Chapter 7

Evaluating Recommender Systems

"*True genius resides in the capacity for evaluation of uncertain, hazardous, and conflicting information.*"– Winston Churchill

7.1 Introduction

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.

Recommender systems can be evaluated using either *online* methods or *offline* methods. In an online system, the user reactions are measured with respect to the presented recommendations. Therefore, user participation is essential in online systems. For example, in an online evaluation of a news recommender system, one might measure the *conversion rate* of users clicking on articles that were recommended. Such testing methods are referred to as *A/B testing*, and they measure the direct impact of the recommender system

C.C. Aggarwal, *Recommender Systems: The Textbook*,
DOI 10.1007/978-3-319-29659-3_7

on the end user. At the end of the day, increasing the conversion rate on profitable items is the most important goal of a recommender system, and it can provide a true measure of the effectiveness of the system. However, since online evaluations require active user participation, it is often not feasible to use them in benchmarking and research. There are usually significant challenges in gaining access to user conversion data from systems with large-scale user participation. Even if such access is gained, it is usually specific to a single large-scale system. On the other hand, one often desires to use data sets of different types, and from multiple domains. Testing over multiple data sets is particularly important for assuring greater generalization power of the recommender system so that one can be assured that the algorithm works under a variety of settings. In such cases, offline evaluations with historical data sets are used. Offline methods are, by far, the most common methods for evaluating recommender systems from a research and practice perspective. Therefore, most of this chapter will focus on offline methods, although some discussion of online methods is also included for completeness.

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.

In this chapter, we will first begin by discussing the general goals of evaluating recommender systems beyond the most basic criterion of accuracy. Examples of such goals include diversity and novelty. The main challenge with *quantifying* such goals is that they are often subjective goals based on user experience. From a quantification perspective, accuracy is a concrete goal that is relatively easy to measure and is therefore used more frequently for bench-marking and testing. A few quantification methods do exist for evaluating the secondary goals such as diversity and novelty. Although the majority of this chapter will focus on accuracy metrics, various quantification measures for the secondary goals will also be discussed.

This chapter is organized as follows. An overview of the different types of evaluation systems is provided in section 7.2. Section 7.3 studies the general goals of evaluating recommender systems. The appropriate design of accuracy testing methods is discussed in section 7.4. Accuracy metrics for recommender systems are discussed in section 7.5. The limitations of evaluation measures are discussed in 7.6. A summary is given in section 7.7.

7.2 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.

7.2.1 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.

7.2.2 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 [305]. 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. Although reinforcement learning has been studied extensively in the classification and regression modeling literature [579], the corresponding work in the recommendation domain is rather limited [389, 390, 585]. A significant research opportunity exists for the further development of such algorithms.

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.

7.2.3 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 [311]. 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 long-term 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.

7.3 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.

7.3.1 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. A detailed discussion is provided in section 7.5 of this chapter. In the most general case, ratings are numeric quantities that need to be estimated. Therefore, the accuracy metrics are often similar to those used in regression modeling. Let R be the ratings matrix in which r_{uj} 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 $e_{uj} = \hat{r}_{uj} - r_{uj}$. 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.

As the various methods for computing accuracy are discussed in detail in section 7.5, they are not discussed in detail here. The goal of this short section is to briefly introduce a few measures to ensure continuity in further discussion. The main components of accuracy evaluation are as follows:

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 overfitting. 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. These issues will be discussed in greater detail in section 7.4.

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. Different classes of methods are used for the two cases:

   - *Accuracy of estimating ratings:* As discussed above, the entry-specific error is given by $e_{uj} = \hat{r}_{uj} - r_{uj}$ for user u and item j. This error can be leveraged in various ways to compute the overall error over the set E of entries in the ratings matrix on which the evaluation is performed. An example is the *mean squared error*, which is denoted by *MSE*:

     $$MSE = \frac{\sum_{(u,j) \in E} e_{uj}^2}{|E|} \tag{7.1}$$

     The square-root of the aforementioned quantity is referred to as the *root mean squared error*, or *RMSE*.

     $$RMSE = \sqrt{\frac{\sum_{(u,j) \in E} e_{uj}^2}{|E|}} \tag{7.2}$$

     Most of these measures are borrowed from the literature on regression modeling. Other important ways of measuring the error, such as the mean absolute error, are discussed in section 7.5.

- *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. These methods are discussed in detail in section 7.5.

Some measures of accuracy are also designed to maximize the profit for the merchant because all items are not equally important from the perspective of the recommendation process. These metrics incorporate item-specific costs into the computation. The main problem with accuracy metrics is that they often do not measure the true effectiveness of a recommender system in real settings. For example, an obvious recommendation might be accurate, but a user might have eventually bought that item anyway. Therefore, such a recommendation might have little usefulness in terms of improving the conversion rate of the system. A discussion of the challenges associated with the use of accuracy metrics may be found in [418].

7.3.2 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 *item-space coverage*, respectively.

User-space coverage measures the fraction of users for which at least k ratings may be predicted. The value of k should be set to the expected size of the recommendation list. When fewer than k ratings can be predicted for a user, it is no longer possible to present a meaningful recommendation list of size k to the user. Such a situation could occur when a user has specified very few ratings in common with other users. Consider a user-based neighborhood algorithm. It is difficult to robustly compute the peers of that user, because of very few mutually specified ratings with other users. Therefore, it is often difficult to make sufficient recommendations for that user. For very high levels of sparsity, it is possible that no algorithm may be able to predict even one rating for that user. However, different algorithms may have different levels of coverage, and the coverage of a user can be estimated by running each algorithm and determining the number of items for which a prediction is made. A tricky aspect of user-space coverage is that any algorithm can provide full coverage by simply predicting random ratings for user-item combinations, whose ratings it cannot reliably predict. Therefore, user-space coverage should always be evaluated in terms of the trade-off between accuracy and coverage. For example, in a neighborhood-based recommender increasing the size of the neighborhood provides a curve showing the trade-off between coverage and accuracy.

An alternative definition of user-space coverage is in terms of the minimum amount of profile that must be built for a user before it is possible to make recommendations for that user. For a particular algorithm, it is possible to estimate through experiments the minimum number of observed ratings of any user for which a recommendation could be made. However, it is often difficult to evaluate this quantity because the metric is sensitive to the identity of the items for which the user specifies ratings.

The notion of *item-space coverage* is analogous to that of user-space coverage. Item-space coverage measures the fraction of items for which the ratings of at least k users can be predicted. In practice, however, this notion is rarely used, because recommender systems generally provide recommendation lists for users, and they are only rarely used for generating recommended users for items.

A different form of item-space coverage evaluation is defined by the notion of *catalog coverage*, which is specifically suited to recommendation *lists*. Note that the aforementioned definition was tailored to the prediction of the values of ratings. Imagine a scenario where every entry in the ratings matrix can be predicted by an algorithm, but the same set of top-k items is always recommended to every user. Therefore, even though the aforementioned definition of item-space coverage would suggest good performance, the actual coverage across all users is very limited. In other words, the recommendations are not diverse across users, and the catalog of items is not fully covered. Let T_u represent the list of top-k items recommended to user $u \in \{1 \dots m\}$. The catalog coverage CC is defined as the fraction of items that are recommended to at least one user.

$$CC = \frac{| \cup_{u=1}^{m} T_u|}{n} \tag{7.3}$$

Here, the notation n represents the number of items. It is easy to estimate this fraction through the use of experiments.

7.3.3 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. Therefore, it is possible to use such systems only by setting the same level of confidence in both cases.

While confidence measures the system's faith in the recommendation, trust measures the user's faith in the evaluation. The notion of social trust is discussed in more detail in Chapter 11. Broadly speaking, trust measures the level of faith that the *user* has in the reported ratings. Even if the predicted ratings are accurate, they are often not useful if the user fails to trust the provided ratings. Trust is closely related to, but not quite the same as, accuracy. For example, when explanations are provided by the recommender system, the user is more likely to trust the system, especially if the explanations are logical.

Trust often does not serve the same goals as the usefulness (utility) of a recommendation. For example, if a recommender system suggests a few items already liked and known by the user, it can be argued that there is little utility provided to the user from such a recommendation. On the other hand, such items can increase the trust of the user in the system. This goal is directly in contradiction to other goals such as novelty in which recommendations already known by the user are undesirable. It is common for the various goals in recommender systems to trade-off against one another. The simplest way to measure trust is to conduct user surveys during the experiments in which the users are explicitly queried about their trust in the results. Such experiments are also referred to as *online experiments*. Numerous online methods for trust evaluation are discussed in [171, 175, 248, 486]. Generally, it is hard to measure trust through offline experiments.

7.3.4 Novelty

The novelty of a recommender system evaluates the likelihood of a recommender system to give recommendations to the user that they are not aware of, or that they have not seen before. A discussion of the notion of novelty is provided in [308]. Unseen recommendations often increase the ability of the user to discover important insights into their likes and dislikes that they did not know previously. This is more important than discovering items that they were already aware of but they have not rated. In many types of recommender systems, such as content-based methods, the recommendations tend to be somewhat obvious because of the propensity of the system to recommend expected items. While a small number of such recommendations can improve the trust of the end user in the underlying system, they are not always useful in terms of improving conversion rates. The most natural way of measuring novelty is through online experimentation in which users are explicitly asked whether they were aware of an item previously.

As discussed in the introduction, online experimentation is not always feasible because of the lack of access to a system supporting a large base of online users. Fortunately, it is possible to approximately estimate novelty with offline methods, as long as time stamps are available with the ratings. The basic idea is that novel systems are better at recommending items that are more likely to be selected by the user in the *future*, rather than at the present time. Therefore, all ratings that were created after a certain point in time t_0 are removed from the training data. Furthermore, some of the ratings occurring before t_0 are also removed. The system is then trained with these ratings removed. These removed items are then used for scoring purposes. For each item rated before time t_0 and correctly recommended, the novelty evaluation score is penalized. On the other hand, for each item rated after time t_0 and correctly recommended, the novelty evaluation score is rewarded. Therefore, this evaluation measures a type of *differential* accuracy between future and past predictions. In some measures of novelty, it is assumed that popular items are less likely to be novel, and less credit is given for recommending popular items.

7.3.5 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. This notion also appears in the context of information retrieval applications [670]. The work in [214] proposed both online and offline methods for evaluating serendipity:

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.

It is noteworthy that it is not sufficient to measure the fraction of non-obvious items, because a system might recommend unrelated items. Therefore, the usefulness of the items is always incorporated in the measurement of serendipity. Serendipity has important long-term effects on improving the conversion rate of a recommender system, even when it is opposed to the immediate goal of maximizing accuracy. A number of metrics for serendipity evaluation are discussed in [214, 450].

7.3.6 Diversity

The notion of diversity implies that the set of proposed recommendations *within a single recommended list* should be as diverse as possible. For example, consider the case where three movies are recommended to a user in the list of top-3 items. If all three movies are of a particular genre and contain similar actors, then there is little diversity in the recommendations. If the user dislikes the top choice, then there is a good chance that she might dislike all of them. Presenting different types of movies can often increase the chance that the user might select one of them. Note that the diversity is always measured over a *set* of recommendations, and it is closely related to novelty and serendipity. Ensuring greater diversity can often increase the novelty and serendipity of the recommendations. Furthermore, greater diversity of recommendations can also increase the sales diversity and catalog coverage of the system.

Diversity can be measured in terms of the content-centric similarity between pairs of items. The vector-space representation of each item description is used for the similarity computation. For example, if a set of k items are recommended to the user, then the pairwise similarity is computed between every pair of items in the list. The average similarity between all pairs can be reported as the diversity. Lower values of the average similarity indicate greater diversity. Diversity can often provide very different results from those of accuracy metrics. A discussion of the connection of diversity and similarity is provided in [560].

7.3.7 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 [158, 329, 393, 444]. 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. Attack models for recommender systems are discussed in Chapter 12. The evaluation of such models is also studied in the same chapter. The corresponding measures can be used to estimate the robustness and stability of such systems against attacks.

7.3.8 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 [527, 528, 587]. A variety of measures are used for determining the scalability of a system:

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.

The importance of scalability has become particularly great in recent years because of the increasing importance of the "big-data" paradigm.

7.4 Design Issues in Offline Recommender Evaluation

In this section, we will discuss the issue of recommender evaluation design. 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.

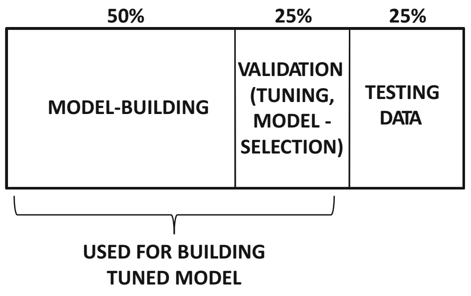
2. *Validation data:* This part of the data is used for model selection and parameter tuning. For example, the regularization parameters in a latent factor model may be determined by testing the accuracy over the validation data. In the event that multiple models have been built from the training data, 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.

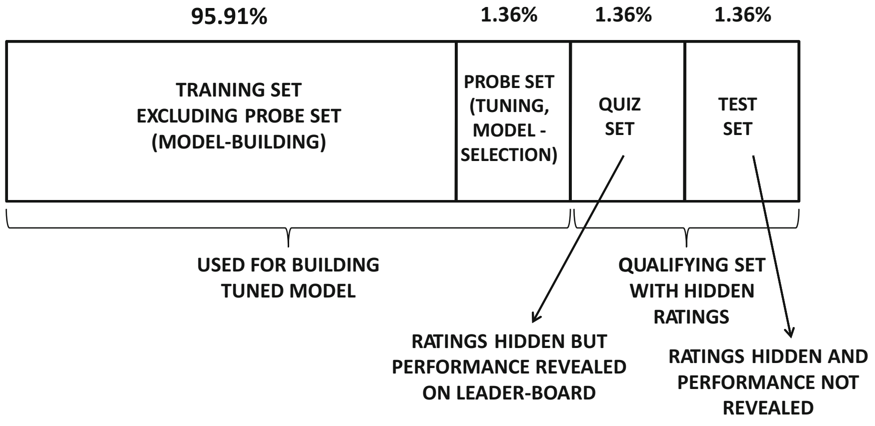
An example of a division of the ratings matrix into training, validation, and testing data is illustrated in Figure 7.1(a). Note that the validation data may also be considered a part of the training data because they are used to create the final tuned model. The division of the ratings matrix into the ratios 2:1:1 is particularly common. In other words, half the specified ratings are used for model-building, and a quarter may be used for each of model-selection and testing, respectively. However, when the sizes of the ratings matrices are large, it is possible to use much smaller proportions for validation and testing. This was the case for the Netflix Prize data set.

7.4.1 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


50%
25%
25%
MODEL-BUILDING
VALIDATION (TUNING, MODEL - SELECTION)
TESTING DATA
USED FOR BUILDING TUNED MODEL

(a) Proportional division of ratings


95.91%
1.36%
1.36%
1.36%
TRAINING SET EXCLUDING PROBE SET (MODEL-BUILDING)
PROBE SET (TUNING, MODEL - SELECTION)
QUIZ SET
TEST SET
USED FOR BUILDING TUNED MODEL
QUALIFYING SET WITH HIDDEN RATINGS
RATINGS HIDDEN BUT PERFORMANCE REVEALED ON LEADER-BOARD
RATINGS HIDDEN AND PERFORMANCE NOT REVEALED

(b) Division in Netflix Prize data set (not drawn to scale)

Figure 7.1: Partitioning a ratings matrix for evaluation design

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 [554], 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 $\langle User, Movie, GradeDate \rangle$ 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 out-of-sample data set was used to determine the final winners of the contest.

The overall division of the Netflix data set is shown in Figure 7.1(b). The only difference from the division in Figure 7.1(a) is the presence of an additional quiz set. It is, in fact, possible to remove the quiz set entirely without affecting the Netflix contest in any significant way, except that participants would no longer be able to obtain an idea of the quality of their submissions. Indeed, the Netflix Prize evaluation design is an excellent example of the importance of not using any knowledge of the performance on the test set at any stage of the training process *until the very end*. Benchmarking in research and practice often fails to meet these standards in one form or the other.

7.4.2 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 divide[1] 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. However, these methods are also used for dividing the training data into the model-building and validation portions. This hierarchical division is illustrated in Figure 7.2. In the following, we will consistently use the terminology of the first level of division in Figure 7.2 into “training” and “testing” data, even though the same approach can also be used for the second level division into model building and validation portions. This consistency in terminology is followed to avoid confusion.

7.4.2.1 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

[1]The actual design in methods such as cross-validation is slightly more complex because the data are segmented in multiple ways, even though they are always divided into two parts during a particular execution phase of training.

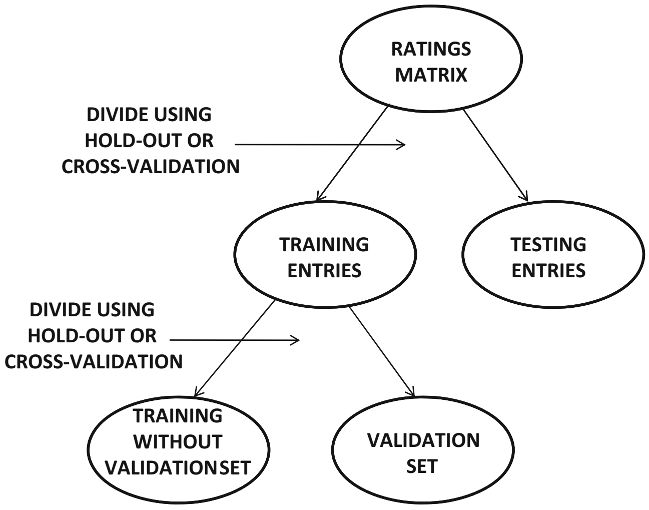

RATINGS MATRIX
DIVIDE USING HOLD-OUT OR CROSS-VALIDATION
TRAINING ENTRIES
TESTING ENTRIES
DIVIDE USING HOLD-OUT OR CROSS-VALIDATION
TRAINING WITHOUT VALIDATIONSET
VALIDATION SET

Figure 7.2: Hierarchical division of rated entries into training, validation, and testing portions

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 lower average rating than the ratings matrix, and also the held-out entries. This will lead to a pessimistic bias in the evaluation.

7.4.2.2 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, leave-one-out cross-validation is not very difficult to implement for the specific case of neighborhood-based collaborative filtering algorithms.

7.4.3 Comparison with Classification Design

The evaluation design in collaborative filtering is very similar to that in classification. This is not a coincidence. Collaborative filtering is a generalization of the classification problem, in which any missing entry can be predicted rather than simply a particular variable, which is designated as the dependent variable. The main difference from classification is that the data are segmented on a row-wise basis (between training and test *rows*) in classification, whereas the data are segmented on an entry-wise basis (between training and test *entries*) in collaborative filtering. This difference closely mirrors the nature of the relationship between the classification and the collaborative filtering problems. Discussions of evaluation designs in the context of the classification problem can be found in [18, 22].

One difference from classification design is that the performance on hidden entries often does not reflect the true performance of the system in real settings. This is because the hidden ratings are not chosen at random from the matrix. Rather, the hidden ratings are typically items that the user has chosen to consume. Therefore, such entries are likely to have higher values of the ratings as compared to truly missing values. This is a problem of *sample selection bias*. Although this problem could also arise in classification, it is far more pervasive in collaborative filtering applications. A brief discussion of this issue is provided in section 7.6.

7.5 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.

7.5.1 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 r_{uj} 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 $e_{uj} = \hat{r}_{uj} - r_{uj}$. 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*:

$$MSE = \frac{\sum_{(u,j) \in E} e_{uj}^2}{|E|} \tag{7.4}$$

Clearly, smaller values of the *MSE* are indicative of superior performance. The square-root of this value is referred to as the *root mean squared error (RMSE)*, and it is often used instead of the *MSE*:

$$RMSE = \sqrt{\frac{\sum_{(u,j) \in E} e_{uj}^2}{|E|}} \tag{7.5}$$

The *RMSE* is in units of ratings, rather than in units of squared ratings like the *MSE*. The *RMSE* was used as the standard metric for the Netflix Prize contest. One characteristic of the *RMSE* is that it tends to disproportionately penalize large errors because of the squared term within the summation. One measure, known as the *mean-absolute-error (MAE)*, does not disproportionately penalize larger errors:

$$MAE = \frac{\sum_{(u,j) \in E} |e_{uj}|}{|E|} \tag{7.6}$$

Other related measures such as the normalized *RMSE* (*NRMSE*) and normalized *MAE* (*NMAE*) are defined in a similar way, except that each of them is divided by the range $r_{max} - r_{min}$ of the ratings:

$$NRMSE = \frac{RMSE}{r_{max} - r_{min}}$$
$$NMAE = \frac{MAE}{r_{max} - r_{min}}$$

The normalized values of the *RMSE* and *MAE* always lie in the range $(0, 1)$, and therefore they are more interpretable from an intuitive point of view. It is also possible to use these values to compare the performance of a particular algorithm over different data sets with varying scales of ratings.

7.5.1.1 RMSE versus MAE

Is *RMSE* or *MAE* better as an evaluation measure? There is no clear answer to this question, as this depends on the application at hand. As the *RMSE* sums up the squared errors, it is more significantly affected by large error values or outliers. A few badly predicted ratings can significantly ruin the *RMSE* measure. In applications where robustness of prediction across various ratings is very important, the *RMSE* may be a more appropriate measure. On the other hand, the *MAE* is a better reflection of the accuracy when the importance of outliers in the evaluation is limited. The main problem with *RMSE* is that it is not a true reflection of the average error, and it can sometimes lead to misleading results [632]. Clearly, the specific choice should depend on the application at hand. A discussion of the relative benefits of the two kinds of measures can be found in [141].

7.5.1.2 Impact of the Long Tail

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. As discussed in Chapter 2, ratings matrices exhibit a long-tail property, in which the vast majority of items are bought (or rated) rarely. We have replicated Figure 2.1 of Chapter 2 in Figure 7.3. 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. Unfortunately, items in the long tail often contribute to the vast majority of profit for merchants [49]. As a result, the most important items often get weighted the least in the evaluation process. Furthermore, it is often much harder to predict the values of

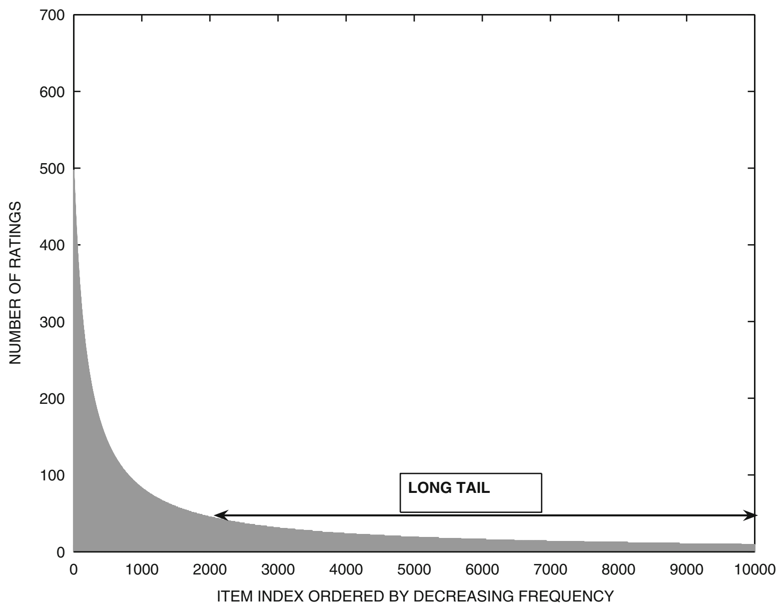


Figure 7.3: The long tail of rating frequencies (Revisiting Figure 2.1 of Chapter 2)

the ratings in the long tail because of greater local sparsity [173]. Therefore, the prediction accuracies on sparse items will typically be different from those on popular items. One way of handling this problem is to compute the *RMSE* or *MAE* separately for all the hidden ratings associated with each item, and then average over the different items in a weighted way. In other words, the accuracy computations of Equations 7.5 and 7.6 can be weighted with an item-specific weight, depending on the relative importance, profit, or utility to the merchant. It is also possible to perform these computations with user-specific weighting (rather than item-specific weighting), although the practical applicability of user-specific weighting is limited.

7.5.2 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 $I_u = \{1, 3, 5\}$.

We would like to measure how well the ground-truth orderings of the ratings in I_u are related to the ordering predicted by the recommender system for the set I_u. An important issue to keep in mind is that ratings are typically chosen from a discrete scale, and many ties exist in the ground truth. Therefore, it is important for the ranking measures to not penalize the system for ranking one item above another when they are tied in the ground truth. The most common class of methods is to use *rank correlation coefficients.* The two most commonly used rank correlation coefficients are as follows:

1. *Spearman rank correlation coefficient:* The first step is to rank all items from 1 to $|I_u|$, 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 tie-breaking might lead to some noise in the evaluation. For this purpose, an approach referred to as *tie-corrected 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 $\{1.5, 1.5, 3, 4\}$.

2. *Kendall rank correlation coefficient:* For each pair of items $j, k \in I_i$, 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:

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

   Note that this value is a *customer-specific* value of the Kendall coefficient. The value of τ_u may be averaged over all users u to obtain a heuristic global measure. Alternatively, one can perform the Kendall coefficient computation of Equation 7.8 over all hidden user-item pairs, rather than only the ones for customer u, in order to obtain a global value τ.

A number of other measures, such as the *normalized distance-based performance measure (NDPM)*, have been proposed in the literature. Refer to the bibliographic notes.

7.5.3 Evaluating Ranking via Utility

In the previous discussion, the ground-truth ranking is compared to the recommender system's ranking. Utility-based methods use the ground-truth *rating* in combination with the recommender system's ranking. For the case of implicit feedback data sets, the rating is substituted with a 0-1 value, depending on whether or not the customer has consumed the item. The overall goal of utility-based methods is to create a crisp quantification of how *useful* the customer might find the recommender system's ranking. An important principle underlying such methods is that recommendation lists are short compared to the total number of items. Therefore, most of the utility of a particular ranking should be based on the relevance of items, which are high in the recommended list. In this sense, the *RMSE* measure has a weakness because it equally weights the errors on the low-ranked items as compared to those on the highly-ranked items. It has been suggested [713] that small changes in *RMSE* such as 1%, can lead to large changes of more than 15% in the *identities* of the top-rated items. These are the only items that the end-user of the recommender system will actually see. Correspondingly, utility-based measures quantify the utility of a recommendation list by giving greater importance to the top-ranked items.

As in the previous sections, it is assumed that the ground-truth rating of each item in I_u is hidden from the recommender system before evaluation. Here, I_u represents the set of items rated by user u, which are hidden from the recommender system before evaluation. We will develop both user-specific and global utility quantifications.

In utility-based ranking, the basic idea is that each item in I_u has a utility to the user, which depends both on its position in the recommended list and its ground-truth rating. An item that has a higher ground-truth rating obviously has greater utility to the user. Furthermore, items ranked higher in the recommended list have greater utility to the user i because they are more likely to be noticed (by virtue of their position) and eventually selected. Ideally, one would like items with higher ground-truth rating to be placed as high on the recommendation list as possible.

How are these rating-based and ranking-based components defined? For any item $j \in I_u$, its rating-based utility to the user i is assumed to be $\max\{r_{uj} - C_u, 0\}$, where C_u is a break-even (neutral) rating value for user u. For example, C_u might be set to the mean rating of user u. On the other hand, the ranking-based utility of the item is $2^{-(v_j-1)/\alpha}$, where v_j is the rank of item j in the list of recommended items and α is a half-life parameter. In other words, the ranking-based utility exponentially decays with its rank, and moving down the ranks by α reduces the utility by a factor of 2. The logic of the decay-based ranking component is to ensure that the final utility of a particular ranking is regulated primarily by the top few items. After all, the user rarely browses the items that are very low in the list. The utility $F(u, j)$ of item $j \in I_u$ to user u is defined as the product of the rating-based and ranking-based utility values:

$$F(u, j) = \frac{\max\{r_{uj} - C_u, 0\}}{2^{(v_j-1)/\alpha}} \tag{7.10}$$

The R-score, which is specific to user u, is the sum of $F(u, j)$ over all the hidden ratings in I_u:

$$\text{R-score}(u) = \sum_{j \in I_u} F(u, j) \tag{7.11}$$

Note that the value of v_j can take on any value from 1 to n, where n is the total number of items. However, in practice, one often restricts the size of the recommended list to a maximum value of L. One can therefore compute the R-score over a recommended list of specific size L instead of using all the items, as follows:

$$\text{R-score}(u) = \sum_{j \in I_u, v_j \leq L} F(u, j) \tag{7.12}$$

The idea here is that ranks below L have no utility to the user because the recommended list is of size L. This variation is based on the principle that recommended lists are often very short compared to the total number of items. The overall R-score may be computed by summing this value over all the users.

$$\text{R-score} = \sum_{u=1}^{m} \text{R-score}(u) \tag{7.13}$$

The exponential decay in the utility implies that users are only interested in top-ranked items, and they do not pay much attention to lower-ranked items. This may not be true in all applications, especially in news recommender systems, where users typically browse multiple items lower down the list of recommended items. In such cases, the discount rate should be set in a milder way. An example of such a measure is the discounted cumulative gain (DCG). In this case, the discount factor of item j is set to $\log_2(v_j + 1)$, where v_j is the rank of item j in the test set I_u. Then, the discounted cumulative gain is defined as follows:

$$DCG = \frac{1}{m} \sum_{u=1}^{m} \sum_{j \in I_u} \frac{g_{uj}}{\log_2(v_j + 1)} \tag{7.14}$$

Here, g_{uj} represents the utility (or gain) of the user u in consuming item j. Typically, the value of g_{uj} is set to an exponential function of the relevance (e.g., non-negative ratings or user hit rates):

$$g_{uj} = 2^{rel_{uj}} - 1 \tag{7.15}$$

Here, rel_{uj} is the ground-truth relevance of item j for user u, which is computed as a heuristic function of the ratings or hits. In many settings, the raw ratings are used. It is common to compute the discounted cumulative gain over a recommendation list of specific size L, rather than using all the items:

$$DCG = \frac{1}{m} \sum_{u=1}^{m} \sum_{j \in I_u, v_j \leq L} \frac{g_{uj}}{\log_2(v_j + 1)} \tag{7.16}$$

The basic idea is that recommended lists have size no larger than L.

Then, the normalized discounted cumulative gain (NDCG) is defined as ratio of the discounted cumulative gain to its ideal value, which is also referred to as ideal discounted cumulative gain (IDCG).

$$NDCG = \frac{DCG}{IDCG} \tag{7.17}$$

The ideal discounted cumulative gain is computed by repeating the computation for DCG, except that the ground-truth rankings are used in the computation.

Another measure that is commonly used, is the average reciprocal hit rate (ARHR) [181]. This measure is designed for implicit feedback data sets, in which each value of $r_{uj} \in \{0, 1\}$. Therefore, a value of $r_{uj} = 1$ represents a "hit" where a customer has bought or clicked on an item. A value of $r_{uj} = 0$ corresponds to a situation where a customer has not bought or clicked on an item. In this implicit feedback setting, missing values in the ratings matrix are assumed to be 0.

In this case, the rank-based discount rate is $1/v_j$, where v_j is the rank of item j in the recommended list, and the item utility is simply the hidden "rating" value $r_{uj} \in \{0,1\}$. Note that the discount rate is not as rapid as the R-score metric, but it is faster than DCG. Therefore, the combined utility of an item is given by r_{uj}/v_j. This expression represents the contribution of item $j \in I_u$ to the utility. Then, the ARHR metric for the user i is defined by summing up these values over all the hidden items in I_u:

$$ARHR(u) = \sum_{j \in I_u} \frac{r_{uj}}{v_j} \tag{7.18}$$

It is also possible to define the average reciprocal hit-rate for a recommended list of size L by adding only those utility values for which $v_j \leq L$.

$$ARHR(u) = \sum_{j \in I_u, v_j \leq L} \frac{r_{uj}}{v_j} \tag{7.19}$$

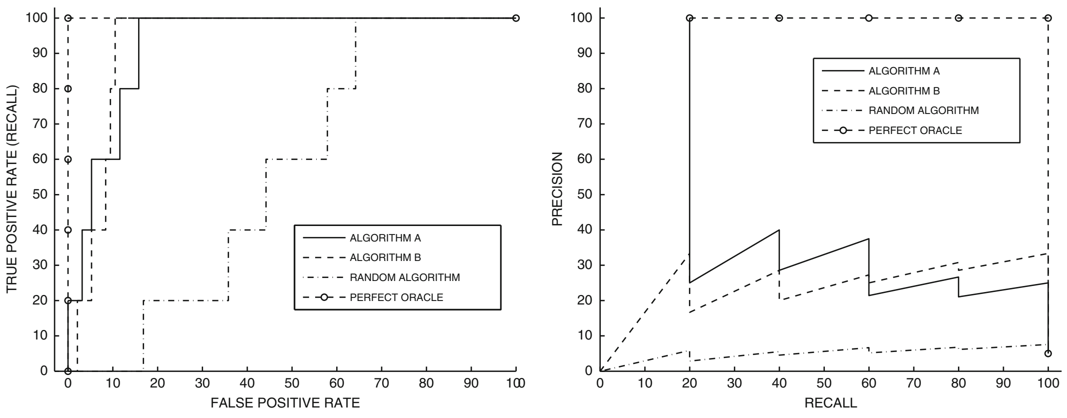
One quirk of the average reciprocal hit-rate is that it is typically used when the value of $|I_u|$ is exactly 1, and when the value r_{uj} of the corresponding (hidden) item $j \in I_u$ is always 1. Therefore, there is exactly one hidden item for each user, and the user has always bought or clicked on this item. In other words, the average reciprocal hit-rate rewards the utility (in a rank-reciprocal way) for recommending the single correct answer at a high position on the recommended list. This was the setting in which this measure was introduced [181], although one can generalize it to arbitrary settings in terms of the number of hidden items and explicit-feedback settings. The aforementioned expression provides this generalized definition because one can use a set I_u of arbitrary size in an explicit feedback setting. The global ARHR value is computed by averaging this value over the m users:

$$ARHR = \frac{\sum_{u=1}^{m} ARHR(u)}{m} \tag{7.20}$$

The average reciprocal hit-rate is also referred to as the *mean reciprocal rank (MRR).* In cases where the value of $|I_u|$ is 1, the average reciprocal hit-rate always lies in the range $(0,1)$. In such cases, the hidden entry is usually an item for which $r_{uj} = 1$ and the length of the recommendation list is restricted to L. Note that only "hits" contribute to the utility in these cases. A simplification of this measure is the *hit-rate*, in which the rank-reciprocal weighting is not used, and the value of $|I_u|$ is exactly 1. Therefore, the hit-rate (HR) is simply the fraction of users for which the correct answer is included in the recommendation list of length L. The disadvantage of the hit-rate is that it gives equal importance to a hit, irrespective of its position in the recommended list.

The ARHR and HR are almost always used in implicit feedback data sets, in which missing values are treated as 0. Nevertheless, the definition of Equation 7.19 is stated in a more general way. Such a definition can also be used in the context of explicit feedback data sets, in which the values of r_{uj} need not be drawn from $\{0,1\}$. In such cases, the ratings of any number of items of each user are hidden, and the values of the hidden ratings can be arbitrary. Furthermore, the missing values need not be treated as 0s, and I_u is always selected from the observed items.

A related measure is the *mean average precision (MAP),* which computes the fraction of relevant items in a recommended list of length L for a given user. Various equally spaced



(a) ROC (b)Precision-recall

Figure 7.4: ROC curve and precision-recall curves

values of L are used, and the precision is averaged over these recommendation lists of varying lengths. The resulting precision is then averaged over all the users.

Numerous other measures have been proposed in the literature to evaluate the effectiveness of rankings. For example, the *lift index* [361] divides the ranked items into deciles to compute a utility score. Refer to the bibliographic notes.

7.5.4 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).

Table 7.1: Rank of ground-truth positive instances

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 denoted by $\mathcal{S}(t)$. Note that $|\mathcal{S}(t)| = t$. Therefore, as t changes, the size of $\mathcal{S}(t)$ changes as well. Let $\mathcal{G}$ 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 false-positive rate $FPR(t)$ is the percentage of the falsely reported positives in the recommended list out of the ground-truth negatives (i.e., irrelevant items not consumed by the user). Therefore, if $\mathcal{U}$ represents the universe of all items, the ground-truth negative set is given by $(\mathcal{U} - \mathcal{G})$, and the falsely reported part in the recommendation list is $(\mathcal{S}(t) - \mathcal{G})$. Therefore, the false-positive rate is defined as follows:

$$FPR(t) = 100 \cdot \frac{|\mathcal{S}(t) - \mathcal{G}|}{|\mathcal{U} - \mathcal{G}|} \tag{7.22}$$

The false-positive rate can be viewed as a kind of "bad" recall, in which the fraction of the ground-truth negatives (i.e., items not consumed), which are incorrectly captured in the recommended list $\mathcal{S}(t)$, is reported. The ROC curve is defined by plotting the $FPR(t)$ on the X-axis and $TPR(t)$ on the Y-axis for varying values of t. In other words, the ROC curve plots the "good" recall against the "bad" recall. Note that both forms of recall will be at 100% when $\mathcal{S}(t)$ is set to the entire universe of items. Therefore, the end points of the ROC curve are always at $(0, 0)$ and $(100, 100)$, and a random method is expected to exhibit performance along the diagonal line connecting these points. The *lift* obtained above this diagonal line provides an idea of the accuracy of the approach. The area under the ROC curve provides a concrete quantitative evaluation of the effectiveness of a particular method. Although one can directly use the area shown in Figure 7.4(a), the staircase-like ROC curve is often modified to use local linear segments which are not parallel to either the X-axis or the Y-axis. The trapezoidal rule [195] is then used to compute the area slightly more accurately. From a practical point of view, this change often makes very little difference to the final computation.

To illustrate the insights gained from these different graphical representations, consider an example of a scenario with 100 items, in which 5 items are truly relevant. Two algorithms A and B are applied to this data set that rank all items from 1 to 100, with lower ranks being selected first in the recommended list. Thus, the true-positive rate and false-positive rate values can be generated from the ranks of the 5 relevant items. In Table 7.1, some hypothetical ranks for the 5 truly relevant items have been illustrated for the different algorithms. In addition, the ranks of the ground-truth positive items for a random algorithm have been indicated. This algorithm ranks all the items randomly. Similarly, the ranks for a "perfect oracle" algorithm, which ranks the correct top 5 items in the recommended list, have also been illustrated in the table. The resulting ROC curves are illustrated in Figure 7.4(a). The corresponding precision-recall curves are illustrated in Figure 7.4(b). Note that the ROC curves are always increasing monotonically, whereas the precision-recall curves are not monotonic. While the precision-recall curves are not quite as nicely interpretable as the ROC curves, it is easy to see that the *relative trends* between different algorithms are the same in both cases. In general, ROC curves are used more frequently because of greater ease in interpretability.

What do these curves really tell us? For cases in which one curve strictly dominates another, it is clear that the algorithm for the former curve is superior. For example, it is immediately evident that the oracle algorithm is superior to all algorithms and that the random algorithm is inferior to all the other algorithms. On the other hand, algorithms A and B show domination at different parts of the ROC curve. In such cases, it is hard to say that one algorithm is strictly superior. From Table 7.1, it is clear that Algorithm A ranks three relevant items very highly, but the remaining two items are ranked poorly. In the case of Algorithm B, the highest ranked items are not as well ranked as Algorithm A, though all 5 relevant items are determined much earlier in terms of rank threshold. Correspondingly, Algorithm A dominates on the earlier part of the ROC curve, whereas

Algorithm B dominates on the later part. It is possible to use the area under the ROC curve as a proxy for the overall effectiveness of the algorithm. However, not all parts of the ROC curve are equally important because there are usually practical limits on the size of the recommended list.

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 user-item 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.

7.5.5 Which Ranking Measure is Best?

Although ROC curves are often used for evaluating recommender systems, they do not always reflect the performance from the end-user perspective. In many settings, the end user sees only a small subset of top-ranked items. Measures such as ROC and Kendall coefficient, which treat higher and lower ranked items equally, are unable to capture the greater importance of higher ranked items. For example, the relative ranking between two items ranked first and second on the recommendation list is far more important than the relative ranking of two items, which are ranked 100th and 101st on the list. In this context, utility-based measures such as NDCG do a much better job than rank-correlation coefficients or ROC measures at distinguishing between higher-ranked and lower-ranked items.

7.6 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. As shown in Figure 7.3, 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 affect[2] 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 [564]. 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. This issue is stated in [184] in a somewhat different context as follows:

> "Intuitively, a simple process could explain the results: users chose to rate songs they listen to, and listen to music they expect to like, while avoiding genres they dislike. Therefore, most of the songs that would get a bad rating are not voluntarily rated by the users. Since people rarely listen to random songs, or rarely watch random movies, we should expect to observe in many areas a difference between the distribution of ratings for random items and the corresponding distribution for the items selected by the users."

[2]A related effect is that observed ratings are likely to be specified by users who are frequent raters. Frequent raters may show different patterns of rating values compared to infrequent raters.

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* [402, 565]. 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 [565, 566]. 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 [309] 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 [565], 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 [309] that the use of straightforward RMSE measures on the observed ratings often correlate quite well with the precision on all items.

Another source of evaluation bias is the fact that user interests may evolve with time. As a result, the performance on a hold-out set might not reflect future performance. Although it is not a perfect solution, the use of *temporal* divisions between training and test ratings seems like a reasonable choice. Even though temporal division results in training and testing tests with somewhat different distributions, it also reflects the real-world setting more closely. In this sense, the Netflix Prize contest again provides an excellent model of realistic evaluation design. Several other variations of temporal methods in the evaluation process are discussed in [335].

7.6.1 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 (i.e., matrix F in section 3.6.4.6), 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., $10^7 \times 10^5$), 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.

7.7 Summary

The evaluation of recommender systems is crucial in order to obtain a clear idea about the quality of different algorithms. The most direct method of measuring the effectiveness of a recommender system is to compute the conversion rate at which recommended items are converted to actual usages. This can be done through either user studies or online studies. Such studies are often difficult for researchers and practitioners because of the difficulty in obtaining access to the relevant infrastructure with large groups of users. Offline methods have the advantage that they can be used with multiple historical data sets. In such cases, it is dangerous to use accuracy as the only criterion, because maximizing accuracy does not always lead to long-term maximization of conversion rates. A variety of criteria, such as coverage, novelty, serendipity, stability, and scalability, are used to evaluate the effectiveness of recommender systems.

The proper design of recommender evaluation systems is necessary to ensure that there are no biases in the evaluation process. For example, in a collaborative filtering application, it is important to ensure that all the ratings are evaluated with an out-of-sample approach. A variety of methods such as hold-out and cross-validation are used in order to ensure out-of-sample evaluation. The error is computed with measures, such as the MAE, MSE, and RMSE. In some measures, items are weighted differently to account for their differential importance. In order to evaluate the effectiveness of ranking methods, rank correlation, utility-based measures or usage-based measures may be used. For usage-based measures, precision and recall are used to characterize the trade-off inherent in varying the size of the recommended list. The F1-measure is also used, which is the harmonic mean between the precision and the recall.

7.8 Bibliographic Notes

Excellent discussions on evaluating recommender systems may be found in [246, 275, 538]. Evaluation can be performed either with user studies or with historical data sets. The earliest work on evaluation with user studies may be found in [339, 385, 433]. An early study of evaluation of recommendation algorithms with historical data sets may be found in [98]. Metrics for evaluating recommender systems in the presence of cold-start are discussed in [533]. Controlled experiments for online evaluation in Web applications are discussed in [305]. A general study of online evaluation design is provided in [93]. The evaluation of multi-armed bandit systems is discussed in [349]. A comparison of online recommender systems with respect to human decisions is provided in [317].

The work in [246] presents several variants of accuracy metrics for evaluation. This article is perhaps one of the foremost authorities on the evaluation of recommender systems. The pitfalls of using the *RMSE* as an evaluation measure are presented in [632]. A brief technical note on the relative merits of using *MAE* and *RMSE* as accuracy measures may be found in [141]. The challenges and pitfalls in the use of accuracy metrics are discussed in [418]. Alternative methods for evaluating recommender systems are provided in [459]. A discussion of the importance of novelty is provided in [308]. Online methods for measuring the novelty of a recommender system are provided in [140, 286]. The use of popularity in the measurement of novelty is discussed in [140, 539, 680]. The work in [670] showed that serendipity can be achieved in a recommender system with the help of user labeling. Metrics for serendipity evaluation are discussed in [214, 450]. The work in [214] also studies the use of coverage metrics. Diversity metrics are discussed in [560]. The impact of recommender systems on sales diversity is discussed in [203]. Robustness and stability metrics for recommender systems are discussed in [158, 329, 393, 444]. A study of the evaluation of classification systems may be found in [18, 22]. The discussions in these books provide an understanding of the standard techniques used, such as hold-out and cross-validation.

Rank correlation methods are discussed in [298, 299]. The normalized distance preference measure is discussed in [505]. The R-score for the utility-based evaluation of rankings is discussed in [98]. The NDCG measure is discussed in [59]. The lift index is discussed in [361], whereas the average reciprocal hit rate (ARHR) is proposed in [181]. A discussion of ROC curves in the context of classification may be found in [195], although the same ideas are also applicable to the case of recommender systems. The use of customer-specific and global ROC curves is discussed in [533].

One limitation of recommender systems is that the values on the ratings are related to their relative frequency and that missing items are often in the long tail. Therefore, the use of cross-validation or hold-out mechanisms leads to a selection bias against less frequent items. A number of recent methods for correcting for missing item bias are discussed in [402, 564–566]. The approach in [565] proposes the use of different assumptions for the relevant and non-relevant items, in terms of deciding which ratings are missing. A training algorithm is also designed in [565] based on these assumptions. A temporal framework for realistic evaluation is discussed in [335]. Recommender systems also need to be evaluated somewhat differently in various settings, such as in the presence of specific contexts. These contexts could include time, location, or social information. An evaluation framework for recommender systems in the context of temporal data is provided in [130]. A recent workshop that was devoted exclusively to recommender systems' evaluation may be found in [4].

7.9 Exercises

1. Suppose that a merchant knows the amount of profit q_i made on the sale of the ith item. Design an error metric for a collaborative filtering system that weights the importance of each item with its profit.

2. Suppose that you designed an algorithm for collaborative filtering and found that it was performing poorly on ratings with value 5, but it was performing well on the other ratings. You used this insight to modify your algorithm and tested the algorithm again. Discuss the pitfalls with the second evaluation. Relate your answer to why Netflix chose to separate the quiz set and the test set in the Netflix Prize data set.

3. Implement an algorithm for constructing the ROC and the precision-recall curves.

4. Suppose you have an implicit feedback data set in which the ratings are unary. Would an ROC curve provide more meaningful results or would the *RMSE* metric?

5. Consider a user John, for whom you have hidden his ratings for *Aliens* (5), *Terminator* (5), *Nero* (1), and *Gladiator* (6). The values in brackets represent his hidden ratings, and higher values are better. Now consider a scenario where the recommender system ranks these movies in the order *Terminator*, *Aliens*, *Gladiator*, *Nero.*

(a) Compute the Spearman rank correlation coefficient as a measure of recommendation ranking quality.

(b) Compute the Kendall rank correlation coefficient as a measure of ranking quality.

6. For the problem in Exercise 5, John's utility for a movie j is given by $\max\{r_{ij} - 3, 0\}$, where r_{ij} is his rating.

(a) Under this utility assumption, compute the R-score specific to John. Assume a half-life value of $\alpha = 1$.

(b) For the same utility assumption, compute the component of the discounted cumulative gain (DCG) specific to John, if there are a total of 10 users in the system.

7. For the problem in Exercise 5, assume that the only hidden ratings belong to John, and the predicted ratings from the recommender system are *Aliens* (4.3), *Terminator* (5.4), *Nero* (1.3), and *Gladiator* (5). The values in brackets represent the predicted ratings.

(a) Compute the MSE of the predicted ratings.

(b) Compute the MAE of the predicted ratings.

(c) Compute the RMSE of the predicted ratings.

(d) Compute the normalized MAE and RMSE, assuming that all ratings lie in the range $\{1 \ldots 6\}$.