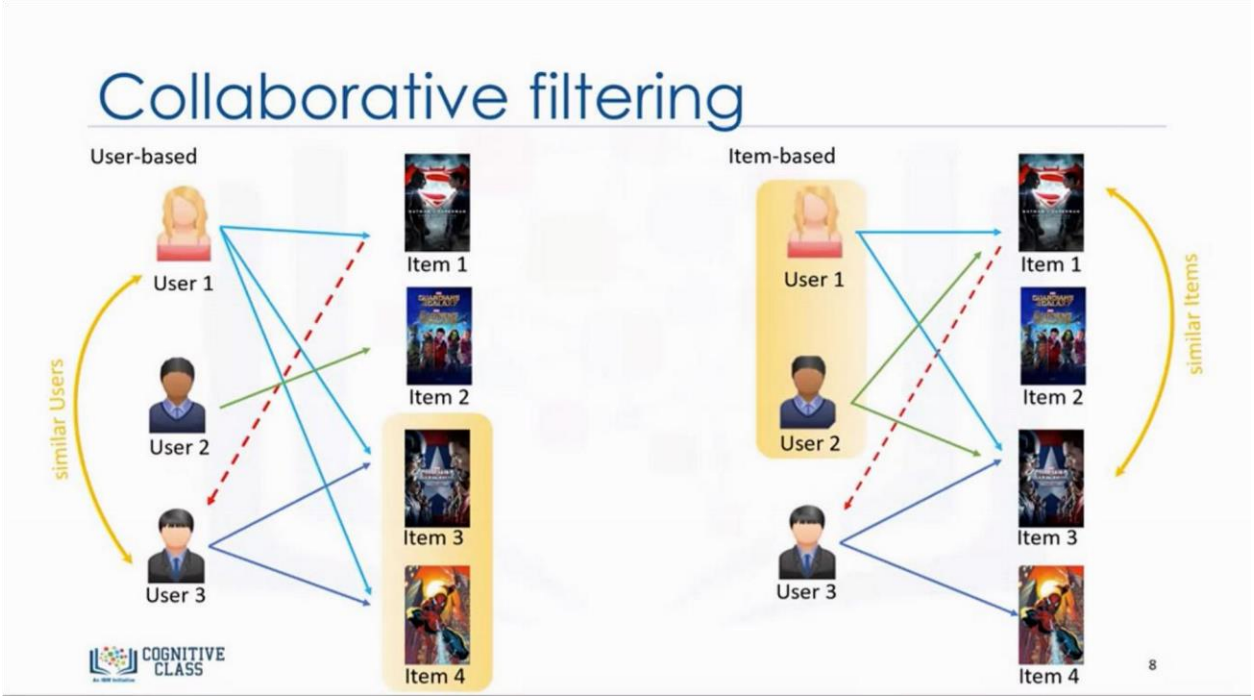
The main idea of collaborative recommendation approaches is to exploit information about the past behavior or the opinions of an existing user community for predicting which items the current user of the system will most probably like or be interested in. These types of systems are in widespread industrial use today, in particular as a tool in online retail sites to customize the content to the needs of a particular customer and to thereby promote additional items and increase sales.

From a research perspective, these types of systems have been explored for many years, and their advantages, their performance, and their limitations are nowadays well understood. Over the years, various algorithms and techniques have been proposed and successfully evaluated on real-world and artificial test data.

Pure collaborative approaches take a matrix of given user–item ratings as the only input and typically produce the following types of output: (a) a (numerical) prediction indicating to what degree the current user will like or dislike a certain item and (b) a list of n recommended items. Such a top-N list should, of course, not contain items that the current user has already bought.

Colloborative Filtering

We shall now take up a significantly different approach to recommendation. Instead of using features of items to determine their similarity, we focus on the similarity of the user ratings for two items. That is, in place of the item-profile vector for an item, we use its column in the utility matrix. Further, instead of contriving a profile vector for users, we represent them by their rows in the utility matrix. Users are similar if their vectors are close according to some distance measure such as Jaccard or cosine distance. Recommendation for a user U is then made by looking at the users that are most similar to U in this sense, and recommending items that these users like. The process of identifying similar users and recommending what similar users like is called collaborative filtering

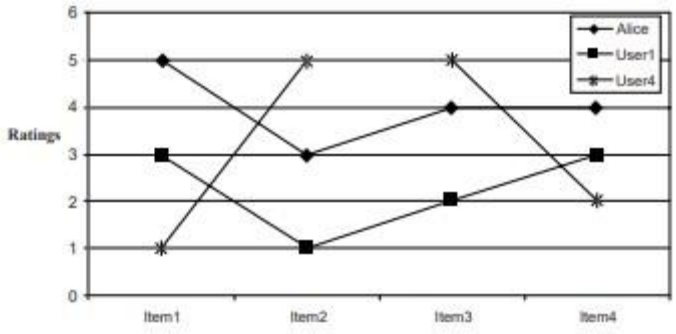


User-based nearest neighbor recommendation

The first approach we discuss here is also one of the earliest methods, called user-based nearest neighbor recommendation. The main idea is simply as follows: given a ratings database and the ID of the current (active) user as an input, identify other users (sometimes referred to as peer users or nearest neighbors) that had similar preferences to those of the active user in the past. Then, for every product p that the active user has not yet seen, a prediction is computed based on the ratings for p made by the peer users. The underlying assumptions of such methods are that (a) if users had similar tastes in the past they will have similar tastes in the future and (b) user preferences remain stable and consistent over time.

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

Let us examine a first example. Table shows a database of ratings of the current user, Alice, and some other users. Alice has, for instance, rated “Item1” with a “5” on a 1-to-5 scale, which means that she strongly liked this item. The task of a recommender system in this simple example is to determine whether Alice will like or dislike “Item5”, which Alice has not yet rated or seen. If we can predict that Alice will like this item very strongly, we should include it in Alice’s recommendation list. To this purpose, we search for users whose taste is similar to Alice’s and then take the ratings of this group for “Item5” to predict whether Alice will like this item.



Neighborhood selection

In our example, we intuitively decided not to take all neighbors into account (neighborhood selection). For the calculation of the predictions, we included only those that had a positive correlation with the active user (and, of course, had rated the item for which we are looking for a prediction). If we included all users in the neighborhood, this would not only negatively influence the performance with respect to the required calculation time, but it would also have an effect on the accuracy of the recommendation, as the ratings of other users who are not really comparable would be taken into account.

The common techniques for reducing the size of the neighborhood are to define a specific minimum threshold of user similarity or to limit the size to a fixed number and to take only the k nearest neighbors into account The value chosen for k – the size of the neighborhood – does not influence coverage. However, the problem of finding a good value for k still exists: When the number of neighbors k taken into account is too high, too many neighbors with limited similarity bring additional "noise" into the predictions. When k is too small the quality of the predictions may be negatively affected. An analysis of the MovieLens dataset indicates that "in most real-world situations, a neighborhood of 20 to 50 neighbors seems reasonable"

Item-based nearest neighbor recommendation

Although user-based CF approaches have been applied successfully in different domains, some serious challenges remain when it comes to large e-commerce sites, on which we must handle millions of users and millions of catalog items. In particular, the need to scan a vast number of potential neighbors makes it impossible to compute predictions in real time. Large-scale e-commerce sites, therefore, often implement a different technique, item-based recommendation, which is more apt for offline preprocessing1 and thus allows for the computation of recommendations in real time even for a very large rating matrix

The main idea of item-based algorithms is to compute predictions using the similarity between items and not the similarity between users. Let us examine our ratings database again and make a prediction for Alice for Item5. We first compare the rating vectors of the other items and look for items that have ratings similar to Item5. In the example, we see that the ratings for Item5 (3, 5, 4, 1) are similar to the ratings of Item1 (3, 4, 3, 1) and there is also a partial similarity with Item4 (3, 3, 5, 2). The idea of item-based recommendation is now to simply look at Alice's ratings for these similar items. Alice gave a "5" to Item1 and a "4" to Item4. An item-based algorithm computes a weighted average of these other ratings and will predict a rating for Item5 somewhere between 4 and 5.

Preprocessing data for item-based filtering

Item-to-item collaborative filtering is the technique used by Amazon.com to recommend books or CDs to their customers. Linden et al. (2003) report on how this technique was implemented for Amazon's online shop, which, in 2003, had 29 million users and millions of catalog items. The main problem with traditional user-based CF is that the algorithm does not scale well for such large numbers of users and catalog items. Given M customers and N catalog items, in the worst case, all M records containing up to N items must be evaluated. For realistic scenarios, Linden et al. (2003) argue that the actual complexity is much lower because most of the customers have rated or bought only a very small number of items. Still, when the number of customers M is around several million, the calculation of predictions in real time is still infeasible, given the short response times that must be obeyed in the online environment.

For making item-based recommendation algorithms applicable also for large scale ecommerce sites without sacrificing recommendation accuracy, an approach based on offline pre computation of the data is typically chosen. The idea is to construct in advance the item similarity matrix that describes the pair wise similarity of all catalog items. At run time, a prediction for a product p and user u is made by determining the items that are most similar to i and by building the weighted sum of u's ratings for these items in the neighborhood. The number of neighbors to be taken into account is limited to the number of items that the active user has rated. As the number of such items is typically rather small, the computation of the prediction can be easily accomplished within the short time frame allowed in interactive online applications

With respect to memory requirements, a full item similarity matrix for N items can theoretically have up to N^2 entries. In practice, however, the number of entries is significantly lower, and further techniques can be applied to reduce the complexity. The options are, for instance, to consider only items that have a minimum number of co-ratings or to memorize only a limited neighborhood for each item; this, however, increases the danger that no prediction can be made for a given item

Measuring Similarity

The first question we must deal with is how to measure similarity of users or items from their rows or columns in the utility matrix. This data is too small to draw any reliable conclusions, but its small size will make clear some of the pitfalls in picking a distance measure. Observe specifically the users A and C. They rated two movies in common, but they appear to have almost diametrically opposite opinions of these movies. We would expect that a good distance measure would make them rather far apart. Here are some alternative measures to consider.

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Example 9.7: A and B have an intersection of size 1 and a union of size 5. Thus, their Jaccard similarity is $1/5$, and their Jaccard distance is $4/5$; i.e., they are very far apart. In comparison, A and C have a Jaccard similarity of $2/4$, so their Jaccard distance is the same, $1/2$. Thus, A appears closer to C than to B. Yet that conclusion seems intuitively wrong. A and C disagree on the two movies they both watched, while A and B seem both to have liked the one movie they watched in common. □

Cosine Distance

We can treat blanks as a 0 value. This choice is questionable, since it has the effect of treating the lack of a rating as more similar to disliking the movie than liking it.

Example 9.8: The cosine of the angle between A and B is

$$\frac{4 \times 5}{\sqrt{4^2 + 5^2 + 1^2}\sqrt{5^2 + 5^2 + 4^2}} = 0.380$$

Jaccard Distance

We could ignore values in the matrix and focus only on the sets of items rated. If the utility matrix only reflected purchases, this measure would be a good one to choose. However, when utilities are more detailed ratings, the Jaccard distance loses important information.

The cosine of the angle between A and C is

$$\frac{5 \times 2 + 1 \times 4}{\sqrt{4^2 + 5^2 + 1^2}\sqrt{2^2 + 4^2 + 5^2}} = 0.322$$

Since a larger (positive) cosine implies a smaller angle and therefore a smaller distance, this measure tells us that A is slightly closer to B than to C. □

Rounding the Data

We could try to eliminate the apparent similarity between movies a user rates highly and those with low scores by rounding the ratings. For instance, we could consider ratings of 3, 4, and 5 as a "1" and consider ratings 1 and 2 as unrated. The utility matrix would then look as in Fig. 9.5. Now, the Jaccard distance between A and B is 3/4, while between A and C it is 1; i.e., C appears further from A than B does, which is intuitively correct. Applying cosine distance to Fig. 9.5 allows us to draw the same conclusion.

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	1			1			
B	1	1	1				
C					1	1	
D		1					1

Figure 9.5: Utilities of 3, 4, and 5 have been replaced by 1, while ratings of 1 and 2 are omitted

Normalizing Ratings

If we normalize ratings, by subtracting from each rating the average rating of that user, we turn low ratings into negative numbers and high ratings into positive numbers. If we then take the cosine distance, we find that users with opposite views of the movies they viewed in common will have vectors in almost opposite directions, and can be considered as far apart as possible. However, users with similar opinions about the movies rated in common will have a relatively small angle between them

Ratings

Before we discuss further techniques for reducing the computational complexity and present additional algorithms that operate solely on the basis of a user–item ratings matrix, we present a few general remarks on ratings in collaborative recommendation approaches.

Implicit and explicit ratings

Among the existing alternatives for gathering users' opinions, asking for explicit item ratings is probably the most precise one. In most cases, five-point or seven-point Likert response scales ranging from "Strongly dislike" to "Strongly like" are used; they are then internally transformed to numeric values so the previously mentioned similarity measures can be applied. Some aspects of the usage of different rating

scales, such as how the users' rating behavior changes when different scales must be used and how the quality of recommendation

What has been observed is that in the movie domain, a five-point rating scale may be too narrow for users to express their opinions, and a ten-point scale was better accepted. The main arguments for this approach are that there is no precision loss from the discretization, user preferences can be captured at a finer granularity, and, finally, end users actually "like" the graphical interaction method, which also lets them express their rating more as a "gut reaction" on a visual level.

The main problems with explicit ratings are that such ratings require additional efforts from the users of the recommender system and users might not be willing to provide such ratings as long as the value cannot be easily seen. Thus, the number of available ratings could be too small, which in turn results in poor recommendation quality.

Besides that, one can observe that in the last few years in particular, with the emergence of what is called Web 2.0, the role of online communities has changed and users are more willing to contribute actively to their community's knowledge. Still, in light of these recent developments, more research focusing on the development of techniques and measures that can be used to persuade the online user to provide more ratings is required.

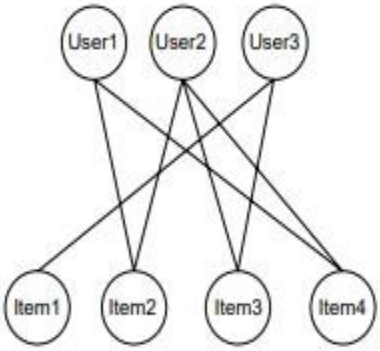
Implicit ratings are typically collected by the web shop or application in which the recommender system is embedded. When a customer buys an item, for instance, many recommender systems interpret this behavior as a positive rating. The system could also monitor the user's browsing behavior. If the user retrieves a page with detailed item information and remains at this page for a longer period of time, for example, a recommender could interpret this behavior as a positive orientation toward the item.

Although implicit ratings can be collected constantly and do not require additional efforts from the side of the user, one cannot be sure whether the user behavior is correctly interpreted. A user might not like all the books he or she has bought; the user also might have bought a book for someone else. Still, if a sufficient number of ratings is available, these particular cases will be factored out by the high number of cases in which the interpretation of the behavior was right

Data sparsity and the cold-start problem

In the rating matrices used in the previous examples, ratings existed for all but one user–item combination. In real-world applications, of course, the rating matrices tend to be very sparse, as customers typically provide ratings for (or have bought) only a small fraction of the catalog items.

In general, the challenge in that context is thus to compute good predictions when there are relatively few ratings available. One straightforward option for dealing with this problem is to exploit additional information about the users, such as gender, age, education, interests, or other available information that can help to classify the user. The set of similar users (neighbors) is thus based not only on the analysis of the explicit and implicit ratings, but also on information external to the ratings matrix



however, no longer "purely" collaborative, and new questions of how to acquire the additional information and how to combine the different classifiers arise. Still, to reach the critical mass of users needed in a collaborative approach, such techniques might be helpful in the ramp-up phase of a newly installed recommendation service.

A 0 in this matrix should not be interpreted as an explicit (poor) rating, but rather as a missing rating. Assume that we are looking for a recommendation for User1. When using a standard CF approach, User2 will be considered a peer for User1 because they both bought Item2 and Item4. Thus Item3 will be recommended to User1 because the nearest neighbor, User2, also bought or liked it. Huang et al. (2004) view the recommendation problem as a graph analysis problem, in which recommendations are determined by determining paths between users and items. In a standard user-based or item-based CF approach, paths of length 3 will be considered – that is, Item3 is relevant for User1 because there exists a three-step path (User1–Item2–User2–Item3) between them. Because the number of such paths of length 3 is small in sparserating databases, the idea is to also consider longer paths (indirect associations)to compute recommendations.

	Item1	Item2	Item3	Item4
User1	0	1	0	1
User2	0	1	1	1
User3	1	0	1	0

Further model-based and preprocessing-based approaches

Collaborative recommendation techniques are often classified as being either memory-based or model-based. The traditional user-based technique is said to be memory-based because the original rating database is held in memory and used directly for generating the recommendations. In model-based approaches, on the other hand, the raw data are first processed offline, as described for item based filtering or some dimensionality reduction techniques. At run time, only the precomputed or "learned" model is required to make predictions. Although memory-based approaches are theoretically more precise because full data are available for generating recommendations, such systems face problems of scalability if we think again of databases of tens of millions of users and millions of items.

The Duality of Similarity

The utility matrix can be viewed as telling us about users or about items, or both. It is important to realize that any of the techniques for finding similar users can be used on columns of the utility matrix to find similar items. There are two ways in which the symmetry is broken in practice.

1. We can use information about users to recommend items. That is, given a user, we can find some number of the most similar users, perhaps using the techniques. We can base our recommendation on the decisions made by these similar users, e.g., recommend the items that the greatest number of them have purchased or rated highly. However, there is no symmetry. Even if we find pairs of similar items, we need to take an additional step in order to recommend items to users. This point is explored further at the end of this subsection.
2. There is a difference in the typical behavior of users and items, as it pertains to similarity. Intuitively, items tend to be classifiable in simple terms. For example, music tends to belong to a single genre. It is impossible, e.g., for a piece of music to be both 60's rock and 1700's baroque. On the other hand, there are individuals who like both 60's rock and 1700's baroque, and who buy examples of both types of music. The consequence is that it is easier to discover items that are similar because they belong to the same genre, than it is to detect that two users are similar because they prefer one genre in common, while each also likes some genres that the other doesn't care for.

As we suggested, one way of predicting the value of the utility matrix entry for user U and item I is to find the n users (for some predetermined n) most similar to U and average their ratings for item I, counting only those among the n similar users who have rated I. It is generally better to normalize the matrix first. That is, for each of the n users subtract their average rating for items from their rating for i. Average the difference for those users who have rated I, and then add this average to the average rating that U gives for all items. This normalization adjusts the estimate in the case that U tends to give very high or very low ratings, or a large fraction of the similar users who rated I (of which there may be only a few) are users who tend to rate very high or very low.

Dually, we can use item similarity to estimate the entry for user U and item I. Find the m items most similar to I, for some m, and take the average rating, among the m items, of the ratings that U has given. As for user-user similarity, we consider only those items among the m that U has rated, and it is probably wise to normalize item ratings first.

Note that whichever approach to estimating entries in the utility matrix we use, it is not sufficient to find only one entry. In order to recommend items to a user U, we need to estimate every entry in the row of the utility matrix for U, or at least find all or most of the entries in that row that are blank but have a high estimated value. There is a tradeoff regarding whether we should work from similar users or similar items.

• If we find similar users, then we only have to do the process once for user U. From the set of similar users we can estimate all the blanks in the utility matrix for U. If we work from similar items, we have to compute similar items for almost all items, before we can estimate the row for U.

- On the other hand, item-item similarity often provides more reliable information, because of the phenomenon observed above, namely that it is easier to find items of the same genre than it is to find users that like only items of a single genre.

Whichever method we choose, we should precompute preferred items for each user, rather than waiting until we need to make a decision. Since the utility matrix evolves slowly, it is generally sufficient to compute it infrequently and assume that it remains fixed between recomputations.

Clustering Users and Items

It is hard to detect similarity among either items or users, because we have little information about user-item pairs in the sparse utility matrix. In the perspective even if two items belong to the same genre, there are likely to be very few users who bought or rated both. Likewise, even if two users both like a genre or genres, they may not have bought any items in common.

One way of dealing with this pitfall is to cluster items and/or users. Select any of the distance measures suggested, or any other distance measure, and use it to perform a clustering of, say, items. Any of the methods suggested in Chapter 7 can be used. However, we shall see that there may be little reason to try to cluster into a small number of clusters immediately. Rather, a hierarchical approach, where we leave many clusters unmerged may suffice as a first step. For example, we might leave half as many clusters as there are items.

	HP	TW	SW
A	4	5	1
B	4.67		
C		2	4.5
D	3		3

Dimensionality Reduction

An entirely different approach to estimating the blank entries in the utility matrix is to conjecture that the utility matrix is actually the product of two long, thin matrices. This view makes sense if there are a relatively small set of features of items and users that determine the reaction of most users to most items. In this section, we sketch one approach to discovering two such matrices; the approach is called "UV-decomposition," and it is an instance of a more general theory called SVD (singular-value decomposition).

UV-Decomposition

Consider movies as a case in point. Most users respond to a small number of features; they like certain genres, they may have certain famous actors or actresses that they like, and perhaps there are a few directors with a significant following. If we start with the utility matrix M, with n rows and m columns (i.e., there are n users and m items), then we might be able to find a matrix U with n rows and d columns and a matrix V with d rows and m columns, such that UV closely approximates M in those entries where M is nonblank. If so, then we have established that there are d dimensions that allow us to characterize both users and items closely. We can then use the entry in the product UV to estimate the corresponding blank entry in utility matrix M. This process is called *UV-decomposition* of M.

$$\begin{bmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \\ u_{51} & u_{52} \end{bmatrix} \times \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \end{bmatrix}$$

V shown with their entries as variables to be determined. This example is essentially the smallest nontrivial case where there are more known entries than there are entries in U and V combined, and we therefore can expect that the best decomposition will not yield a product that agrees exactly in the nonblank entries of M.

Root-Mean-Square Error

While we can pick among several measures of how close the product UV is to M, the typical choice is the root-mean-square error (RMSE), where we

1. Sum, over all nonblank entries in M the square of the difference between that entry and the corresponding entry in the product UV .
2. Take the mean (average) of these squares by dividing by the number of terms in the sum (i.e., the number of nonblank entries in M).
3. Take the square root of the mean.

Minimizing the sum of the squares is the same as minimizing the square root of the average square, so we generally omit the last two steps in our running example.

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

The Netflix Challenge

A significant boost to research into recommendation systems was given when Netflix offered a prize of $1,000,000 to the first person or team to beat their own recommendation algorithm, called CineMatch, by 10%. After over three years of work, the prize was awarded in September, 2009. The Netflix challenge consisted of a published dataset, giving the ratings by approximately half a million users on (typically small subsets of) approximately 17,000 movies. This data was selected from a larger dataset, and proposed algorithms were tested on their ability to predict the ratings in a secret remainder of the larger dataset. The information for each (user, movie) pair in the published dataset included a rating (1–5 stars) and the date on which the rating was made.

The RMSE was used to measure the performance of algorithms. CineMatch has an RMSE of approximately 0.95; i.e., the typical rating would be off by almost one full star. To win the prize, it was necessary that your algorithm have an RMSE that was at most 90% of the RMSE of CineMatch. The bibliographic notes for this chapter include references to descriptions of the winning algorithms. Here, we mention some interesting and perhaps unintuitive facts about the challenge.

- CineMatch was not a very good algorithm. In fact, it was discovered early that the obvious algorithm of predicting, for the rating by user u on movie m, the average of:

1. The average rating given by u on all rated movies and
2. The average of the ratings for movie m by all users who rated that movie was only 3% worse than CineMatch.

- The UV-decomposition algorithm described in Section 9.4 was found by three students (Michael Harris, Jeffrey Wang, and David Kamm) to give a 7% improvement over CineMatch, when coupled with normalization and a few other tricks.

- The winning entry was actually a combination of several different algorithms that had been developed independently. A second team, which submitted an entry that would have won, had it been submitted a few minutes earlier, also was a blend of independent algorithms. This strategy – combining different algorithms – has been used before in a number of hard problems and is something worth remembering.

- Several attempts have been made to use the data contained in IMDB, the Internet movie database, to match the names of movies from the Netflix challenge with their names in IMDB, and thus extract useful information not contained in the Netflix data itself. IMDB has information about actors and directors, and classifies movies into one or more of 28 genres. It was found that genre and other information was not useful. One possible reason is the machine-learning algorithms were able to discover the relevant information anyway, and a second is that the entity resolution problem of matching movie names as given in Netflix and IMDB data is not that easy to solve exactly.

- Time of rating turned out to be useful. It appears there are movies that are more likely to be appreciated by people who rate it immediately after viewing than by those who wait a while and then rate it. “Patch Adams” was given as an example of such a movie. Conversely, there are other movies that were not liked by those who rated it immediately, but were better appreciated after a while; “Memento” was cited as an example. While one cannot tease out of the data information about how long was the delay between viewing and rating, it is generally safe to assume that most people see a movie shortly after it comes out. Thus, one can examine the ratings of any movie to see if its ratings have an upward or downward slope with time.

UNIT IV ATTACK-RESISTANT RECOMMENDER SYSTEMS

Introduction

The input to recommender systems is typically provided through open platforms. Almost anyone can register and submit a review at sites such as Amazon.com and Epinions.com. Like any other datamining system, the effectiveness of a recommender system depends almost exclusively on the quality of the data available to it. Unfortunately, there are significant motivations for participants to submit incorrect feedback about items for personal gain or for malicious reasons:

- The manufacturer of an item or the author of a book might submit fake (positive) reviews on Amazon in order to maximize sales. Such attacks are also referred to as product push attacks.
- The competitor of an item manufacturer might submit malicious reviews about the item. Such attacks are also referred to as nuke attacks.

It is also possible for an attack to be designed purely to cause mischief and disrupt the underlying system, although such attacks are rare relative to attacks motivated by personal gain. This chapter studies only attacks that are motivated to achieve a particular outcome in the recommendation process. The person making the attack on the recommender system is also referred to as the adversary.

By creating a concerted set of fake feedbacks from many different users, it is possible to change the predictions of the recommender system. Such users become shills in the attack process. Therefore, such attacks are also referred to as shilling attacks. It is noteworthy that the addition of a single fake user or rating is unlikely to achieve the desired outcome.

Attacks can also be classified based on the amount of knowledge required to mount them successfully. Some attacks require only limited knowledge about the ratings distribution. Such attacks are referred to as low-knowledge attacks. On the other hand, attacks that require a large amount of knowledge about the ratings distribution are referred to as high knowledge attacks. As a general rule, a trade-off exists between the amount of knowledge required to make an attack and the efficiency of the attack. If adversaries have more knowledge about the ratings distribution, then they can generally make more efficient attacks

Understanding the Trade-Offs in Attack Models

Attack models have a number of natural trade-offs between the efficiency of the attack and the amount of knowledge required to mount a successful attack. Furthermore, the effectiveness of a particular attack may depend on the specific recommendation algorithm being used. In order to understand this point, we will use a specific example.

Consider the toy example illustrated in Table where we have 5 items and 6 (real) users. The ratings are all drawn from the range of 1 to 7, where 1 indicates extreme dislike and 7 indicates extreme like. Furthermore, an attacker has injected 5 fake profiles, which are denoted with the labels Fake-1, Fake-2, Fake-3, Fake-4, and Fake-5. The goal of this attacker is to inflate the ratings of item 3. Therefore, this attacker has chosen a rather naive attack, in which they have inserted fake profiles containing a single

item corresponding to the item However, such an attack is not particularly efficient. It is highly detectable because only a single item is included in every injected profile with a very similar rating. Furthermore, such ratings injections are unlikely to have a large impact on most recommendation algorithms. For example, consider the non-personalized recommendation algorithm in which the highest rated item is recommended. In such a case, the naive attack algorithm will increase the predicted rating of item 3, and it will be more likely to be recommended. The attack might also increase the predicted rating of item 3 in cases where item bias is explicitly used as a part of the model construction. There is, however, little chance that such an attack will significantly affect a neighborhood-based algorithm. Consider, for example, a user-based neighborhood algorithm in which the profiles are used to make predictions for Mary. None of the injected profiles will be close to the rating profile of Mary; therefore, Mary's predicted

Table 12.1: A naive attack: injecting fake user profiles with a single pushed item

Item ⇒ / **User ⇓**	1	2	3	4	5
John	1	2	1	6	7
Sayani	2	1	2	7	6
Mary	1	1	?	7	7
Alice	7	6	5	1	2
Bob	?	7	6	2	1
Carol	7	7	6	?	3
Fake-1	?	?	7	?	?
Fake-2	?	?	6	?	?
Fake-3	?	?	7	?	?
Fake-4	?	?	6	?	?
Fake-5	?	?	7	?	?

Table 12.2: Slightly better than naive attack: injecting fake user profiles with a single pushed item and random ratings on other items

Item ⇒ / **User ⇓**	1	2	3	4	5
John	1	2	1	6	7
Sayani	2	1	2	7	6
Mary	1	1	?	7	7
Alice	7	6	5	1	2
Bob	?	7	6	2	1
Carol	7	7	6	?	3
Fake-1	2	4	7	6	1
Fake-2	7	2	6	1	5
Fake-3	2	1	7	6	7
Fake-4	1	7	6	2	4
Fake-5	3	5	7	7	4

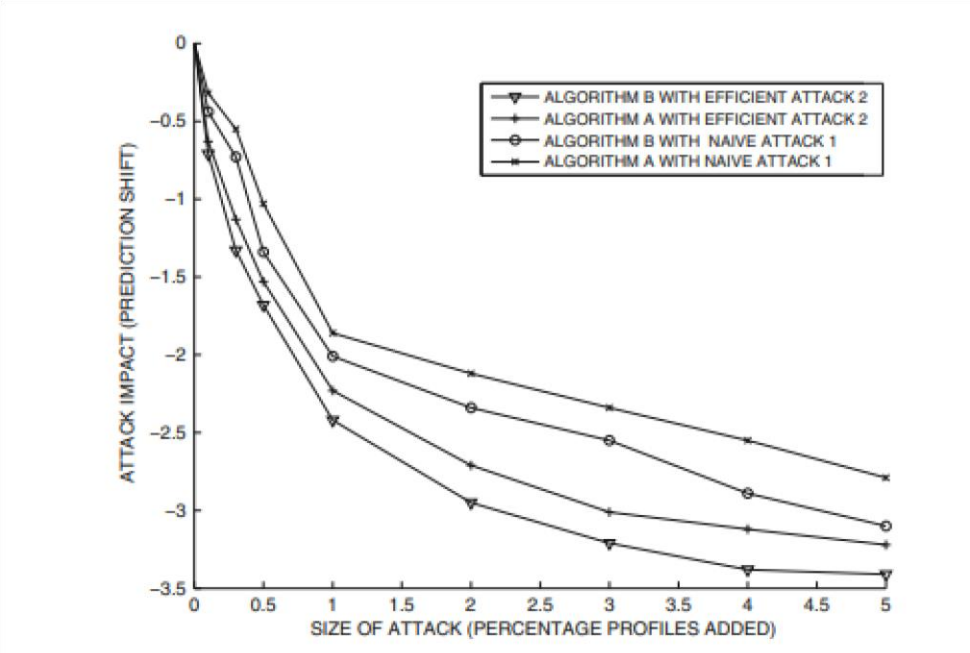
ways be available in practical settings. It is noteworthy that the efficiency of a particular attack also depends on the particular algorithm being attacked. For example, user-based and item-based neighborhood algorithms have very different levels of propensity of being attacked. If an item-based algorithm were to be applied to the high-knowledge case then the predicted ratings of Mary for item 3

would not be affected very significantly. This is because the item-based algorithm uses the ratings of other users only in the item-item similarity computation. The fake profiles affect the similarity computation used to discover the most similar items to item 3;subsequently Mary's own ratings on these items are used to make the prediction. In order to change the most similar items to item 3, one must typically inject a large number of ratings, which makes the attack more detectable. Furthermore, it is much harder to change the predictions in a particular direction for the target item by changing its similar items; after all, Mary's own ratings on these items are used to make the prediction rather than those in the fake profiles. Algorithms that are less prone to attacks are referred to as robust algorithms. It is one of the goals of recommender systems to design algorithms that are more robust to attacks. The aforementioned examples lead us to the following observations:

1. Carefully designed attacks are able to affect the predictions with a small number of fake profile insertions. On the other hand, a carelessly injected attack may have no effect on the predicted ratings at all.
2. When more knowledge about the statistics of the ratings database is available, an attacker is able to make more efficient attacks. However, it is often difficult to obtain a significant amount of knowledge about the ratings database.

Quantifying Attack Impact

Consider a ratings matrix R with user set U and item set I. The first step is to select a subset UT $\subseteq$ U of test users. Furthermore, let IT $\subseteq$ I be the set of test items pushed in the testing process. Then, the attack is performed one at a time for each item j $\in$ IT , and the effect on predicted rating of users in UT on item j is measured. The average prediction shift over all users and items is measured. Therefore, the attack needs to be performed |IT | times in order to measure the prediction shit over all test items



Types of Attacks

Although the rating of a particular item may be targeted in an attack, it is important to inject ratings of other items in order to make the attack effective. If fake profiles of only a single (pushed or nuked) item are inserted, they generally do not significantly affect the outcome of many recommendation algorithms. Furthermore, such attacks are generally easy to detect using automated methods. Therefore, the ratings of additional items are included in the injected profile. Such items are referred to as filler items filler items is particularly emphasized by the example

The ratings of most of the items will not be specified in the fake user profiles, just as in the case of genuine user profiles. For example, if a target movie, such as Gladiator, is rated frequently with another movie, such as Nero, then it is generally beneficial to add filler ratings of Nero when trying either to use either a push attack or a nuke attack on Gladiator. It would not be quite as beneficial to add filler ratings for a completely unrelated item like Shrek. However, such attacks require a greater knowledge of the ratings distribution because correlated sets of items need to be identified. Therefore, there is a natural trade-off between the efficiency and knowledge requirements of different types of attacks

Some attacks are specifically designed to be push attacks or to be nuke attacks. Although many attacks can be used in both capacities, each attack is generally more effective in one of the two settings. There are also subtle differences in the evaluation of these two types of attacks. The two types of attacks often show very different behavior in terms of the prediction shift and the hit-ratio. For example, it is much easier to nuke an item with a few bad ratings, given that only a few top items are recommended in any given setting. In other words, the effects on the hit-ratio can be more drastic than the effects on the prediction shift in the case of a nuke attack. Therefore, it is important to use multiple measures while evaluating push attacks and nuke attacks.

Random Attack

In the random attack, the filler items are assigned their ratings from a probability distribution that is distributed around the global mean of all ratings across all items. Because the global mean is used, the ratings of the various filler items are drawn from the same probability distribution. The filler items are chosen randomly from the database, and therefore the selection of items to rate is also not dependent on the target item. However, in some cases, the same set of filler items may be used for each profile. There is no advantage in choosing the same set of filler items for each profile because it does not reduce the required level of knowledge to mount the attack, but it only makes the attack more conspicuous

The target item is either set to the maximum possible rating value r_{max} or the minimum possible rating value r_{min}, depending on whether it is a push attack or a nuke attack. The main knowledge required to mount this attack is the mean values of all ratings. It is not very difficult to determine the global mean of the ratings in most settings. The limited knowledge required for a random attack comes at a disadvantage for the attacker, because such attacks are often not very efficient.

Average Attack

The average attack is similar to the random attack in terms of how the filler items are selected for rating. The same set of filler items are selected for each profile. However, the average attack differs from the random attack in terms of how the ratings are assigned to the selected items. In the average attack, the ratings that are assigned to the filler items have values that are specified at or approximately at the average of the specific item. The target item is assigned either the maximum rating or the minimum rating depending on whether the attack is a push attack or a nuke attack. Note that the average attack requires a greater amount of knowledge than the random attack, because knowing the global mean is not sufficient. One also needs to know the mean of each filler item. Furthermore, the attack is somewhat conspicuous because the same set of filler items is used for each fake profile.

In order to reduce the possibility of detection, one can also use randomly selected filler items for each injected user profile. The drawback of doing this is that a greater amount of knowledge will be required for making the attack. For example, the global mean of each of the injected filler items will be required. However, this can sometimes be reasonable in settings where the ratings are public. For example, the ratings on Amazon.com are public, and the average values can be computed easily. In other systems, such as IMDb, the average rating of each item is often directly advertised. Alternatively, one can randomly select items out of a small set of candidate items in order to determine the fillers for each fake profile. Such a strategy requires much less knowledge

Bandwagon Attack

The main problem with many of the aforementioned attacks is the inherent sparsity of the ratings matrix, which prevents the injected profiles from being sufficiently similar to the existing profiles. When too many items are selected as filler items, the attack becomes conspicuous. On the other hand, when a small number of filler items are selected randomly for a fake profile, then it might not have an inadequate number of observed ratings in common with other users. In user-based collaborative filtering, a fake profile has no impact when it does not have any rated items in common with the target user to whom the recommendation is being made. The efficiency of the attack reduces as a result**.**

The basic idea of the bandwagon attack is to leverage the fact that a small number of items are very popular in terms of the number of ratings they receive. For example, a blockbuster movie or a widely used textbook might receive many ratings. Therefore, if these items are always rated in the fake user profile, it increases the chance of a fake user profile being similar to the target user. In such cases, the predicted ratings of the target user are more likely to be affected by the attack. Therefore, the knowledge about the popularity of the items is used to improve the efficiency of the attack. In addition to the popular items, a set of random items is used as additional filler items. It is noteworthy that the notion of "popular" items in this particular case does not necessarily refer to the most frequently rated items, but rather refers to widely liked items. Such items are likely to be frequently rated in a positive way in the ratings database. One does not need to use the ratings matrix in order to determine the most popular items. It is usually easy to determine the most popular products of any type from sources independent of the ratings matrix. This is the main reason that

the bandwagon attack requires much less knowledge as compared to the average attack. Bandwagon attacks can often perform almost as well as the average attack, in spite of their smaller knowledge requirements. In general, bandwagon attacks can influence user-based collaborative filtering algorithms significantly, but they have greater difficulty in influencing item-based algorithms.

Popular Attack

The popular attack shares a number of similarities with the bandwagon attack in that it also uses popular items in order to create the filler items. However, the popular items might be either widely liked or widely disliked items, but they must have many ratings. The popular attack also assumes more knowledge about the ratings database to set the ratings of these popular items. Furthermore, it does not assume the existence of an additional set of filler items. Therefore, more popular items have to be used in this attack than in the case of the bandwagon attack.

In order to set the ratings on the popular items in an intelligent way, more knowledge needs to be assumed about the underlying rating database. In particular, it is assumed that average values of the ratings of the popular items are known. In order to achieve a push attack, the ratings of the various filler items in a fake user profile are set as follows:

1. If the average rating of a filler item in the ratings matrix is less than the global rating average over all items, then the rating of that item is set to its minimum possible value r_{min}.

2. If the rating of a filler item is greater than the overall average rating of all items, then the rating of the item is set to r_{min+1}.
3. The rating of the target item is always set to r_{max} in the fake user profile.

Love/Hate Attack

The love/hate attack is specifically designed to be a nuke attack, and its main advantage is that it requires very little knowledge to mount this attack. In the love/hate attack, the nuked item is set to the minimum rating value r_{min}, whereas the other items are set to the maximum rating value r_{max}. In spite of the minimal knowledge requirements, this attack is very effective. As discussed earlier, nuke attacks are generally easier to mount than push attacks. Therefore, such low-knowledge attacks often have a better chance of being successful in the case of nuke attacks as compared to push attacks. For example, a symmetrically designed attack in which the ratings of the filler items are set to r_{min} and the rating of the target item is set to r_{max}, is not quite as successful for pushing items. The love/hate attack is highly specific to user-based collaborative filtering algorithms, and it is almost completely ineffective with itembased collaborative filtering algorithms.

Reverse Bandwagon Attack

This attack is specifically designed to nuke items. The reverse bandwagon attack is a variation of the bandwagon attack, in which widely disliked items are used as filler items to mount the attack. The fact that such items are “widely disliked” means that they have received many ratings. For example, if a movie is highly promoted before its release but then turns out to be a box-office failure, then it will receive many low ratings. These items are selected as filler items. Such filler items are assigned low ratings together with the

nuked item. As in the case of the bandwagon attack, it is often not very difficult to discover such items from other channels. This attack works very well as a nuke attack when an itembased collaborative filtering algorithm is used for recommendation. Although it can also be used in the case of user-based collaborative filtering algorithms, many other attack methods, such as the average attack, are generally more effective.

Probe Attack

An important aspect of many of the aforementioned methods is that the ratings are often artificially set to values, such as r_{min} and r_{min+1}, in an identical way across many profiles. The use of such ratings tends to make the attack rather conspicuous, and therefore easily detectable. The probe attack tries to obtain more realistic ratings for items directly from a user-based recommender system in order to use these values in the attack. In other words, the operation of a recommender system is probed to mount the attack.

In the probe attack, a seed profile is created by the attacker, and the predictions generated by the recommender system are used to learn related items and their ratings. Since these recommendations have been generated by user-neighbors of this seed profile, they are highly likely to be correlated with the seed profile. One can also use this approach to learn the ratings of items within a specific genre. For example, in a movie recommendation scenario, consider the case where the target item to be pushed or nuked corresponds to action movies. The seed profile might contain the ratings of a set of popular action movies. The seed profile can then be extended further by observing the operation of a user-based collaborative filtering algorithm when the seed profile is used as the target user. The recommended items and their predicted ratings can be used to augment the seed profile in a realistic way. The rating of the target item is set to r_{max} or r_{min}, depending on whether it is pushed or nuked, respectively. The ratings of other filler items learned from the probing approach are set to the average values predicted by the recommender system.

Segment Attack

Almost all the aforementioned attack methods work effectively with user-based collaborative filtering algorithms, but they do not work quite as effectively with item-based algorithms. The only exception is the reverse bandwagon attack, which is designed only to nuke items, but not to push items. It is generally harder to attack item-based collaborative filtering algorithms. One of the reasons for this phenomenon is that an item-based algorithm leverages the target user's own ratings in order to make attacks. The target user is always an authentic user. Obviously, one cannot use fake profile injection to manipulate a genuine user's specified ratings.

Detecting Attacks on Recommender Systems

An adversarial relationship exists between attackers and the designers of recommender systems. From the point of view of maintaining a robust recommender system, the best way to thwart attacks is to detect them. Detection allows corrective measures (such as the removal of fake user profiles) to be taken. Accordingly, the detection of fake user profiles is a pivotal element in the design of a robust recommender system. However, the removal of fake profiles is a mistake-prone process, in which genuine profiles might be removed. It is important not to make too many mistakes, because the removal of authentic profiles can be counter-productive

Different attack algorithms have been designed for each case. Furthermore, attack detection algorithms may be either supervised or unsupervised. The difference between these two types of detection algorithms is as follows:

1. **Unsupervised attack detection algorithms:** In this case, ad hoc rules are used to detect fake profiles. For example, if a profile (or significant portion of it) is identical to many other profiles, then it is likely that all these profiles have been injected for the purpose of creating an attack. The basic idea in this class of algorithms is to identify the key characteristics of attack profiles that are not similar to genuine profiles. Such characteristics can be used to design unsupervised heuristics for fake profile detection.
2. **Supervised attack detection algorithms:** Supervised attack detection algorithms use classification models to detect attacks. Individual user profiles or groups of user profiles are characterized as multidimensional feature vectors. In many cases, these multidimensional feature vectors are derived using the same characteristics that are leveraged for the unsupervised case. For example, the number of profiles to which a given user profile is identical can be used as a feature for that user profile. Multiple features can be extracted corresponding to various characteristics of different types of attacks. A binary classifier can then be trained in which known attack profiles are labeled as +1, and the remaining profiles are labeled as −1. The trained classifier is used to predict the likelihood that a given profile is genuine.

Supervised attack detection algorithms are generally more effective than unsupervised methods because of their ability to learn from the underlying data. On the other hand, it is often difficult to obtain examples of attack profiles.

Attack detection methods are either individual profile detection methods or group profile detection methods. When detecting individual attack profiles, each user profile is assessed independently to determine whether or not it might be an attack. In the case of group detection, a set of profiles is assessed as a group. Note that both the unsupervised and supervised methods can be applied to either individual or group profile detection. In the following, we will discuss various methods for detecting attack profiles as individuals, and for detecting attack profiles as groups.

Individual Attack Profile Detection

Individual attack-profile detection is also referred to as single attack-profile detection. An unsupervised method for individual attack-profile detection is discussed. In this technique, a set of features is extracted from each user profile. The features are such that unusually high or unusually low values are indicative of an attack, depending on the feature at hand. In many cases, these features measure the consistency of a particular profile with other profiles in the system. Therefore, the fraction of features that take on abnormal values can be used as a measure to detect attacks. Other heuristic functions can also be used in conjunction with these features, which can enumerated as follows:

1. **Number of prediction differences (NPD):** For a given user, the NPD is defined as the number of prediction changes after removing that user from the system. Generally attack profiles tend to

have larger prediction differences than usual, because the attack profiles are designed to manipulate the system predictions in the first place.

2. **Degree of disagreement with other users (DD):** For the ratings matrix R = [rij]m×n, let νj be the mean rating of item j. Then, the degree to which the user i differs from other users on item j is given by |rij − νj |. This value is then averaged over all the |Ii| ratings observed for user i to obtain the degree of disagreement DD(i) of user i:

$$DD(i) = \frac{\sum_{j \in I_i} |r_{ij} - \nu_j|}{|I_i|}$$

Users with a larger degree of disagreement with other users are more likely to be attack profiles. This is because attack profiles tend to be different from the distribution of the other ratings.

3. **Rating deviation from mean agreement (RDMA):** The rating deviation from mean agreement is defined as the average absolute difference in the ratings from the mean rating of an item. The mean rating is biased with the inverse frequency if_j of each item j while computing the mean. The inverse frequency if_j is defined as the inverse of the number of users that have rated item j. Let the biased mean rating of an item j be $\nu^b{}_j$. Let Ii be the set of items rated by user i. Then, the value RDMA(i) for user i is defined as follows:

$$RDMA(i) = \frac{\sum_{j \in I_i} |r_{ij} - \nu_j^b| \cdot if_j}{|I_i|}$$

4. **Standard deviation in user ratings:** This is the standard deviation in the ratings of a particular user. If μ_i is the average rating of user i, and Ii is the set of items rated by that user, then the standard deviation σ_i is computed as follows:

$$\sigma_i = \frac{\sum_{j \in I_i} (r_{ij} - \mu_i)^2}{|I_i| - 1}$$

Even though the ratings of fake profiles differ significantly from other users, they are often quite selfsimilar because many filler items are set to the same rating value. As a result, the standard deviation σ_i tends to be small for fake profiles.

5. **Degree of similarity with top-k neighbors (SN):** In many cases, attack profiles are inserted in a coordination fashion, with the result being that the similarity of a user with her closest neighbors is increased. Therefore, if w_{ij} is the similarity between the users i and j, and N(i) is the set of neighbors of user i, then the degree of similarity SN(i) is defined as follows:

$$SN(i) = \frac{\sum_{j \in N(i)} w_{ij}}{|N(i)|}$$

It is noteworthy that most of these metrics, with the exception of RDMA, have also been proposed in the context of finding influential users in a recommender system. This coincidence is because fake profiles are designed by attackers to manipulate the predicted ratings as unusually influential entities in the recommender system. Furthermore, all these metrics, with the exception of the standard deviation, take on larger values in the case of an attack profile. The algorithm in declares a profile to be an attack when all these metrics take on abnormal values in the direction indicative

of an attack. Many variations of these basic principles are also possible for designing attack detection methods. Other features may be extracted as well. For example, the presence of an unusually large number of ratings in a profile may be considered suspicious.

The aforementioned features are useful not only for unsupervised attack detection algorithms, but also for supervised methods. The main difference between supervised and unsupervised methods is that examples of previous attacks are available. In such cases, these features are used to create a multidimensional representation and a classification model is constructed. For a given user profile for which the attack behavior is unknown, these features can be extracted. The classification model, which is built on the training data of example attacks, can be used on these features to assess the likelihood that it is indeed an attack. The generic features introduced in are as follows:

Weighted deviation from mean agreement (WDMA): The WDMA metric is similar to the RDMA metric, but it places greater weight on the ratings of rare items. Therefore, the square of the inverse frequency is used instead of the inverse frequency in the WDMA computation, the WDMA feature is computed as follows:

$$WDMA(i) = \frac{\sum_{j \in I_i} |r_{ij} - \nu_j| \cdot if_j^2}{|I_i|}$$

Weighted degree of agreement (WDA): The second variation of the RDMA metric uses only the numerator of the RDMA metric, defined by the right-hand side of Equation

$$WDA(i) = \sum_{j \in I_i} |r_{ij} - \nu_j| \cdot if_j$$

Modified degree of similarity: The modified degree of similarity is computed in a similar way to the degree of similarity defined by Equation. The main difference is that the similarity value w_{ij} in Equation is proportionally discounted by the number of users who rate both items i and j. This discounting is based on the intuition that the computed similarity is less reliable when the number of items in common between users i and j is small.

Group Attack Profile Detection

In these cases, the attack profiles are detected as groups rather than as individuals. The basic principle here is that the attacks are often based on groups of related profiles, which are very similar. Therefore, many of these methods use clustering strategies to detect attacks. Some of these methods perform the detection at recommendation time, whereas others use more conventional preprocessing strategies in which detection is performed a priori, with the fake profiles are removed up front.

Preprocessing Methods

The most common approach is to use clustering to remove fake profiles. Because of the way in which attack profiles are designed, authentic profiles and fake profiles create separate clusters. This is because many of the ratings in fake profiles are identical, and are therefore more likely to create tight clusters. In fact, the relative tightness of the clusters containing the fake profiles is one way of detecting them. The method proposed in uses PLSA to perform the clustering of the user profiles. Note that PLSA already creates a soft clustering, in which each user profile has a particular probability of belonging to an aspect. This soft clustering is converted to a hard clustering by assigning each user profile to the cluster with which it has the largest probability of membership. Although the PLSA approach is used for clustering in this case, virtually any clustering algorithm can be used in principle. After the hard clusters have been identified, the average Mahalanobis radius of each cluster is computed. The cluster with the smallest Mahalanobis radius is assumed to contain fake users. This approach is based on the assumption of the relative tightness of the clusters containing fake profiles. Such an approach works well for relatively overt attacks, but not necessarily for subtle attacks.

A simpler approach uses only principal component analysis (PCA). The basic idea is that the covariance between fake users is large. On the other hand, fake users often exhibit very low co- variances with other users, when the users are treated as dimensions. How can one identify such highly inter-correlated dimensions with PCA, which are not correlated with the normal users? This problem is related to that of variable selection in PCA. Let us examine the transpose of the ratings matrix in which users are treated as dimensions. According to the theory of variable selection in principal component analysis, this problem amounts to that of finding the dimensions (users in the transposed ratings matrix)with small coefficients in the small eigenvectors. Such dimensions (users) are likely to be fake profiles.

The ratings matrix is first normalized to zero mean and unit standard deviation and then the covariance matrix of its transpose is computed. The smallest eigenvector of this matrix is computed. Those dimensions (users) with small contributions (coefficients) in the eigenvector are selected. A slightly more enhanced approach is discussed in. In this case, the top (smallest) 3 to 5 eigenvectors are identified instead of using only the smallest Eigen vector. The sum of the contributions over these 3 to 5 eigenvectors is used in order to determine the spam users.

Online Methods

In these methods, the fake profiles are detected during recommendation time. Consider a scenario in which a user-based neighborhood algorithm is used during recommendation time. The basic idea is to create two clusters from the neighborhood of the active user. Note that the main goal of the attacker is to either push or nuke a particular item. Therefore, if a sufficiently large difference exists in the average ratings of the active items in the two clusters, it is assumed that an attack has taken place. The cluster in which the active item has the smaller variance of ratings is assumed to be the attack cluster. All profiles in this attack cluster are removed. This detection method has the merit of being able to be directly integrated into attack-resistant recommendation algorithms during the process of neighborhood formation. Therefore, this approach is not just a method to remove fake profiles, but also an online method for providing more robust recommendations. If desired, the fake profiles can be removed incrementally during the operation of the system.

Strategies for Robust Recommender Design

A variety of strategies are available for building recommenders in a more robust way. These strategies range from the use of better recommender-system design to better algorithmic design. In the following sections, we will discuss the use of some of these strategies.

Preventing Automated Attacks with CAPTCHAs

It is noteworthy that it requires a significant number of fake profiles in order to result in a significant shift in the predicted ratings. It is not uncommon for the adversary to require between 3% to 5% of the number of authentic profiles to be fake profiles to initiate an attack. For example, consider a ratings matrix containing over a million authentic users.

In such a case, as many as 50, 000 fake profiles may be required. It is hard to insert so many fake profiles manually. Therefore, attackers often resort to automated systems to interact with the Web interface of the rating system, and insert the fake profiles.

How can one detect such automated attacks? CAPTCHAs are designed to tell the difference between humans and machines in the context of Web interaction. The acronym CAPTCHA stands for **"Completely Automated Public Turing test to tell Computers and Humans Apart."** The basic idea is to present a human with distorted text, which is hard for a machine to decipher but can still be read by a human. This distorted text serves as a "challenge" text or word that needs to be entered into the Web interface in order to allow further interaction. An example of a CAPTCHA is illustrated in Figure The recommender system can prompt for CAPTCHAs to allow the entry of ratings, especially when a large number of them are entered from the same IP address.

Using Social Trust

The previous chapter reviewed methods for using social trust in the context of a recommender system. In these methods, the social trust between participants is used to influence the ratings. For example, users may specify trust relationships based on their experience with the ratings of other users. These trust relationships are then used to make more robust recommendations. Such methods are able to reduce the effectiveness of attacks, because users are unlikely to specify trust relationships towards fake profiles, which are rather contrived.

A global measure of the reputation of each user is used in the recommendation process. Each user is weighted with her reputation score while making the recommendation. The reputation is itself learned on the basis of the accuracy of a user predicting the rating of her neighbors. A theoretical bound on the impact of a negative attack was also shown by this work.

Designing Robust Recommendation Algorithms

It is evident from the discussion in this chapter that different algorithms have different levels of susceptibility to attacks. For example, user-based algorithms are generally much easier to attack than item-based algorithms. Therefore, a number of algorithms have specifically been designed with attack resistance in mind. This section will discuss some of these algorithms.

Incorporating Clustering in Neighborhood Methods

how clustering can be used in the context of neighborhood based methods. This work clusters the user profiles with the use of PLSA and k-means techniques. An aggregate profile is created from each cluster. The aggregate profile is based on the average rating of each item in the segment. Then, a similar approach to user-based collaborative filtering is used, except that the aggregate (clustered) profiles are used instead of the individual profiles. For each prediction, the closest aggregated profiles to the target users are used to make recommendations. It was shown in that the clustering-based approach provides significantly more robust results than a vanilla nearest-neighbor method. The main reason for the robustness of this approach is that the clustering process generally maps all the profiles to a single cluster, and therefore limits its influence on the prediction when alternative clusters are available.

Fake Profile Detection during Recommendation Time

The attack detection algorithms discussed in the earlier sections can also be used to make robust recommendations, particularly when the detection is done during recommendation time. Such a method is discussed in section. In this approach, the neighborhood of the active user is partitioned into two users. An attack is suspected when the active item has very different average values in the two clusters. The cluster that is the most self-similar (i.e., smaller radius) is considered the attack-cluster. The profiles from this cluster are removed. The recommendations are then performed using the profiles from the remaining cluster. This approach has the dual purpose of being both an attack-detection method and a robust recommendation algorithm.

Association-Based Algorithms

Rule-based collaborative filtering algorithms are such algorithms are robust to the average attack when the maximum attack size is less than 15%. The reason for this phenomenon is that there is generally not sufficient support for the attack profiles in order to mount a successful attack. However, such an algorithm is not immune to the segment attack.

Robust Matrix Factorization

Matrix factorization methods are generally more robust to attacks because of their natural ability to treat the attack profiles as noise. It has been shown how PLSA methods can be used to detect and remove attacks. Note that many matrix factorization recommenders are themselves based on PLSA. Therefore, if the attack profiles are removed in the intermediate step and the probabilistic parameters are renormalized, they can directly be used for recommendation.

Another approach is to modify the optimization function used for matrix factorization to make it more robust to attacks. In matrix factorization, the $m \times n$ ratings matrix

R is factorized into user factors and item factors as follows:

$$R \approx UV^T$$

Here $U = [u_{is}]$ and $V = [v_{js}]$ are $m \times k$ and $n \times k$ matrices. The predicted value $\hat{r}_{ij}$ of an entry is as follows:

$$\hat{r}_{ij} = \sum_{s=1}^{k} u_{is} v_{js} \tag{12.12}$$

Therefore, the error of predicting an observed entry is given by $e_{ij} = r_{ij} - \hat{r}_{ij}$. As discussed in Chapter 3, the matrix entries of U and V are determined by minimizing the sum of squares of e_{ij} over all the *observed* entries in the matrix R, along with some regularization terms.

How can one change the objective function to de-emphasize the contribution of attacking profiles? The main insight here is attacking profiles often cause outlier entries with large absolute values $|e_{ij}|$ in the *residual matrix* $(R - UV^T)$. Therefore, if one simply used the Frobenius norm of the observed portion of $(R - UV^T)$, the presence of fake profiles would significantly change the user factors and item factors. The natural solution is to de-emphasize the contribution of entries in the residual matrix with large absolute values. Let S be the set of observed entries in the ratings matrix R. In other words, we have:

$$S = \{(i, j) : r_{ij} \text{ is observed}\} \tag{12.13}$$

As discussed in Chapter 3, the objective function of matrix factorization is defined as follows:

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j) \in S} e_{ij}^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2$$

In order to de-emphasize the impact of very large absolute values of e_{ij}, a new set of error terms is defined:

$$\epsilon_{ij} = \begin{cases} e_{ij} & \text{if } |e_{ij}| \leq \Delta \\ f(|e_{ij}|) & \text{if } |e_{ij}| > \Delta \end{cases} \tag{12.14}$$

In order to de-emphasize the impact of very large absolute values of e_{ij}, a new set of error terms is defined:

$$\epsilon_{ij} = \begin{cases} e_{ij} & \text{if } |e_{ij}| \leq \Delta \\ f(|e_{ij}|) & \text{if } |e_{ij}| > \Delta \end{cases} \tag{12.14}$$

Here Δ is a user-defined threshold, which defines the case when an entry becomes large. $f(|e_{ij}|)$ is a *damped* (i.e., sublinear) function of $|e_{ij}|$ satisfying $f(\Delta) = \Delta$. This condition ensures that ϵ_{ij} is a continuous function of e_{ij} at $e_{ij} = \pm\Delta$. The damping ensures that large values of the error are not given undue importance. An example of such a damped function is as follows:

$$f(|e_{ij}|) = \sqrt{\Delta(2|e_{ij}| - \Delta)} \tag{12.15}$$

This type of damped function has been used in [428]. The objective function for robust matrix factorization then replaces the error values e_{ij} with the adjusted values ϵ_{ij} as follows:

$$\text{Minimize } J^{robust} = \frac{1}{2} \sum_{(i,j) \in S} \epsilon_{ij}^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2$$

An iterative re-weighted least-squares algorithm, which is described in [426], is used for the optimization process. Here, we describe a simplified algorithm. The first step is to compute the gradient of the objective function J^{robust} with respect to each of the decision variables:

$$\frac{\partial J^{robust}}{\partial u_{iq}} = \frac{1}{2} \sum_{j:(i,j) \in S} \frac{\partial \epsilon_{ij}^2}{\partial u_{iq}} + \lambda u_{iq}, \quad \forall i \in \{1 \ldots m\}, \forall q \in \{1 \ldots k\}$$

$$\frac{\partial J^{robust}}{\partial v_{jq}} = \frac{1}{2} \sum_{i:(i,j) \in S} \frac{\partial \epsilon_{ij}^2}{\partial v_{jq}} + \lambda v_{jq} \quad \forall j \in \{1 \ldots n\}, \forall q \in \{1 \ldots k\}$$

Note that the aforementioned gradients contain a number of partial derivatives with respect to the decision variables. The value of $\frac{\partial e_{ij}^2}{\partial u_{iq}}$ can be computed as follows:

$$\frac{\partial e_{ij}^2}{\partial u_{iq}} = \begin{cases} 2 \cdot e_{ij}(-v_{jq}) & \text{if } |e_{ij}| \leq \Delta \\ 2 \cdot \Delta \cdot \text{sign}(e_{ij})(-v_{jq}) & \text{if } |e_{ij}| > \Delta \end{cases}$$

Here, the sign function takes on the value of $+1$ for positive quantities and -1 for negative quantities. The case-wise description of derivative can be consolidated to simplified form as follows:

$$\frac{\partial e_{ij}^2}{\partial u_{iq}} = 2 \cdot \min\{|e_{ij}|, \Delta\} \cdot \text{sign}(e_{ij}) \cdot (-v_{jq})$$

It is noteworthy that the gradient is damped when the error is larger than Δ. This damping of the gradient directly makes the approach more robust to a few large errors in the ratings matrix. Similarly, we can compute the partial derivative with respect to v_{jq} as follows:

$$\frac{\partial e_{ij}^2}{\partial v_{jq}} = \begin{cases} 2 \cdot e_{ij}(-u_{iq}) & \text{if } |e_{ij}| \leq \Delta \\ 2 \cdot \Delta \cdot \text{sign}(e_{ij})(-u_{iq}) & \text{if } |e_{ij}| > \Delta \end{cases}$$

As before, it is possible to consolidate this derivative as follows:

$$\frac{\partial e_{ij}^2}{\partial v_{jq}} = 2 \cdot \min\{|e_{ij}|, \Delta\} \cdot \text{sign}(e_{ij}) \cdot (-u_{iq})$$

One can now derive the update steps as follows, which need to be executed for each user i and each item j:

One can now derive the update steps as follows, which need to be executed for each user i and each item j:

$$u_{iq} \Leftarrow u_{iq} + \alpha \left(\sum_{j:(i,j) \in S} \min\{|e_{ij}|, \Delta\} \cdot \text{sign}(e_{ij}) \cdot v_{jq} - \lambda \cdot u_{iq} \right) \quad \forall i, \quad \forall q \in \{1 \ldots k\}$$

$$v_{jq} \Leftarrow v_{jq} + \alpha \left(\sum_{i:(i,j) \in S} \min\{|e_{ij}|, \Delta\} \cdot \text{sign}(e_{ij}) \cdot u_{iq} - \lambda \cdot v_{jq} \right) \quad \forall j, \quad \forall q \in \{1 \ldots k\}$$

These updates are performed to convergence. The aforementioned steps correspond to global updates. These updates can be executed within the algorithmic framework of gradient descent (cf. Figure 3.8 of Chapter 3).

One can also isolate the gradients with respect to the errors in individual entries and process them in random order. Such an approach corresponds to *stochastic* gradient descent. For each observed entry $(i, j) \in S$, the following update steps are executed:

$$u_{iq} \Leftarrow u_{iq} + \alpha \left(\min\{|e_{ij}|, \Delta\} \cdot \text{sign}(e_{ij}) \cdot v_{jq} - \frac{\lambda \cdot u_{iq}}{n_i^{user}} \right) \quad \forall q \in \{1 \ldots k\}$$

$$v_{jq} \Leftarrow v_{jq} + \alpha \left(\min\{|e_{ij}|, \Delta\} \cdot \text{sign}(e_{ij}) \cdot u_{iq} - \frac{\lambda \cdot v_{jq}}{n_j^{item}} \right) \quad \forall q \in \{1 \ldots k\}$$

Here n_i^{user} denotes the number of observed ratings for user i and n_j^{item} denotes the number of observed ratings for item j. One cycles through the observed entries in the matrix in

Here n^{user}_i denotes the number of observed ratings for user i and n^{item}_j denotes the number of observed ratings for item j. One cycles through the observed entries in the matrix in random order and performs the aforementioned update steps until convergence is reached. This is based on the framework with the modified set of update steps discussed above. These update steps are different from traditional matrix factorization only in terms of capping the absolute values of the gradient components, when the error is larger than Δ. This is consistent with the stated goals of a robust matrix factorization approach, where large errors might be the result of anomalies in the ratings matrix structure. These anomalies might be indicative of attacks.

It is important to note that this approach will work only when the number of attack profiles is small compared to the correct entries in the ratings matrix. On the other hand, if the number of attack profiles is very large, it will significant affect the factor matrices, and the damping approach will not work. Robust matrix factorization and PCA has a rich history in the context of the recovery of the structure of corrupted matrices. Refer to the bibliographic notes for pointers to work in this area.

UNIT V EVALUATING RECOMMENDER SYSTEMS

The evaluation of collaborative filtering shares a number of similarities with that of classification. This similarity is due to the fact that collaborative filtering can be viewed as a generalization of the classification and regression modeling problem (cf. section 1.3.1.3 of Chapter 1). Nevertheless, there are many aspects to the evaluation process that are unique to collaborative filtering applications. The evaluation of content-based methods is even more similar to that of classification and regression modeling, because content-based methods often use text classification methods under the covers.

This chapter will introduce various mechanisms for evaluating various recommendation algorithms and also relate these techniques to the analogous methods used in classification and regression modeling. A proper design of the evaluation system is crucial in order to obtain an understanding of the effectiveness of various recommendation algorithms. As we will see later in this chapter, the evaluation of recommender systems is often multifaceted, and a single criterion cannot capture many of the goals of the designer. An incorrect design of the experimental evaluation can lead to either gross underestimation or overestimation of the true accuracy of a particular algorithm or model.

When working with offline methods, accuracy measures can often provide an incomplete picture of the true conversion rate of a recommender system. Several other secondary measures also play a role. Therefore, it is important to design the evaluation system carefully so that the measured metrics truly reflect the effectiveness of the system from the user perspective. In particular, the following issues are important from the perspective of designing evaluation methods for recommender systems:

1. Evaluation goals: While it is tempting to use accuracy metrics for evaluating recommender systems, such an approach can often provide an incomplete picture of the user experience. Although accuracy metrics are arguably the most important components of the evaluation, many secondary goals such as novelty, trust, coverage, and serendipity are important to the user experience. This is because these metrics have important short and long-term impacts on the conversion rates. Nevertheless, the actual quantification of some of these factors is often quite subjective, and there are often no hard measures to provide a numerical metric.
2. Experimental design issues: Even when accuracy is used as the metric, it is crucial to design the experiments so that the accuracy is not overestimated or underestimated. For example, if the same set of specified ratings is used both for model construction and for accuracy evaluation, then the accuracy will be grossly overestimated. In this context, careful experimental design is important.
3. Accuracy metrics: In spite of the importance of other secondary measures, accuracy metrics continue to be the single most important component in the evaluation. Recommender systems can be evaluated either in terms of the prediction accuracy of a rating or the accuracy of ranking the items. Therefore, a number of common metrics such as the mean absolute error and mean squared error are used frequently. The evaluation of rankings can be performed with the use of various methods, such as utility-based

computations, rank-correlation coefficients, and the receiver operating characteristic curve.

Evaluation Paradigms:

There are three primary types of evaluation of recommender systems, corresponding to user studies, online evaluations, and offline evaluations with historical data sets. The first two types involve users, although they are conducted in slightly different ways. The main differences between the first two settings lie in how the users are recruited for the studies. Although online evaluations provide useful insights about the true effects of a recommendation algorithm, there are often significant practical impediments in their deployment. In the following, an overview of these different types of evaluation is provided

User Studies

In user studies, test subjects are actively recruited, and they are asked to interact with the recommender system to perform specific tasks. Feedback can be collected from the user before and after the interaction, and the system also collects information about their interaction with the recommender system. These data are then used to make inferences about the likes or dislikes of the user. For example, users could be asked to interact with the recommendations at a product site and give their feedback about the quality of the recommendations. Such an approach could then be used to judge the effectiveness of the underlying algorithms. Alternatively, users could be asked to listen to several songs, and then provide their feedback on these songs in the form of ratings. An important advantage of user studies is that they allow for the collection of information about the user interaction with the system. Various scenarios can be tested about the effect of changing the recommender system on the user interaction, such as the effect of changing a particular algorithm or user-interface. On the other hand, the active awareness of the user about the testing of the recommender system can often bias her choices and actions. It is also difficult and expensive to recruit large cohorts of users for evaluation purposes. In many cases, the recruited users are not representative of the general population because the recruitment process is itself a bias-centric filter, which cannot be fully controlled. Not all users would be willing to participate in such a study, and those who do agree might have unrepresentative interests with respect to the remaining population. For example, in the case of the example of rating songs, the (voluntary) participants are likely to be music enthusiasts. Furthermore, the fact that users are actively aware of their recruitment for a particular study is likely to affect their responses. Therefore, the results from user evaluations cannot be fully trusted.

Online Evaluation

Online evaluations also leverage user studies except that the users are often real users in a fully deployed or commercial system. This approach is sometimes less susceptible to bias from the recruitment process, because the users are often directly using the system in the natural course of affairs. Such systems can often be used to evaluate the comparative performance of various algorithms. Typically, users can be sampled randomly, and the various algorithms can be tested with each sample of users. A typical example of a metric, which is used to measure the effectiveness of the recommender system on the users, is the conversion rate. The conversion rate measures the frequency with which a user selects a recommended item. For example, in a news recommender system, one might compute the fraction of times that a user selects a recommended article. If desired, expected costs or profits can be added to the items to make the measurement sensitive to the importance of the item. These methods are also referred to as A/B testing, and they measure the direct impact of the recommender system on the end user. The basic idea in these methods is to compare two algorithms as follows:

1. Segment the users into two groups A and B.

2. Use one algorithm for group A and another algorithm for group B for a period of time, while keeping all other conditions (e.g., selection process of users) across the two groups as similar as possible.

3. At the end of the process, compare the conversion rate (or other payoff metric) of the two groups. This approach is very similar to what is used for clinical trials in medicine. Such an approach is the most accurate one for testing the long-term performance of the system directly in terms of goals such as profit. These methods can also be leveraged for the user studies discussed in the previous section.

One observation is that it is not necessary to strictly segment the users into groups in cases where the payoff of each interaction between the user and the recommender can be measured separately. In such cases, the same user can be shown one of the algorithms at random, and the payoff from that specific interaction can be measured. Such methods of evaluating recommender systems have also been generalized to the development of more effective recommendation algorithms. The resulting algorithms are referred to as multi-arm bandit algorithms. The basic idea is similar to that of a gambler (recommender system) who is faced with a choice of selecting one of a set of slot machines (recommendation algorithms) at the casino. The gambler suspects that one of these machines has a better payoff (conversion rate) than others. Therefore, the gambler tries a slot machine at random 10% of the time in order to explore the relative payoffs of the machines. The gambler greedily selects the best paying slot machine the remaining 90% of the time in order to exploit the knowledge learned in the exploratory trials. The process of exploration and exploitation is fully interleaved in a random way. Furthermore, the gambler may choose to give greater weight to recent results as compared to older results for evaluation. This general approach is related to the notion of reinforcement learning, which can often be paired with online systems

The main disadvantage is that such systems cannot be realistically deployed unless a large number of users are already enrolled. Therefore, it is hard to use this method during the start up phase. Furthermore, such systems are usually not openly accessible, and they are only accessible to the owner of the specific commercial system at hand. Therefore, such tests can be performed only by the commercial entity, and for the limited number of scenarios handled by their system. This means that the tests are often not generalizable to system-independent benchmarking by scientists and practitioners. In many cases, it is desirable to test the robustness of a recommendation algorithm by stress-testing it under a variety of settings and data domains. By using multiple settings, one can obtain an idea of the generalizability of the system. Unfortunately, online methods are not designed for addressing such needs. A part of the problem is that one cannot fully control the actions of the test users in the evaluation process.

Offline Evaluation with Historical Data Sets

In offline testing, historical data, such as ratings, are used. In some cases, temporal information may also be associated with the ratings, such as the time-stamp at which each user has rated the item. A well known example of a historical data set is the Netflix Prize data set. This data set was originally released in the context of an online contest, and has since been used as a standardized benchmark for testing many algorithms. The main advantage of the use of historical data sets is that they do not require access to a large user base. Once a data set has been collected, it can be used as a standardized benchmark to compare various algorithms across a variety of settings. Furthermore, multiple data sets from various domains (e.g., music, movies, news) can be used to test the generalizability of the recommender system.

Offline methods are among the most popular techniques for testing recommendation algorithms, because standardized frameworks and evaluation measures have been developed for such cases. Therefore, much of this chapter will be devoted to the study of offline evaluation. The main disadvantage of offline evaluations is that they do not measure the actual propensity of the user to react to the recommender system in the future. For example, the data might evolve over time, and the current predictions may not reflect the most appropriate predictions for the future. Furthermore, measures such as accuracy do not capture important characteristics of recommendations, such as serendipity and novelty. Such recommendations have important longterm effects on the conversion rate of the recommendations. Nevertheless, in spite of these disadvantages, offline methods continue to be the most widely accepted techniques for recommender system evaluation. This is because of the statistically robust and easily understandable quantifications available through such testing methods.

General Goals of Evaluation Design

In this section, we will study some of the general goals in evaluating recommender systems. Aside from the well known goal of accuracy, other general goals include factors such as diversity, serendipity, novelty, robustness, and scalability. Some of these goals can be concretely quantified, whereas others are subjective goals based on user experience. In such cases, the only way of measuring such goals is through user surveys. In this section, we will study these different goals.

Accuracy

Accuracy is one of the most fundamental measures through which recommender systems are evaluated. In this section, we provide a brief introduction to this measure Therefore, the accuracy metrics are often similar to those used in regression modeling. Let R be the ratings matrix in which ruj is the known rating of user u for item j. Consider the case where a recommendation algorithm estimates this rating as $\hat{r}uj$. Then, the entry-specific error of the estimation is given by the quantity $euj = \hat{r}uj - ruj$. The overall error is computed by averaging the entry-specific errors either in terms of absolute values or in terms of squared values. Furthermore, many systems do not predict ratings; rather they only output rankings of top-k recommended items. This is particularly common in implicit feedback data sets. Different methods are used to evaluate the accuracy of ratings predictions and the accuracy of rankings.

1. **Designing the accuracy evaluation**: All the observed entries of a ratings matrix cannot be used both for training the model and for accuracy evaluation. Doing so would grossly overestimate the accuracy because of over fitting. It is important to use only a different set of entries for evaluation than was used for training. If S is the observed entries in the ratings matrix, then a small subset $E \subset S$ is used for evaluation, and the set $S - E$ is used for training. This issue is identical to that encountered in the evaluation of classification algorithms. After all, as discussed in earlier chapters, collaborative filtering is a direct generalization of the classification and regression modeling problem. Therefore, the standard methods that are used in classification and regression modeling, such as hold-out and cross-validation, are also used in the evaluation of recommendation algorithms.

2. **Accuracy metrics:** Accuracy metrics are used to evaluate either the prediction accuracy of estimating the ratings of specific user-item combinations or the accuracy of the top k ranking predicted by a recommender system. Typically, the ratings of a set E of entries in the ratings matrix are hidden, and the accuracy is evaluated over these hidden entries

3. **Accuracy of estimating rankings:** Many recommender systems do not directly estimate ratings; instead, they provide estimates of the underlying ranks. Depending on the nature of the ground-truth, one can use rank-correlation measures, utility-based measures, or the receiver operating characteristic. The latter two methods are designed for unary (implicit feedback) data sets

Coverage:

Even when a recommender system is highly accurate, it may often not be able to ever recommend a certain proportion of the items, or it may not be able to ever recommend to a certain proportion of the users. This measure is referred to as coverage. This limitation of recommender systems is an artifact of the fact that ratings matrices are sparse. For example, in a rating matrix contains a single entry for each row and each column, then no meaningful recommendations can be made by almost any algorithm. Nevertheless, different recommender systems have different levels of propensity in providing coverage. In practical settings, the systems often have 100% coverage because of the use of defaults for ratings that are not possible to predict. An example of such a default would be to report the average of all the ratings of a user for an item when the rating for a specific user-item combination cannot be predicted. Therefore, the trade-off between accuracy and coverage always needs to be incorporated into the evaluation process. There are two types of coverage, which are referred to as user-space coverage and itemspace coverage, respectively.

Confidence and Trust:

The estimation of ratings is an inexact process that can vary significantly with the specific training data at hand. Furthermore, the algorithmic methodology might also have a significant impact on the predicted ratings. This always leads to uncertainty in the user about the accuracy of the predictions. Many recommender systems may report ratings together with confidence estimates. For example, a confidence interval on the range of predicted ratings may be provided. In general, recommender systems that can accurately recommend smaller confidence intervals are more desirable because they bolster the user's trust in the system. For two algorithms that use the same method for reporting confidence, it is possible to measure how well the predicted error matches these confidence intervals. For example, if two recommender systems provide 95% confidence intervals for each rating, one can measure the absolute width of the intervals reported by the two algorithms. The algorithm with the smaller confidence interval width will win as long as both algorithms are correct (i.e., within the specified intervals) at least 95% of the time on the hidden ratings. If one of the algorithms falls below the required 95% accuracy, then it automatically loses. Unfortunately, if one system uses 95% confidence intervals and another uses 99% confidence intervals, it is not possible to meaningfully compare them.

Serendipity

The word "serendipity" literally means "lucky discovery." Therefore, serendipity is a measure of the level of surprise in successful recommendations. In other words, recommendations need to be unexpected. In contrast, novelty only requires that the user was not aware of the recommendation earlier. Serendipity is a stronger condition than novelty. All serendipitious recommendations are novel, but the converse is not always true. Consider the case where a particular user frequently eats at Indian restaurants. The recommendation of a new Pakistani restaurant to that user might be novel if that user has not eaten at that restaurant earlier. However, such a recommendation is not

serendipitious, because it is well known that Indian and Pakistani food are almost identical. On the other hand, if the recommender system suggests a new Ethiopian restaurant to the user, then such a recommendation is serendipitious because it is less obvious. Therefore, one way of viewing serendipity is as a departure from "obviousness.

There are several ways of measuring serendipity in recommender systems

1. **Online methods**: The recommender system collects user feedback both on the usefulness of a recommendation and its obviousness. The fraction of recommendations that are both useful and non-obvious, is used as a measure of the serendipity
2. **Offline methods:** One can also use a primitive recommender to generate the information about the obviousness of a recommendation in an automated way. The primitive recommender is typically selected as a content-based recommender, which has a high propensity for recommending obvious items. Then, the fraction of the recommended items in the top-k lists that are correct (i.e., high values of hidden ratings), and are also not recommended by the primitive recommender are determined. This fraction provides a measure of the serendipity.

Robustness and Stability

A recommender system is stable and robust when the recommendations are not significantly affected in the presence of attacks such as fake ratings or when the patterns in the data evolve significantly over time. In general, significant profit-driven motivations exist for some users to enter fake ratings. ant profit-driven motivations exist for some users to enter fake ratings. For example, the author or publisher of a book might enter fake positive ratings about a book at Amazon.com, or they might enter fake negative ratings about the books of a rival

Scalability

In recent years, it has become increasingly easy to collect large numbers of ratings and implicit feedback information from various users. In such cases, the sizes of the data sets continue to increase over time. As a result, it has become increasingly essential to design recommender systems that can perform effectively and efficiently in the presence of large amounts of data

1. Training time: Most recommender systems require a training phase, which is separate from the testing phase. For example, a neighborhood-based collaborative filtering algorithm might require pre-computation of the peer group of a user, and a matrix factorization system requires the determination of the latent factors. The overall time required to train a model is used as one of the measures. In most cases, the training is done offline. Therefore, as long as the training time is of the order of a few hours, it is quite acceptable in most real settings.

2. Prediction time: Once a model has been trained, it is used to determine the top recommendations for a particular customer. It is crucial for the prediction time to be low, because it determines the latency with which the user receives the responses.
3. Memory requirements: When the ratings matrices are large, it is sometimes a challenge to hold the entire matrix in the main memory. In such cases, it is essential to design the algorithm to minimize memory requirements. When the memory requirements become very high, it is difficult to use the systems in large-scale and practical settings.

Design Issues in Offline Recommender Evaluation

The discussions in this section and the next pertain to accuracy evaluation of offline and historical data sets. It is crucial to design recommender systems in such a way that the accuracy is not grossly overestimated or underestimated. For example, one cannot use the same set of specified ratings for both training and evaluation. Doing so would grossly overestimate the accuracy of the underlying algorithm. Therefore, only a part of the data is used for training, and the remainder is often used for testing. The ratings matrix is typically sampled in an entry-wise fashion. In other words, a subset of the entries are used for training, and the remaining entries are used for accuracy evaluation. Note that this approach is similar to that used for testing classification and regression modeling algorithms. The main difference is that classification and regression modeling methods sample rows of the labeled data, rather than sampling the entries. This difference is because the unspecified entries are always restricted to the class variable in classification, whereas any entry of the ratings matrix can be unspecified. The design of recommender evaluation systems is very similar to that of classifier evaluation systems because of the similarity between the recommendation and classification problems.

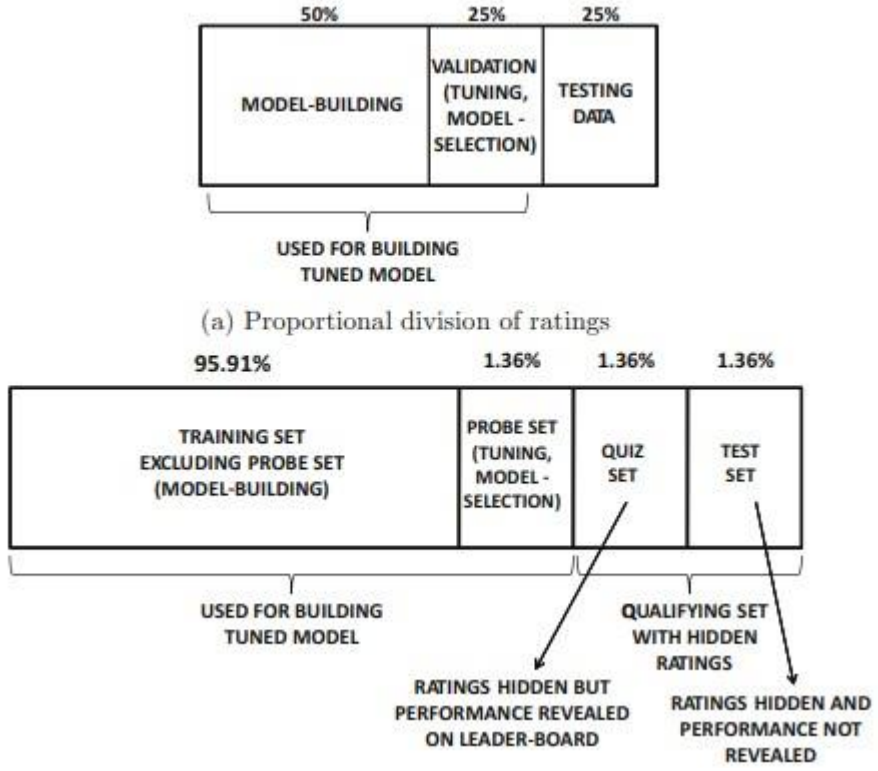
A common mistake made by analysts in the benchmarking of recommender systems is to use the same data for parameter tuning and for testing. Such an approach grossly overestimates the accuracy because parameter tuning is a part of training, and the use of test data in the training process leads to overfitting. To guard against this possibility, the data are often divided into three parts:

1. Training data: This part of the data is used to build the training model. For example, in a latent factor model, this part of the data is used to create the latent factors from the ratings matrix. One might even use these data to create multiple models in order to eventually select the model that works best for the data set at hand.
2. Validation data: This part of the data is used for model selection and parameter tuning. For example, the regularization parameters in a latent factor model may be determined by testing the accuracy over the validation data. In the event that multiple models have been built from the training data, the validation data are used to determine the accuracy of each model and select the best one.

3. Testing data: This part of the data is used to test the accuracy of the final (tuned) model. It is important that the testing data are not even looked at during the process of parameter tuning and model selection to prevent overfitting. The testing data are used only once at the very end of the process. Furthermore, if the analyst uses the results on the test data to adjust the model in some way, then the results will be contaminated with knowledge from the testing data.

Case Study of the Netflix Prize Data Set

A particularly instructive example of a well-known data set used in collaborative filtering is the Netflix Prize data set, because it demonstrates the extraordinary lengths to which Netflix went to prevent overfitting on the test set from the contest participants. In the Netflix data



set, the largest portion of the data set contained 95.91% of the ratings. This portion of the data set was typically used by the contest participants for model-building. Another 1.36% of the data set was revealed to the participants as a probe set. Therefore, the model-building portion of the data and the probe data together contained 95.91 + 1.36 = 97.27% of the data. The probe set was

typically used by contests for various forms of parameter tuning and model selection, and therefore it served a very similar purpose as a validation set. However, different contestants used the probe set in various ways, especially since the ratings in the probe set were more recent, and the statistical distribution of the ratings in the training and probe sets were slightly different. For the case of ensemble methods the probe set was often used to learn the weights of various ensemble components. The combined data set with revealed ratings (including the probe set) corresponds to the full training data, because it was used to build the final tuned model. An important peculiarity of the training data was that the distributions of the probe set and the model-building portion of the training set were not exactly identical, although the probe set reflected the statistical characteristics of the qualifying set with hidden ratings. The reason for this difference was that most of the ratings data were often quite old and they did not reflect the true distribution of the more recent or future ratings. The probe and qualifying sets were based on more recent ratings, compared to the 95.91% of the ratings in the first part of the training data.

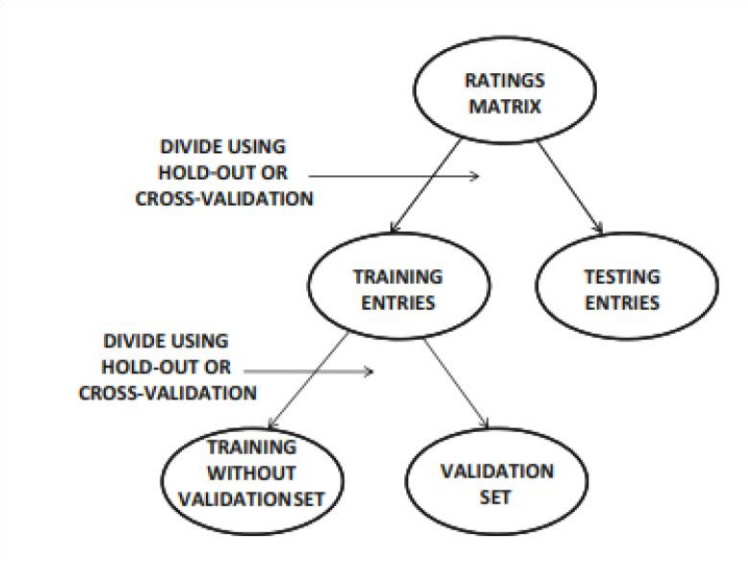
The ratings of the remaining 2.7% of the data were hidden, and only triplets of the form User, Movie, GradeDate were supplied without actual ratings. The main difference from a test set was that participants could submit their performance on the qualifying set to Netflix, and the performance on half the qualifying data, known as the quiz set, was revealed to the participants on a leader-board. Although revealing the performance on the quiz set to the participants was important in order to give them an idea of the quality of their results, the problem with doing so was that participants could use the knowledge of the performance of their algorithm on the leader-board to over-train their algorithm on the quiz set with repeated submissions. Clearly, doing so results in contamination of the results from knowledge of the performance on the quiz set, even when ratings are hidden. Therefore, the part of the qualifying set that was not in the quiz set was used as the test set, and the results on only this part of the qualifying set were used to determine the final performance for the purpose of prize determination. The performance on the quiz set had no bearing on the final contest, except to give the participants a continuous idea of their performance during the contest period. Furthermore, the participants were not informed about which part of the qualifying set was the quiz set. This arrangement ensured that a truly outof-sample data set was used to determine the final winners of the contest.

Segmenting the Ratings for Training and Testing

In practice, real data sets are not pre-partitioned into training, validation, and test data sets. Therefore, it is important to be able to divide the entries of a ratings matrix into these portions automatically. Most of the available division methods, such as hold-out and cross validation, are used to divide1 the data set into two portions instead of three. However, it is possible to obtain three portions as follows. By first dividing the rating entries into training and test portions, and then further segmenting the validation portion from the training data, it is possible to obtain the required three segments. Therefore, in the following, we will discuss the segmentation of the ratings matrix into training and testing portions of the entries using methods such as hold-out and cross-validation

Hold-Out

In the hold-out method, a fraction of the entries in the ratings matrix are hidden, and the remaining entries are used to build the training model. The accuracy of predicting the hidden entries is then reported as the overall accuracy. Such an approach ensures that the reported accuracy is not a result of overfitting to the specific data set, because the entries used for evaluation are hidden during training. Such an approach, however, underestimates the true accuracy. First, all entries are not used in training, and therefore the full power of the data is not used. Second, consider the case where the held-out entries have a higher average rating than the full ratings matrix. This means that the held-in entries have a loweraverage rating than the ratings matrix, and also the held-out entries. This will lead to a pessimistic bias in the evaluation.



Cross-Validation

In the cross-validation method, the ratings entries are divided into q equal sets. Therefore, if S is the set of specified entries in the ratings matrix R, then the size of each set, in terms of the number of entries, is $|S|/q$. One of the q segments is used for testing, and the remaining $(q - 1)$ segments are used for training. In other words, a total of $|S|/q$ entries are hidden during each such training process, and the accuracy is then evaluated over these entries. This process is repeated q times by using each of the q segments as the test set. The average accuracy over the q different test sets is reported. Note that this approach can closely estimate the true accuracy when the value of q is large. A special case is one where q is chosen to be equal to the number of specified entries in the ratings matrix. Therefore, $|S| - 1$ rating entries are used for training, and the one entry is used for testing. This approach is referred to as leave-one-out cross-validation. Although such an approach can closely approximate the accuracy, it is usually too expensive to train the model $|S|$ times. In practice, the value of q is fixed to a number such as 10. Nevertheless, leaveone-out cross

validation is not very difficult to implement for the specific case of neighborhoodbased collaborative filtering algorithms.

Limitations of Evaluation Measures

Accuracy-based evaluation measures have a number of weaknesses that arise out of selection bias in recommender systems. In particular, the missing entries in a ratings matrix are not random because users have the tendency of rating more popular items. a few items are rated by many users, whereas the vast majority of items may be found in the long tail. The distributions of the ratings on popular items are often different from those on items in the long tail. When an item is very popular, it is most likely because of the notable content in it. This factor will affect2 the rating of that item as well. As a result, the accuracy of most recommendation algorithms is different on the more popular items versus the items in the long tail More generally, the fact that a particular user has chosen not to rate a particular item thus far has a significant impact on what her rating would be if the user were forced to rate all items.

These factors cause problems of bias in the evaluation process. After all, in order to perform the evaluation on a given data set, one cannot use truly missing ratings; rather, one must simulate missing items with the use of hold-out or cross-validation mechanisms on ratings that are already specified. Therefore, the simulated missing items may not show similar accuracy to that one would theoretically obtain on the truly consumed items in the future. The items that are consumed in the future will not be randomly selected from the missing entries for the reasons discussed above. This property of rating distributions is also known as Missing Not At Random (MNAR), or selection bias. This property can lead to an incorrect relative evaluation of algorithms. For example, a popularity-based model in which items with the highest mean rating are recommended might do better in terms of gaining more revenue for the merchant than its evaluation on the basis of randomly missing ratings might suggest. This problem is aggravated by the fact that items in the long tail are especially important to the recommender system, because a disproportionate portion of the profits in such systems are realized through such items

There are several solutions to this issue. The simplest solution is to not select the missing ratings at random but to use a model for selecting the test ratings based on their likelihood of being rated in the future. Another solution is to not divide the ratings at random between training and test, but to divide them temporally by using more recent ratings as a part of the test data; indeed, the Netflix Prize contest used more recent ratings in the qualifying set, although some of the recent ratings were also provided as a part of the probe set. An approach that has been used in recent years, is to correct for this bias by modeling the bias in the missing rating distribution within the evaluation measures. Although such an approach has some merits, it does have the drawback that the evaluation process itself now assumes a model of how the ratings behave. Such an approach might inadvertently favor algorithms that use a model similar to that used for the prediction of ratings as for the evaluation process. It is noteworthy that many recent algorithms use implicit feedback within

the prediction process. This raises the possibility that a future prediction algorithm might be designed to be tailored to the model used for adjusting for the effect of user selection bias within the evaluation. Although the assumptions in which relate the missing ratings to their relevance, are quite reasonable, the addition of more assumptions (or complexity) to evaluation mechanisms increases the possibility of "gaming" during benchmarking. At the end of the day, it is important to realize that these limitations in collaborative filtering evaluation are inherent; the quality of any evaluation system is fundamentally limited by the quality of the available ground truth. In general, it has been shown through experiments on Netflix data

Avoiding Evaluation Gaming

The fact that missing ratings are not random can sometimes lead to unintended (or intended) gaming of the evaluations in settings where the user-item pairs of the test entries are specified. For example, in the Netflix Prize contest, the coordinates of the user-item pairs in the qualifying set were specified, although the values of the ratings were not specified. By incorporating the coordinates of the user-item pairs within the qualifying set as implicit feedback one can improve the quality of recommendations. It can be argued that such an algorithm would have an unfair advantage over one that did not include any information about the identities of rated items in the qualifying set. The reason is that in real-life settings, one would never have any information about future coordinates of rated items, as are easily available in the qualifying set of the Netflix Prize data. Therefore, the additional advantage of incorporating such implicit feedback would disappear in real-life settings. One solution would be to not specify the coordinates of test entries and thereby evaluate over all entries. However, if the ratings matrix has very large dimensions (e.g., 107×105), it may be impractical to perform the prediction over all entries. Furthermore, it would be difficult to store and upload such a large number of predictions in an online contest like the Netflix Prize. In such cases, an alternative would be to include (spurious) unrated entries within the test set. Such entries are not used for evaluation but they have the effect of preventing the use of coordinates of the test entries as implicit feedback.

Accuracy Metrics in Offline Evaluation

Offline evaluation can be performed by measuring the accuracy of predicting rating values (e.g., with RMSE) or by measuring the accuracy of ranking the recommended items. The logic for the latter set of measures is that recommender systems often provide ranked lists of items without explicitly predicting ratings. Ranking-based measures often focus on the accuracy of only the ranks of the top-k items rather than all the items. This is particularly true in the case of implicit feedback data sets. Even in the case of explicit ratings, the ranking-based evaluations provide a more realistic perspective of the true usefulness of the recommender system because the user only views the top-k items rather than all the items. However, for bench-marking, the accuracy of ratings predictions

is generally preferred because of its simplicity. In the Netflix Prize competition, the RMSE measure was used for final evaluation. In the following, both forms of accuracy evaluation will be discussed.

Measuring the Accuracy of Ratings Prediction

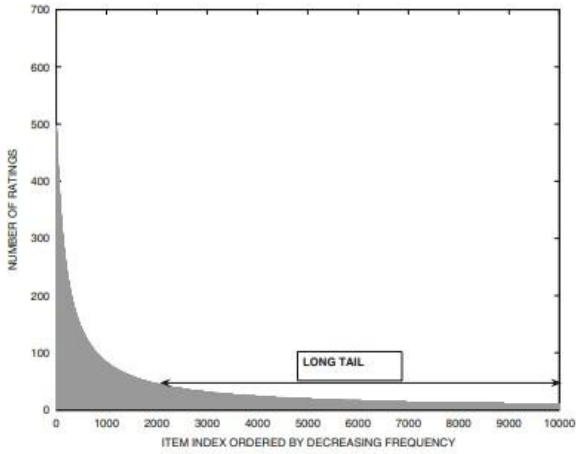
Once the evaluation design for an offline experiment has been finalized, the accuracy needs to be measured over the test set. As discussed earlier, let S be the set of specified (observed) entries, and $E \subset S$ be the set of entries in the test set used for evaluation. Each entry in E is a user-item index pair of the form (u, j) corresponding to a position in the ratings matrix. Note that the set E may correspond to the held out entries in the hold-out method, or it may correspond to one of the partitions of size |S|/q during cross-validation.

Let ruj be the value of the (hidden) rating of entry $(u, j) \in E$, which is used in the test set. Furthermore, let $\hat{r}uj$ be the predicted rating of the entry (u, j) by the specific training algorithm being used. The entry-specific error is given by $euj = \hat{r}uj - ruj$. This error can be leveraged in various ways to compute the overall error over the set E of entries on which the evaluation is performed. An example is the mean squared error, denoted by MSE:

$$MSE = \frac{\sum_{(u,j) \in E} e_{uj}^2}{|E|}$$

RMSE versus MAE

One problem with these metrics is that they are heavily influenced by the ratings on the popular items. The items that receive very few ratings are ignored. The X-axis represents the indices of the items in decreasing order of popularity, and the Y –axis indicates the rating frequency. It is evident that only a few items receive a large number of ratings, whereas most of the remaining items receive few ratings. The latter constitute the long tail



Evaluating Ranking via Correlation

The aforementioned measures are designed to evaluate the prediction accuracy of the actual rating value of a user-item combination. In practice, the recommender system creates a ranking of items for a user, and the top-k items are recommended. The value of k may vary with the system, item, and user at hand. In general, it is desirable for highly rated items to be ranked above items which are not highly rated. Consider a user u, for which the ratings of the set I_u of items have been hidden by a hold-out or cross-validation strategy. For example, if the ratings of the first, third, and fifth items (columns) of user (row) u are hidden for evaluation purposes, then we have $I_u = \{1, 3, 5\}$.

The two most commonly used rank correlation coefficients are as follows:

Spearman rank correlation coefficient: The first step is to rank all items from 1 to |Iu|, both for the recommender system prediction and for the ground-truth. The Spearman correlation coefficient is simply equal to the Pearson correlation coefficient applied on these ranks. The computed value always ranges in (−1, +1), and large positive values are more desirable.

The Spearman correlation coefficient is specific to user u, and it can then be averaged over all users to obtain a global value. Alternatively, the Spearman rank correlation can be computed over all the hidden ratings over all users in one shot, rather than computing user-specific values and averaging them.

One problem with this approach is that the ground truth will contain many ties, and therefore random tiebreaking might lead to some noise in the evaluation. For this purpose, an approach referred to as tiecorrected Spearman is used. One way of performing the correction is to use the average rank of all the ties, rather than using random tie-breaking. For example, if the ground-truth rating of the top-2 ratings is identical in a list of four items, then instead of using the ranks {1, 2, 3, 4}, one might use the ranks {1.5, 1.5, 3, 4}.

Kendall rank correlation coefficient: For each pair of items j, k ∈ Ii, the following credit C(j, k) is computed by comparing the predicted ranking with the ground-truth ranking of these items:

$$C(j,k) = \begin{cases} +1 & \text{if items } j \text{ and } k \text{ are in the same relative order in} \\ & \text{ground-truth ranking and predicted ranking (concordant)} \\ -1 & \text{if items } j \text{ and } k \text{ are in a different relative order in} \\ & \text{ground-truth ranking and predicted ranking (discordant)} \\ 0 & \text{if items } j \text{ and } k \text{ are tied in either the} \\ & \text{ground-truth ranking or predicted ranking} \end{cases} \tag{7.7}$$

Then, the Kendall rank correlation coefficient τ_u, which is specific to user u, is computed as the average value of $C(j,k)$ over all the $|I_u|(|I_u|-1)/2$ pairs of test items for user i:

$$\tau_u = \frac{\sum_{j<k} C(j,k)}{|I_u| \cdot (|I_u|-1)/2} \tag{7.8}$$

A different way of understanding the Kendall rank correlation coefficient is as follows:

$$\tau_u = \frac{\text{Number of concordant pairs} - \text{Number of discordant pairs}}{\text{Number of pairs in } I_u} \tag{7.9}$$

Evaluating Ranking via Receiver Operating Characteristic

Ranking methods are used frequently in the evaluation of the actual consumption of items. For example, Netflix might recommend a set of ranked items for a user, and the user might eventually consume only a subset of these items. Therefore, these methods are well suited to implicit feedback data sets, such as sales, click-throughs, or movie views. Such actions can be represented in the form of unary ratings matrices, in which missing values are considered to be equivalent to 0. Therefore, the ground-truth is of a binary nature.

The items that are eventually consumed are also referred to as the ground-truth positives or true positives. The recommendation algorithm can provide a ranked list of any number of items. What percentage of these items is relevant? A key issue here is that the answer to this question depends on the size of the recommended list. Changing the number of recommended items in the ranked list has a direct effect on the trade-off between the fraction of recommended items that are actually consumed and the fraction of consumed items that are captured by the recommender system. This trade-off can be measured in two different ways with the use of a precision-recall or a receiver operating characteristic (ROC) curve. Such trade-off plots are commonly used in rare class detection, outlier analysis evaluation, and information retrieval. In fact, such trade-off plots can be used in any application where a binary ground truth is compared to a ranked list discovered by an algorithm.

The basic assumption is that it is possible to rank all the items using a numerical score, which is the output of the algorithm at hand. Only the top items are recommended. By varying the size of the recommended list, one can then examine the fraction of relevant (ground-truth positive) items in the list, and the fraction of relevant items that are missed by the list. If the recommended list is too small, then the algorithm will miss relevant items (false-negatives). On the other hand, if a very large list is recommended, this will lead to too many spurious recommendations that are never used by the user (false-positives).

Algorithm	Rank of items that are truly used (ground-truth positives)
Algorithm A	1, 5, 8, 15, 20
Algorithm B	3, 7, 11, 13, 15
Random Algorithm	17, 36, 45, 59, 66
Perfect Oracle	1, 2, 3, 4, 5

This leads to a trade-off between the false-positives and false-negatives. The problem is that the correct size of the recommendation list is never known exactly in a real scenario. However, the entire trade-off curve can be quantified using a variety of measures, and two algorithms can be compared over the entire trade-off curve. Two examples of such curves are the precision-recall curve and the receiver operating characteristic (ROC) curve

Assume that one selects the top-t set of ranked items to recommend to the user. For any given value t of the size of the recommended list, the set of recommended items is denote by S(t). Note that $|S(t)| = t$. Therefore, as t changes, the size of S(t) changes as well. LetG represent the true set of relevant items (ground-truth positives) that are consumed by the user. Then, for any given size t of the recommended list, the precision is defined as the percentage of recommended items that truly turn out to be relevant (i.e., consumed by the user).

$$Precision(t) = 100 \cdot \frac{|\mathcal{S}(t) \cap \mathcal{G}|}{|\mathcal{S}(t)|}$$

The value of $Precision(t)$ is *not* necessarily monotonic in t because both the numerator and denominator may change with t differently. The *recall* is correspondingly defined as the percentage of *ground-truth* positives that have been recommended as positive for a list of size t.

$$Recall(t) = 100 \cdot \frac{|\mathcal{S}(t) \cap \mathcal{G}|}{|\mathcal{G}|}$$

While a natural trade-off exists between precision and recall, this trade-off is not necessarily monotonic. In other words, an increase in recall does not always lead to a reduction in precision. One way of creating a single measure that summarizes both precision and recall is the F_1-measure, which is the harmonic mean between the precision and the recall.

$$F_1(t) = \frac{2 \cdot Precision(t) \cdot Recall(t)}{Precision(t) + Recall(t)} \tag{7.21}$$

While the $F_1(t)$ measure provides a better quantification than either precision or recall, it is still dependent on the size t of the recommended list and is therefore still not a complete representation of the trade-off between precision and recall. It is possible to visually examine the entire trade-off between precision and recall by varying the value of t and plotting the precision versus the recall. As shown later with an example, the lack of monotonicity of the precision makes the results harder to intuitively interpret.

A second way of generating the trade-off in a more intuitive way is through the use of the ROC curve. The *true-positive rate*, which is the same as the recall, is defined as the percentage of ground-truth positives that have been included in the recommendation list of size t.

$$TPR(t) = Recall(t) = 100 \cdot \frac{|\mathcal{S}(t) \cap \mathcal{G}|}{|\mathcal{G}|}$$

The aforementioned description illustrates the generation of customer-specific ROC curves, because the ROC curves are specific to each user. It is also possible to generate global ROC curves by ranking useritem pairs and then using the same approach as discussed above. In order to rank user-item pairs, it is assumed that the algorithm has a mechanism to rank them by using predicted affinity values. For example, the predicted ratings for user-item pairs can be used to rank them.