Journal of Machine Learning Research 10 (2009) 2935-2962 Submitted 11/09; Published 12/09

A Survey of Accuracy Evaluation Metrics of Recommendation Tasks

Asela Gunawardana ASELAG@MICROSOFT.COM
Microsoft Research
One Microsoft Way
Redmond, WA 98052-6399, USA

Guy Shani SHANIGU@BGU.AC.IL
Information Systems Engineering
Ben Gurion University
Beer Sheva, Israel

Editor: Lyle Ungar

Abstract

Recommender systems are now popular both commercially and in the research community, where many algorithms have been suggested for providing recommendations. These algorithms typically perform differently in various domains and tasks. Therefore, it is important from the research perspective, as well as from a practical view, to be able to decide on an algorithm that matches the domain and the task of interest. The standard way to make such decisions is by comparing a number of algorithms offline using some evaluation metric. Indeed, many evaluation metrics have been suggested for comparing recommendation algorithms. The decision on the proper evaluation metric is often critical, as each metric may favor a different algorithm. In this paper we review the proper construction of offline experiments for deciding on the most appropriate algorithm. We discuss three important tasks of recommender systems, and classify a set of appropriate well known evaluation metrics for each task. We demonstrate how using an improper evaluation metric can lead to the selection of an improper algorithm for the task of interest. We also discuss other important considerations when designing offline experiments.

Keywords: recommender systems, collaborative filtering, statistical analysis, comparative studies

1. Introduction

Recommender systems can now be found in many modern applications that expose the user to a huge collections of items. Such systems typically provide the user with a list of recommended items they might prefer, or supply guesses of how much the user might prefer each item. These systems help users to decide on appropriate items, and ease the task of finding preferred items in the collection.

For example, the DVD rental provider Netflix[1] displays predicted ratings for every displayed movie in order to help the user decide which movie to rent. The online book retailer Amazon[2] provides average user ratings for displayed books, and a list of other books that are bought by users who buy a specific book. Microsoft provides many free downloads for users, such as bug fixes, products and so forth. When a user downloads some software, the system presents a list

1. This can be found at `www.netflix.com`.
2. This can be found at `www.amazon.com`.

of additional items that are downloaded together. All these systems are typically categorized as recommender systems, even though they provide diverse services.

In the past decade, there has been a vast amount of research in the field of recommender systems, mostly focusing on designing new algorithms for recommendations. An application designer who wishes to add a recommendation system to her application has a large variety of algorithms at her disposal, and must make a decision about the most appropriate algorithm for her application. Typically, such decisions are based on offline experiments, comparing the performance of a number of candidate algorithms over real data. The designer can then select the best performing algorithm, given structural constraints. Furthermore, most researchers who suggest new recommendation algorithms also compare the performance of their new algorithm to a set of existing approaches. Such evaluations are typically performed by applying some evaluation metric that provides a ranking of the candidate algorithms (usually using numeric scores).

Many evaluation metrics have been used to rank recommendation algorithms, some measuring similar features, but some measuring drastically different quantities. For example, methods such as the Root of the Mean Square Error (RMSE) measure the distance between predicted preferences and true preferences over items, while the Recall method computes the portion of favored items that were suggested. Clearly, it is unlikely that a single algorithm would outperform all others over all possible methods.

Therefore, we should expect different metrics to provide different rankings of algorithms. As such, selecting the proper evaluation metric to use has a crucial influence on the selection of the recommender system algorithm that will be selected for deployment. This survey reviews existing evaluation metrics, suggesting an approach for deciding which evaluation metric is most appropriate for a given application.

We categorize previously suggested recommender systems into three major groups, each corresponding to a different *task*. The first obvious task is to recommend a set of good (interesting, useful) items to the user. In this task it is assumed that all good items are interchangeable. A second, less discussed, although highly important task is utility optimization. For example, many e-commerce websites use a recommender system, hoping to increase their revenues. In this case, the task is to present a set of recommendations that will optimize the retailer revenue. Finally, a very common task is the prediction of user opinion (e.g., rating) over a set of items. While this may not be an explicit act of recommendation, much research in recommender systems focuses on this task, and so we address it here.

For each such task we review a family of common evaluation metrics that measure the performance of algorithms on that task. We discuss the properties of each such metric, and why it is most appropriate for a given task.

In some cases, applying incorrect evaluation metrics may result in selecting an inappropriate algorithm. We demonstrate this by experimenting with a wide collection of data sets, comparing a number of algorithms using various evaluation metrics, showing that the metrics rank the algorithms differently.

We also discuss the proper design of an offline experiment, explaining how the data should be split, which measurements should be taken, how to determine if differences in performance are statistically significant, and so forth. We also describe a few common pitfalls that may produce results that are not statistically sound.

The paper is structured as follows: we begin with some necessary background on recommender approaches (Section 2). We categorize recommender systems into a set of three tasks in Section 3.

We then discuss evaluation protocols, including online experimentation, offline testing, and statistical significance testing of results in Section 4. We proceed to review a set of existing evaluation metrics, mapping them to the appropriate task (Section 5). We then provide (Section 6) some examples of applying different metrics to a set of algorithms, resulting in questionable rankings of these algorithms when inappropriate measures are used. Following this, we discuss some additional relevant topics that arise (Section 7) and some related work (Section 8), and then conclude (Section 9).

2. Algorithmic Approaches

There are two dominant approaches for computing recommendations for the *active user*—the user that is currently interacting with the application and the recommender system. First, the *collaborative filtering* approach (Breese et al., 1998) assumes that users who agreed on preferred items in the past will tend to agree in the future too. Many such methods rely on a matrix of user-item ratings to predict unknown matrix entries, and thus to decide which items to recommend.

A simple approach in this family (Konstan et al., 2006), commonly referred to as *user based collaborative filtering*, identifies a neighborhood of users that are similar to the *active user*. This set of neighbors is based on the similarity of observed preferences between these users and the active user. Then, items that were preferred by users in the neighborhood are recommended to the active user. Another approach (Linden et al., 2003), known as *item based collaborative filtering* recommends items also prefered by users that prefer a particular *active item* to other users that also prefer that active item. In collaborative filtering approaches, the system only has access to the item and user identifiers, and no additional information over items or users is used. For example, websites that present recommendations titled "users who preferred this item also prefer" typically use some type of collaborative filtering algorithm.

A second popular approach is the *content-based* recommendation. In this approach, the system has access to a set of item features. The system then learns the user preferences over features, and uses these computed preferences to recommend new items with similar features. Such recommendations are typically titled "similar items". User's features, if available, such as demographics (e.g., gender, age, geographic location) can also provide valuable information.

Each approach has advantages and disadvantages, and a multitude of algorithms from each family, as well as a number of hybrid approaches have been suggested. This paper, though, makes no distinction between the underlying recommendation algorithms when evaluating their performance. Just as users should not need to take into account the details of the underlying algorithm when using the resulting recommendations, it is inappropriate to select different evaluation metrics for different recommendation approaches. In fact, doing so would make it difficult to decide which approach to employ in a particular application.

3. Recommender Systems Tasks

Providing a single definition for recommender systems is difficult, mainly because systems with different objectives and behaviors are grouped together under that name. Below, we categorize recommender systems into three classes, based on the recommendation task that they are designed for McNee et al. (2006). In fact, there have been several previous attempts to classify existing recommenders (see, e.g., Montaner et al. 2003 and Schafer et al. 1999). We, however, are interested

in the proper evaluation of such algorithms, and our classification is derived from that goal. While there may be recommender systems that do not fit well into the classes that we suggest, we believe that the vast majority of the recommender systems attempt to achieve one of these tasks, and can thus be classified as we suggest.

3.1 Recommending Good Items

Perhaps the most common task of recommendation engines is to recommend good items to users (Herlocker et al., 2004; McNee et al., 2006). Such systems typically present a list of items that the user is predicted to prefer. The user can then select (add to the shopping basket, view, ...) one or more of the suggested items. There are many examples of such systems. In the Amazon website, for instance, when the user is looking at an item, the system presents below a list of other items that the user may be interested in. Another example can be found in Netflix—when a user adds a movie to her queue, the system displays a list of other recommended movies that the user may want to add to the queue too. There are several considerations when creating recommendation lists. We identify below two sub-tasks that comply with different requirements.

3.1.1 RECOMMENDING SOME GOOD ITEMS

In this sub-task, we make the assumption that there is a large number of good items that may appeal to the user, and the user does not have enough resources (time, money) to select all items. In this case we can only present a part of the preferred item set. Thus, it is likely that many preferred items will be missing from the list. In this sub-task, it is more important not to present any disliked item than to find all the good items.

This is typically the case in recommender systems that suggest media items, such as movies, books, or news items. In all these cases the number of alternatives is huge, and the user cannot possibly watch all the recommended movies, or read all the relevant books.

3.1.2 RECOMMENDING ALL GOOD ITEMS

A less popular case is when the system should recommend all important items. Examples of such systems are recommenders that predict which scientific papers should be cited, or legal databases (Herlocker et al., 2004; McNee et al., 2006), where it is important not to overlook any possible case. In this sub-task, the system can present longer lists of items, trying to avoid missing a relevant item.

3.2 Optimizing Utility

With the rise of e-commerce websites, another recommendation task became highly important—maximizing the profits of the website. Online retailers are willing to invest in recommender systems hoping to increase their revenue. There are many ways by which a recommender system can increase revenue. The simplest way is through cross-selling; by suggesting additional items to the users, we increase the probability that the user will buy more than he originally intended. In an online news provider, where most revenue comes from display advertisements, the system can increase profit by keeping the users in the website for longer time periods, as the performance of an advertising campaign is often measured in terms of "x-minute reach," which is the number of consumers in a particular market that are exposed to the ad for x minutes. In such cases, it is in the best interest of the system to suggest items in order to lengthen the session. In a subscription

service, where revenue comes from users paying a regular subscription, the goal may be to allow users to easily reach items of interest. In this case, the system should suggest items such that the user reaches items of interest with minimal effort.

The utility function to be optimized can be more complicated, and in particular, may be a function of the entire set of recommendations and their presentation to the user. For example, in pay-per-click search advertising, the system must recommend advertisements to be displayed on search results pages. Each advertiser bids a fixed amount that is paid only when the user clicks on their ad. If we wish to optimize the expected system profit, both the bids and the probability that the user will click on each ad must be taken into account. This probability depends on the relevance of each ad to the user and the placement of the different ads on the page. Since the different ads displayed compete for the user's attention, the utility function depends on the entire set of ads displayed, and is not additive over the set (Gunawardana and Meek, 2008).

In all of these cases, it may be suboptimal to suggest items based solely on their predicted rating. While it is certainly beneficial to recommend relevant items, other considerations are also important. For example, in the e-commerce scenario, given two items that the system perceives as equally relevant, suggesting the item with the higher profit can further increase revenue. In the online news agency case, recommending longer stories may be beneficial, because reading them will keep the user in the website longer. In the subscription service, recommending items that are harder for the user to reach without the recommender system may be beneficial.

Another common practice of recommendation systems is to suggest recommendations that provide the most "value" to the user. For example, recommending popular items can be redundant, as the user is probably already familiar with them. A recommendation of a preferred, yet unknown item can provide a much higher value for the user.

Such approaches can be viewed as instances of providing recommendations that maximize some utility function that assigns a value to each recommendation. Defining the correct utility function for a given application can be difficult (Braziunas and Boutilier, 2005), and typically system designers make simplifying assumptions about the user utility function. In the e-commerce case the utility function is typically the profit resulting from recommending an item, and in the news scenario the utility can be the expected time for reading a news item, but these choices ignore the effect of the resulting recommendations on long-term profits. When we are interested in novel recommendations, the utility can be the log of the inverse popularity of an item, modeling the amount of new information in a recommended item (Shani et al., 2005), but this ignores other aspects of user-utility such as the diversity of recommendations.

In fact, it is possible to view many recommendation tasks, such as providing novel or serendipitious recommendations as maximizing some utility function. Also, the "recommend good items" of the previous section can be considered as optimizing for a utility function assigning a value of 1 to each successful recommendation. In this paper, due to the popularity of the former task, we choose to keep the two tasks distinct.

3.3 Predicting Ratings

In some cases, a system is required to predict the user ratings over a given set of items. For example, in the Netflix website, when the user is browsing the list of new releases, the system assigns a predicted rating for each movie. In CNET,[3] a website offering electronic product reviews, users can

3. This can be found at `www.cnet.com`.

search for, say, laptops that cost between $400 and $700. The system adds to some laptops in the list an automatically computed rating, based on the laptop features.

It is arguable whether this task is indeed a recommendation task. However, many researchers in the recommendations system community attempting to find good algorithms for this task. Examples include the Netflix competition, which was warmly embraced by the research community, and the numerous papers on predicting ratings on the Netflix or MovieLens[4] data sets.

While such systems do not provide lists of recommended items, predicting that the user will rate an item highly can be considered an act of recommendation. Furthermore, one can view a predicted high rating as a recommendation to use the item, and a predicted low rating as a recommendation to avoid the item. Indeed, it is common practice to use predicted ratings to generate a list of recommendations. Below, we will present several arguments of cases where this common practice may be undesirable.

4. Evaluation Protocols

We now discuss an experimental protocol for evaluating and choosing recommendation algorithms. We review several requirements to ensure that the results of the experiments are statistically sound. We also describe several common pitfalls in such experimental settings. This section reviews the evaluation protocols in related areas such as machine learning and information retrieval, highlighting practices relevant to evaluating recommendation systems. The reader is referred to publications in these fields for more detailed discussions (Salzberg, 1997; Demšar, 2006; Voorhees, 2002a).

We begin by discussing online experiments, which can measure the real performance of the system. We then argue that offline experiments are also crucial, because online experiments are costly in many cases. Therefore, the bulk of the section discusses the offline experimental setting in detail.

4.1 Online Evaluation

In the recommendation and utility optimization tasks, the designer of the system wishes to influence the behavior of users. We are therefore interested in measuring the change in user behavior when interacting with different recommendation systems. For example, if users of one system follow the recommendations more often (in the case of the "recommend good items" task), or if the utility gathered from users of one system exceeds utility gathered from users of the other system (in the utility optimization task), then we can conclude that one system is superior to the other, all else being equal. In the case of ratings prediction tasks, the goal is to provide information to support user browsing and search. Once again, the value of such predictions can depend on a variety of factors such as the user's intent (e.g., how specific their information needs are, how much novelty vs. how much risk they are seeking), the user's context (e.g., what items they are already familiar with, how much they trust the system), and the interface through which the predictions are presented.

For this reason, many real world systems employ an online testing system (Kohavi et al., 2009), where multiple algorithms can be compared. Typically, such systems redirect a small percentage of the traffic to each different recommendation engine, and record the users interactions with the different systems. There are a few considerations that must be made when running such tests. For example, it is important to sample (redirect) users randomly, so that the comparisons between

4. This can be found at `www.movielens.org`.

alternatives are fair. It is also important to single out the different aspects of the recommenders. For example, if we care about algorithmic accuracy, it is important to keep the user interface fixed. On the other hand, if we wish to focus on a better user interface, it is best to keep the underlying algorithm fixed.

However, in a multitude of cases, such experiments are very costly, since creating online testing systems may require much effort. Furthermore, we would like to evaluate our algorithms before presenting their results to the users, in order to avoid a negative user experience for the test users. For example, a test system that provides irrelevant recommendations, may discourage the test users from using the real system ever again. Finally, designers that wish to add a recommendation system to their application before its deployment do not have an opportunity to run such tests.

For these reasons, it is important to be able to evaluate the performance of algorithms in an offline setting, assuming that the results of these offline tests correlate well with the online behavior of users.

4.2 Offline Experimental Setup

As described above, the goal of the offline evaluation is to filter algorithms so that only the most promising need undergo expensive online tests. Thus, the data used for the offline evaluation should match as closely as possible the data the designer expects the recommender system to face when deployed online. Care must be exercised to ensure that there is no bias in the distributions of users, items and ratings selected. For example, in cases where data from an existing system (perhaps a system without a recommender) is available, the experimenter may be tempted to pre-filter the data by excluding items or users with low counts, in order to reduce the costs of experimentation. In doing so, the experimenter should be mindful that this involves a trade-off, since this introduces a systematic bias in the data. If necessary, randomly sampling users and items may be a preferable method for reducing data, although this can also introduce other biases into the experiment (e.g., this could tend to favor algorithms that work better with more sparse data).

In order to evaluate algorithms offline, it is necessary to simulate the online process where the system makes predictions or recommendations, and the user corrects the predictions or uses the recommendations. This is usually done by recording historical user data, and then hiding some of these interactions in order to simulate the knowledge of how a user will rate an item, or which recommendations a user will act upon.

There are a number of ways to choose the ratings/selected items to be hidden. Once again, it is preferable that this choice be done in a manner that simulates the target application as closely as possible. We discuss these concerns explicitly for the case of selecting used items for hiding in the evaluation of recommendation tasks, and note that the same considerations apply when selecting ratings to hide for evaluation of ratings prediction tasks.

Our goal is to simulate sets of past user selections that are representative of what the system will face when deployed. Ideally, if we have access to time-stamps for user selections, we can randomly sample test users, randomly sample a time just prior to a user action, hide all selections (of all users) after that instant, and then attempt to recommend items to that user. This protocol requires changing the set of given information prior to each recommendation, which can be computationally quite expensive. A cheaper alternative is to sample a set of test users, then sample a single test time, and hide all items after the sampled test time for each test user. This simulates a situation where the recommender system is “trained” as of the test time, and then makes recommendations without

taking into account any new data that arrives after the test time. Another alternative is to sample a test time for each test user, and hide the test user's items after that time, without maintaining time consistency across users. This effectively assumes that it is the sequence in which items are selected, and not the absolute times when they are selected that is important. A final alternative is to ignore time; We sample a set of test users, then sample the number n_a of items to hide for each user a, then sample n_a items to hide. This assumes that the temporal aspects of user selections are unimportant. All three of the latter alternatives partition the data into a single training set and single test set. It is important to select an alternative that is most appropriate for the domain and task of interest, rather than the most convenient one.

A common protocol used in many research papers is to use a fixed number of known items or a fixed number of hidden items per test user (so called "given n" or "all but n" protocols). This protocol is useful for diagnosing algorithms and identifying in which cases they work best. However, when we wish to make decisions on the algorithm that we will use in our application, we must ask ourselves whether we are truly interested in presenting recommendations for users who have rated exactly n items, or are expected to rate exactly n items more. If that is not the case, then results computed using these protocol have biases that make them difficult to use in predicting the outcome of using the algorithms online.

The evaluation protocol we suggest above generates a test set (Duda and Hart, 1973) which is used to obtain held-out estimates for algorithm performance, using performance measures which we discuss below. Another popular alternative is to use cross-validation (Stone, 1974), where the data is divided into a number of partitions, and each partition in turn is used as a test set. The advantages of the cross-validation approach are to allow the use of more data in ranking algorithms, and to take into account the effect of training set variation. In the case of recommender systems, the held-out approach usually yields enough data to make reliable decisions. Furthermore, in real systems, the problem of variation in training data is avoided by evaluating systems trained on the historical data specific to the task at hand. In addition, there is a risk that since the results on the different data partitions are not independent of each other, pooling the results across partitions for ranking algorithms can lead to statistically unjustified decisions (Bengio and Grandvalet, 2004).

4.3 Making Reliable Choices

When choosing between algorithms, it is important that we can be confidant that the algorithm that we choose will also be a good choice for the yet unseen data the system will be faced with in the future. As we explain above, we should exercise caution in choosing the data so that it would be most similar to the online application. Still, there is a possibility that the algorithm that performed best on this test set did so because the test set was fortuitously suitable for that algorithm. To reduce the possibility of such statistical mishaps, we must perform significance testing on the results.

Typically we compute a significance level or p-value—the probability that the obtained results were due to luck. Generally, we will reject the null hypothesis that algorithm A is no better than algorithm B if the p-value is above 0.05 (or below 95% confidence). That is, if the probability that the observed ranking is achieved by chance exceeds 0.05. More stringent significance levels (e.g., 0.01 or even lower) can be used in cases where the cost of making the wrong choice is higher.

In order to perform a significance test that algorithm A is indeed better than algorithm B, we require the results of several independent experiments comparing A and B. The protocol we have chosen in generating our test data ensures that we will have this set of results. Assuming that test

users are drawn independently from some population, the performance measures of the algorithms for each test user give us the independent comparisons we need. However, when recommendations or predictions of multiple items are made to the same user, it is unlikely that the resulting per-item performance metrics are independent. Therefore, it is better to compare algorithms on a per-user case. Approaches for use when users have not been sampled independently also exist, and attempt to directly model these dependencies (see, e.g., Larocque et al. 2007). Care should be exercised when using such methods, as it can be difficult to verify that the modeling assumptions that they depend on hold in practice.

Given such paired per-user performance measures for algorithms A and B the simplest test of significance is the sign test (Demšar, 2006). In this test, we count the number of users for whom algorithm A outperforms algorithm B (n_A) and the number of users for whom algorithm B outperforms algorithm A (n_B). The probability that A is not truly better than B is estimated as the probability of at least n_A out of $n_A + n_B$ 0.5-probability Binomial trials succeeding (that is, n_A out of $n_A + n_B$ fair coin-flips coming up "heads").

$$pr(successes \geq n_A | A = B) = 0.5^{n_A+n_B} \sum_{k=n_A}^{n_a+n_B} \frac{(n_A+n_B)!}{k!(n_i+n_B-k)!}.$$

The sign test is an attractive choice due to its simplicity, and lack of assumptions over the distribution of cases. Still this test may lead to mislabeling of significant results as insignificant when the number of test points is small. In these cases, the more sophisticated Wilcoxon signed rank test can be used (Demšar, 2006). As mentioned in Section 4.2, cross-validation can be used to increase the amount of data, and thus the significance of results, but in this case the results obtained on the cross-validated test sets are no longer independent, and care must be exercised to ensure that our decisions account for this (Bengio and Grandvalet, 2004). Also, model-based approaches (e.g., Goutte and Gaussier, 2005) may be useful when the amount of data is small, but once again, care must be exercised to ensure that the model assumptions are reasonable for the application at hand.

Another important consideration is the effect of evaluating multiple versions of algorithms. For example, an experimenter might try out several variants of a novel recommender algorithm and compare them to a baseline algorithm until they find one that passes a sign test at the $p = 0.05$ level and therefore infer that their algorithm improves upon the baseline with 95% confidence. However, this is not a valid inference. Suppose the experimenter evaluated ten different variants all of which are statistically the same as the baseline. If the probability that any one of these trials passes the sign test mistakenly is $p = 0.05$, the probability that at least one of the ten trials passes the sign test mistakenly is $1-(1-0.05)^{20} = 0.40$. This risk is colloquially known as "tuning to the test set" and can be avoided by separating the test set users into two groups—a development (or tuning) set, and an evaluation set. The choice of algorithm is done based on the development test, and the validity of the choice is measured by running a significance test on the evaluation set.

A similar concern exists when ranking a number of algorithms, but is more difficult to circumvent. Suppose the best of $N+1$ algorithms is chosen on the development test set. We can have a confidence $1-p$ that the chosen algorithm is indeed the best, if it outperforms the N other algorithms on the evaluation set with significance $1-(1-p)^{1/N}$. This is known as the Bonferroni correction, and should be used when pair-wise significant tests are used multiple times. Alternatively, the Friedman test for ranking can be used (Demšar, 2006).

5. Evaluating Tasks

An application designer that wishes to employ a recommendation system typically knows the purpose of the system, and can map it into one of the tasks defined above—recommendation, utility optimization, and ratings prediction. Given such a mapping, the designer must now decide which evaluation metric to use in order to rank a set of candidate recommendation algorithms. It is important that the metric match the task, to avoid an inappropriate ranking of the candidates.

Below we provide an overview of a large number of evaluation metrics that have been suggested in the recommendation systems literature. For each such metric we identify its important properties and explain why is it most appropriate for the given task. For each task we also explain a possible evaluation scenario that can be used to evaluate the various algorithms.

5.1 Predicting Ratings

In this task, the system must provide a set of predicted ratings, and is evaluated on the accuracy of these predictions. This is the most common scenario in the evaluation of regression and classification algorithms in the machine learning and statistics literature (Duda and Hart, 1973; Stone, 1974; Bengio and Grandvalet, 2004). Many evaluation metrics that originated in that literature have been applied here.

Most notably, the Root of the Mean Square Error (RMSE) is a popular method for scoring an algorithm. If $p_{i,j}$ is the predicted rating for user i over item j, and $v_{i,j}$ is the true rating, and $K = \{(i, j)\}$ is the set of hidden user-item ratings then the RMSE is defined as:

$$\sqrt{\frac{\sum_{(i,j)\in K}(p_{i,j} - v_{i,j})^2}{n}}.$$

Other variants of this family are the Mean Square Error (which is equivalent to RMSE) and Mean Average Error (MAE), and Normalized Mean Average Error (NMAE) (Herlocker et al., 2004). RMSE tends to penalize larger errors more severely than the other metrics, while NMAE normalizes MAE by the range of the ratings for ease of comparing errors across domains.

RMSE is suitable for the prediction task, because it measures inaccuracies on all ratings, either negative or positive. However, it is most suitable for situations where we do not differentiate between errors. For example, in the Netflix rating prediction, it may not be as important to properly predict the difference between 1 and 2 stars as between 2 and 3 stars. If the system predicts 2 instead of the true 1 rating, it is unlikely that the user will perceive this as a recommendation. However, a predicted rating of 3 may seem like an encouragement to rent the movie, while a prediction of 2 is typically considered negative. It is arguable that the space of ratings is not truly uniform, and that it can be mapped to a uniform space to avoid such phenomena.

5.2 Recommending Good Items

For the task of recommending items, typically we are only interested in binary ratings, that is, either the item was selected (1) or not (0). Compared to ratings data sets, where users typically rate only a very small number of items, making the data set extremely sparse, binary selection data sets are dense, as each item was either selected or not by the user. An example of such data sets are news story click streams, where we set a value of 1 for each item that was visited, and a value of 0

	Recommended	Not recommended
Preferred	True-Positive (tp)	False-Negative (fn)
Not preferred	False-Positive (fp)	True-Negative (tn)

Table 1: Classification of the possible result of a recommendation of an item to a user.

elsewhere. The task is to provide, given an existing list of items that were viewed, a list of additional items that the user may want to visit.

As we have explained above, these scenarios are typically not symmetric. We are not equally interested in good and bad items; the task of the system is to suggest good items, not to discourage the use of bad items. We can classify the results of such recommendations using Table 1.

We can now count the number of examples that fall into each cell in the table and compute the following quantities:

$$\begin{aligned} \text{Precision} &= \frac{\#\text{tp}}{\#\text{tp}+\#\text{fp}}, \\ \text{Recall (True Positive Rate)} &= \frac{\#\text{tp}}{\#\text{tp}+\#\text{fn}}, \\ \text{False Positive Rate (1 - Specificity)} &= \frac{\#\text{fp}}{\#\text{fp}+\#\text{tn}}. \end{aligned}$$

Typically we can expect a trade off between these quantities—while allowing longer recommendation lists typically improves recall, it is also likely to reduce the precision. In some applications, where the number of recommendations that are presented to the user is not preordained, it is therefore preferable to evaluate algorithms over a range of recommendation list lengths, rather than using a fixed length. Thus, we can compute curves comparing precision to recall, or true positive rate to false positive rate. Curves of the former type are known simply as precision-recall curves, while those of the latter type are known as a Receiver Operating Characteristic[5] or ROC curves.

While both curves measure the proportion of preferred items that are actually recommended, precision-recall curves emphasize the proportion of recommended items that are preferred while ROC curves emphasize the proportion of items that are not preferred that end up being recommended.

We should select whether to use precision-recall or ROC based on the properties of the domain and the goal of the application; suppose, for example, that an online video rental service recommends DVDs to users. The precision measure describes what proportion of their recommendations were actually suitable for the user. Whether the unsuitable recommendations represent a small or large fraction of the unsuitable DVDs that could have been recommended (that is, the false positive rate) may not be as relevant.

On the other hand, consider a recommender system for an online dating site. Precision describes what proportion of the suggested pairings for a user result in matches. The false positive rate describes what proportion of unsuitable candidates are paired with the active user. Since presenting unsuitable candidates can be especially undesirable in this setting, the false positive rate could be the most important factor.

5. A reference to their origins in signal detection theory.

Given two algorithms, we can compute a pair of such curves, one for each algorithm. If one curve completely dominates the other curve, the decision about the winning algorithm is easy. However, when the curves intersect, the decision is less obvious, and will depend on the application in question. Knowledge of the application will dictate which region of the curve the decision will be based on. For example, in the "recommend some good items" task it is likely that we will prefer a system with a high precision, while in the "recommend all good items" task, a higher recall rate is more important than precision.

Measures that summarize the precision recall of ROC curve such as F-measure (Rijsbergen, 1979) and the area under the ROC curve (Bamber, 1975) are useful for comparing algorithms independently of application, but when selecting an algorithm for use in a particular task, it is preferable to make the choice based on a measure that reflects the specific needs at hand.

5.2.1 PRECISION-RECALL AND ROC FOR MULTIPLE USERS

When evaluating precision-recall or ROC curves for multiple test users, a number of strategies that can be employed in aggregating the results. The simplest is to aggregate the hidden ratings from the test set into a set of user-item pairs, generate a ranked list of user-item pairs by combining the recommendation lists for the test users, and then compute the precision-recall or ROC curve on this aggregated data.

This aggregation process assumes that we have a means of comparing recommendations made to different users in order to combine the recommendation lists into a single ranked list. Computing ROC curves in this manner treats the recommendations of different items to each user as being independent detection or classification tasks, and the resulting curve is termed a global ROC (GROC) curve (Schein et al., 2002).

A second approach is to compute the precision and recall (or true positive rate and false positive rate) at each recommendation list length N for each user, and then compute the average precision and recall (or true positive rate and false positive rate) at each N(Sarwar et al., 2000). The resulting curves are particularly valuable because they prescribe a value of N for each achievable precision and recall (or true positive rate and false positive rate), and conversely, can be used to estimate performance at a given N. Thus, this approach is useful in the "recommend some good items" scenario, where one important decision is the length of the recommendation list, by comparing performances along different candidate points along the curves. An ROC curve obtained in this manner is termed a Customer ROC (CROC) curve (Schein et al., 2002).

A third approach is to compute a precision-recall curve (or ROC curve) for each user and then average the resulting curves over users. This is the usual manner in which precision-recall curves are computed in the information retrieval community, and in particular in the influential TREC competitions (Voorhees, 2002b). This method is more relevant in the "recommend all good items" sub-task, if the system provides the user with all available recommendations and the user then scans the list linearly, marking each scanned item as relevant or not. The system can then compute the precision of the items scanned so far, and use the precision recall curve to give the user an estimate of what proportion of the good items have yet to be found.

5.3 Optimize Utility

Estimating the utility of a list of recommendations requires a model of the way users interact with the recommendations. For example, if a movie recommender system presents the DVD cover im-

ages of the top five recommendations prominently arranged horizontally across the top of the screen, the user will probably observe them all and select the items of interest. However, if all the recommendations are presented in a textual list several pages long, the user will probably scan down the list and abandon their scan at some point. In the first case, utility delivered by the top five recommendations actually selected would be a good estimate of expected utility, while in the second case, we would have to model the way users scan lists.

The half-life utility score of Breese et al. (1998) suggested such a model. It postulates that the probability that the user will select a relevant item drops exponentially down the list.

This approach evaluates an unbounded recommendation list, that potentially contains all the items in the catalog. Given such a list we assume that the user looks at items starting from the top. We then assume that an item at position k has a probability of $\frac{1}{2^{(k-1)/(\alpha-1)}}$ of being viewed, where α is a half life parameter, specifying the location of the item in the list with 0.5 probability of being viewed.

In the binary case of the recommendation task the half-life utility score is computed by:

$$\begin{aligned} R_a &= \sum_j \frac{1}{2^{(idx(j)-1)/(\alpha-1)}}, \\ R &= \frac{\sum_a R_a}{\sum_a R_a^{max}}, \end{aligned}$$

where the summation in the first equation is over the preferred items only, $idx(j)$ is the index of item j in the recommendation list, and R_a^{max} is the score of the best possible list of recommendations for user a.

More generally, we can plug any utility function $u(a, j)$ that assigns a value to a user item pair into the half-life utility score, obtaining the following formula:

$$R_a = \sum_j \frac{u(a, j)}{2^{(idx(j)-1)/(\alpha-1)}}.$$

Now, R_a^{max} is the score for the list of the recommendation where all the observed items are ordered by decreasing utility. In applications where the probability that a user will select the idxth item if it is relevant is known, a further generalization would be to use these known probabilities instead of the exponential decay.

5.4 Fixed Recommendations Lists

When users add movies to their queues in Netflix, the system presents a list of 10 movies that they may like. However, when users choose to see recommendations (by clicking "movies that you will love") the system presents all the possible recommendations. If there are too many recommended movies to fit a single page, the system allows the user to move to the next page of recommendations.

These two different usage scenarios illustrate a fundamental difference between recommendation applications—in the first, the system is allowed to show a small, fixed number of recommendations. In the second, the system provides as many recommendations as it can. Even though the two cases match a single task—the "recommend good items" task—there are several important distinctions that arise. It is important to evaluate the two cases properly.

When the system is required to present a list with a small, fixed size, that is known *a priori*, methods that present curves (precision-recall), or methods that evaluate the entire list (half-life

utility score), become less appropriate. For example, a system may get a relatively high half-life utility score, only due to items that fall outside the fixed list, while another system that selects all the items in the list correctly, and uninteresting items elsewhere, might get a lower score. Precision-recall curves are typically used to help us select the proper list length, where the precision and recall reach desirable values.

Another important difference, is that for a small list, the order of items in the list is less important, as we can assume that the user looks at all the items in the list. Moreover, may of these lists are presented in a horizontal direction, which also reduces the importance of properly ordering the items.

In these cases, therefore, a more appropriate way to evaluate the recommendation system should focus on the first N movies only. In the "recommend good items" task this can be done, for example, by measuring the precision at N—the number of items that are interesting out of the recommended N items. In the "optimize utility" task, we can do so by measuring the aggregated utility (e.g., sum of utility) of the items that are indeed interesting within the N recommendations.

A final case is when we have unlimited recommendation lists in the "recommend good items" scenario, and we wish to evaluate the entire list. In this case, one can use the half-life utility score with a binary utility of 1 when the (hidden) item was indeed selected by the user, and 0 otherwise. In that case, the half-life utility score prefers a recommender system that places interesting items closer to the head of the list, but provides an evaluation for the entire list in a single score.

6. Empirical Evaluation

In some cases, two metrics may provide a different ranking of two algorithms. When one metric is more appropriate for the task at hand, using the other metric may result in selecting the wrong algorithm. Therefore, it is important to choose the appropriate evaluation metric for the task at hand.

In this section we provide some empirical examples of the phenomenon we describe above, that is, where different metrics rank algorithms differently. Below, we present examples where algorithms are ranked differently by two metrics, one of which is more appropriate for the task of interest.

6.1 Data Sets

We selected publicly available data sets which were naturally suited to the different recommendation tasks we have described above. We begin by describing the properties of each data set we used.

6.1.1 PREDICTION TASK

For the prediction task we selected two data sets that contained ratings over items—the Netflix data set and the BookCrossing data set. In both cases, the prediction task is quite natural. Users of both systems may want to browse the collection of movies or books, and we would want to offer these users an estimated rating for the presented items.

Netflix: In 2004, the online movie rental company Netflix[6] announced a competition for improving its recommendation system. For the purpose of the competition, Netflix has released a data set containing $480,000$ users ratings over $17,700$ movies. Ratings are between 1 and 5 stars for each movie. The data set is very sparse—users mostly rated a small fraction of the available movies. In

6. This can be found at `www.netflix.com`.

our experiments, as we are working with simple algorithms, we have reduced the data set to users who rated more than 100 movies, leaving us with 21,179 users, 17,415 movies, and 117 ratings per user on average. Thus, our results are not comparable to results published in the online competition scoreboard.

BookCrossing: The BookCrossing website[7] allows a community of book readers to share their interests in books, and to review and discuss books. Within that system users can provide ratings on the scale of 1 to 10 stars. The specific data set that we used was collected by a 4 week crawl during August and September 2004 (Ziegler et al., 2005). The data set contains 105,283 users and 340,556 books (we used just the subset containing explicit ratings). Average ratings for a user is 10. This data set is even more sparse than the Netflix data set that we used, as there are more items and less ratings per user.

Both data sets share some common properties. First, people watch many movies and read many books, compared with other domains. For example, most people experience with only a handful of laptop computers, and so cannot form an opinion on most laptops. Ratings are also skewed towards positive ratings in both cases, as people are likely to watch movies that they think they will like, and even more so in the case of books, which require a heavier investment of time.

There are also some distinctions between the data sets. Some people feel compelled to share their opinion about books and movies, without asking for a compensation. However, in the Netflix domain, providing ratings makes it easier to navigate the system and rent movies. Therefore, all users of Netflix have an incentive for providing ratings, while only people who like to share their views of books use the BookCrossing system. We can therefore expect that the ratings of the BookCrossing are less representative of the general population of book readers, than the ratings of Netflix user from the general population of DVD renters.

6.1.2 RECOMMENDATION TASK

One instance of the "recommend good items" task is the case where, given a set of items that the user has used (bought, viewed), we wish to recommend a set of items that are likely to be used. Typically, data sets of usage are binary—an item was either used or wasn't used by the user, and the data set is not sparse, because every item is either used or not used by every user. We used here a data set of purchases from supermarket retailer, and a stream of articles that were viewed in a news website.

Belgian retailer: This data set was collected from an anonymous Belgian retail supermarket store, collected over approximately 5 months, in three non-consecutive periods during 1999 and 2000. The data set is divided into baskets, and we cannot detect return users. There are 88,162 baskets, 16,470 distinct items, and 10 items in an average basket. We do not have access for item prices or profits, so we cannot optimize the retailer revenue. Therefore the task is to recommend more items that the user may want to add to the basket.

News click stream: This is a log of click-stream data of an Hungarian online news portal (Bodon, 2003). The data contains 990,002 sessions, 41,270 news stories, and an average of 8 stories for session. The task is, given the news items that a user has read so far, recommend more news items that the user will likely read.

7. This can be found at `www.bookcrossing.com`.

6.1.3 OPTIMIZING UTILITY TASK

Ta-Feng supermarket: A natural example for an application where optimizing utility is important is maximizing the revenues of a retail company. Such companies may provide recommendation for items, hoping that customers following these recommendations will produce higher revenue. In this case, a natural utility function is the revenue (or profit) from the purchase of an item. The Ta-Feng data set (Hsu et al., 2004) contains transaction information collected over 4 months from November, 2000 to February, 2001. There are $32,266$ users and $23,812$ items, where the average number of items bought by a user is 23. In this task, the utility function is the accumulated profit from selling an item—taking into account both the quantity and the profit per item.

6.2 Recommendation Algorithms

As the focus of this survey is on the correct evaluation of recommender systems, and not on sophisticated algorithms for computing recommendation lists, we limit ourselves to a set of very simple collaborative filtering algorithms. We do this because collaborative filtering is by far the most popular recommendation approach, and because we do not believe that it is appropriate to select the evaluation metric based on the recommendation approach (e.g., collaborative filtering vs. content based).

Moreover, we carefully selected algorithms that are better suited for different tasks, so that we could demonstrate that inappropriate choice of evaluation metric can lead to a bad choice of algorithm. As the algorithms that we choose are computationally intensive, we reduced the size of the data set in some cases, in order to reduce the computation time. This should not be done if it was important to realistically simulate the online case. Below, we present the different algorithms and our prior assumptions about their properties.

6.2.1 PEARSON CORRELATION

Typically, the input for a prediction task is a data set consisting of the ratings provided by n users for m items, where $v_{i,j}$ is the rating of user i for item j. Given such a data set, the simplest collaborative filtering method computes the similarity of the active user a to all other users i in the data set, resulting in a score $w(a,i)$. Then, the predicted rating $p_{a,j}$ for a over item j can be computed by:

$$p_{a,j} = \bar{v}_a + \kappa \sum_{i=1}^{n} w(a,i)(v_{i,j} - \bar{v}_i). \tag{1}$$

Perhaps the most popular method for computing the weights $w(a,i)$ is by using the Pearson correlation coefficient (Resnick and Varian, 1997):

$$w(a,i) = \frac{\sum_j (v_{a,j} - \bar{v}_a)(v_{i,j} - \bar{v}_i)}{\sqrt{\sum_j (v_{a,j} - \bar{v}_a)^2 \sum_j (v_{i,j} - \bar{v}_i)^2}}$$

where the summations are only over the items that both a and i have rated. To reduce the computational overhead, we use in Equation 1 a neighborhood of size N.

This method is specifically designed for the prediction task, as it computes only a predicted score for each item of interest. However, in many cases people used this method for the recommendation task. This is typically done by predicting the scores for all possible items, and then ordering the items by decreasing predicted scores.

This popular usage may not be appropriate. For example, in the movie domain people may associate ratings with quality, as opposed to enjoyment, which is dependent on external factors such as mood, time of day, and so forth. As such, 5 stars movies may be complicated, requiring a substantial effort from the viewer. Thus, a user may rent many light effortless romantic comedies, which may only get a score of 3 stars, and only a few 5 star movies. While it is difficult to measure this effect without owning a rental store, we computed the average number of ratings for movies with different average rating (Figure 6.2.1). This figure may suggest that movies with higher ratings are not always watched more often than movies with lower ratings. If our assumption is true, a system that recommends items to add to the rental queue by order of decreasing predicted rating, may not do as well as a system that predicts the probability of adding a movie to the queue directly.

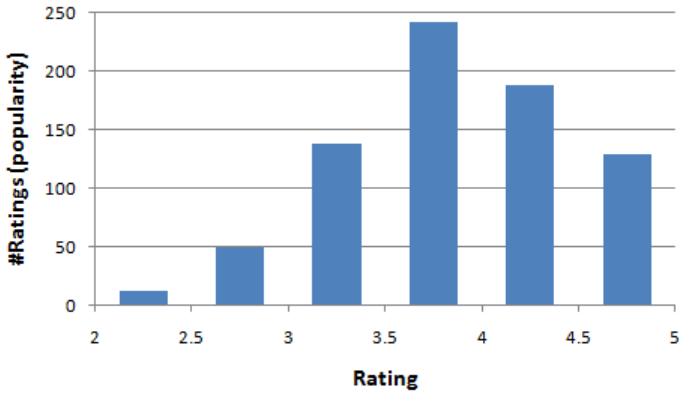


Figure 1: Computing the average number of ratings (popularity) of movies binned given their average ratings.

6.2.2 COSINE SIMILARITY

A second popular collaborative filtering method is the vector similarity metric (Salton, 1971) that measures the cosine angle formed by the two ratings vectors:

$$w(a,i) = \sum_{j \in I_{a,i}} \frac{v_{a,j}}{\sqrt{\sum k \in I_a v_{a,k}^2}} \frac{v_{i,j}}{\sqrt{\sum k \in I_i v_{i,k}^2}}.$$

When computing the cosine similarity, only positive ratings have a role, and negative ratings are discarded. Thus, I_i is the set of items that user i has rated positively and $I_{a,i}$ is the set of items that both users rated positively. Also, the predicted score for a user is computed by:

$$p_{a,j} = \kappa \sum_{i=1}^{n} w(a,i) v_{i,j}.$$

In the case of binary data sets, such as the usage data sets that we selected for the recommendation task, the vector similarity method becomes:

$$w(a,i) = \frac{|I_{a,i}|}{\sqrt{|I_a|} \cdot \sqrt{|I_i|}}$$

where I_a is the set of items that a used, and $I_{a,i}$ is the set of items that both a and i used. The resulting aggregated score can be considered as a non-calibrated measurement of the conditional probability $pr(j|a)$—the probability that user a will choose item j.

In binary usage data sets, the Pearson correlation method would compute similarity using all the items, as each item always has a rating. Therefore, the system would use all the negative "did not use" scores, which typically greatly outnumber the "used" scores. We can therefore expect that Pearson correlation in these cases will result in lower accuracy.

6.2.3 ITEM TO ITEM

The above two methods focused on computing a similarity between users, but another possible collaborative filtering alternative is to focus on the similarity between items. The simplest method for doing so is to use the maximum likelihood estimate for the conditional probabilities of items. Specifically, for the binary usage case, this translates to:

$$pr(j_1|j_2) = \frac{|J_{j_1,j_2}|}{|J_{j_2}|}$$

where J_j is the number of users who used item j, and J_{j_1,j_2} is the number of users that used both j_1 and j_2. While this seems like a very simple estimation, similar estimations are successfully used in deployed commercial applications (Linden et al., 2003).

Typically, an algorithm is given as an input a set of items, and needs to produce a list of recommendations. In that case, we can compute for the conditional probability of each target item given each observed item, and then aggregate the results over the set of given items. In many cases, choosing the maximal estimate has given the best results (Kadie et al., 2002), so we aggregate estimations using a max operator in our experiments.

6.2.4 EXPECTED UTILITY

As optimizing utilities is by far the least explored recommendation task, we choose here to propose a new algorithm that is designed specifically for this task. Intuitively, if the task requires lists that optimize a utility function $u(a, j)$, an obvious method is to order the recommendation by decreasing expected utility:

$$E[j|a] = \tilde{pr}(j|a) \cdot \tilde{u}(a, j)$$

where $\tilde{pr}$ is a conditional probability estimate and $\tilde{u}$ is a utility estimate.

One way to compute the two estimates is by using two different algorithms—a recommendation algorithm for estimating $\tilde{pr}$ and a prediction algorithm for estimating $\tilde{u}$, and then combining their output.

6.3 Experimental Results

Below, we present several examples where different evaluation metrics rank two algorithms differently. We argue that in these cases, using an improper evaluation metric will lead to the selection of an inferior algorithm.

	Netflix	BookCrossing
Pearson	1.07	3.58
Cosine	1.90	4.5

Table 2: RMSE scores for Pearson correlation and Cosine similarity on the Netflix domain (ratings from 1 to 5) and the BookCrossing domain (ratings from 1 to 10).

6.3.1 PREDICTION VS. RECOMMENDATION

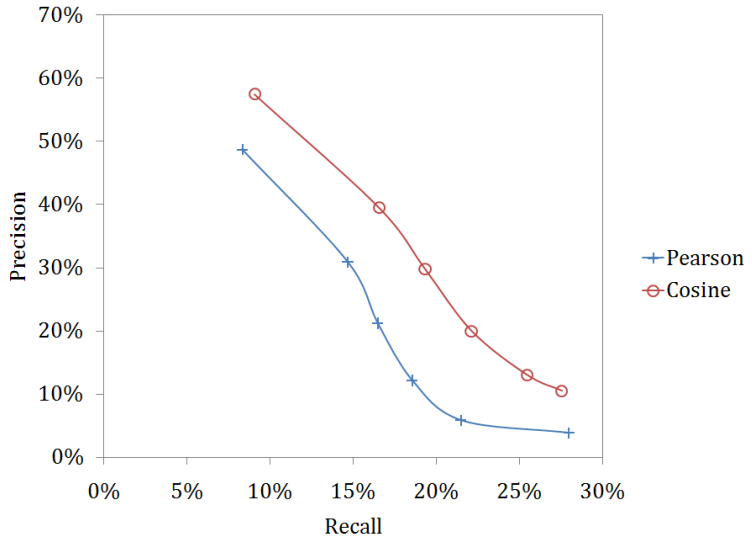
We begin by comparing Pearson correlation and Cosine similarity collaborative filtering algorithms over two tasks—the prediction task and the "recommend good items" task. Both algorithms used neighborhoods consisting of the closest 25 users.

First we evaluated the algorithms in predicting ratings on the Netflix and BookCrossing data sets, where we sampled 2000 test users and a randomly chosen number of test items per test user on each data set. Given Table 2, the algorithm of choice is clear—on both data sets, the predictions given by the Pearson correlation algorithm have lower RMSE scores, and the differences pass a sign test with $p < 0.0001$.

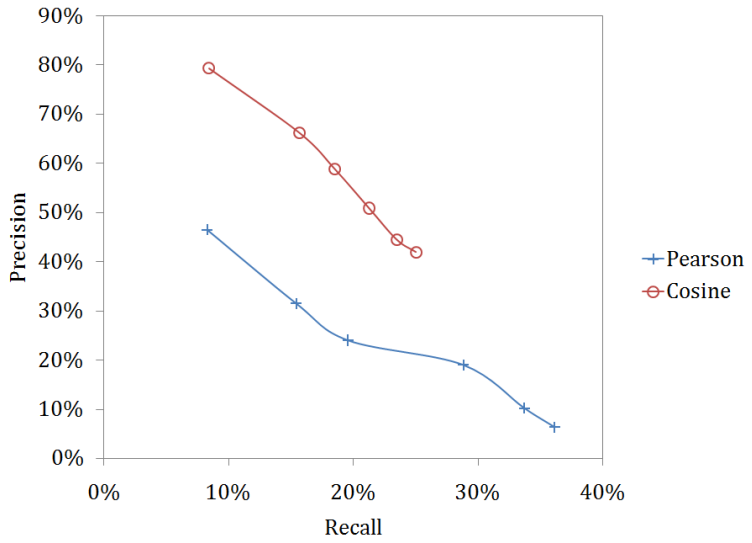
We then evaluated the two algorithms on the recommendation task on the Belgian retailer and news click stream data sets, where we again sampled 2000 test users and a randomly chosen number of test items per test user on each data set. Let us now evaluate the two algorithms on recommendation tasks. To do that, we computed precision-recall curves for the two algorithms on the Belgian retailer data set and the news click stream data set. This was done by computing precision and recall at 1, 3, 5, 10, 25, and 50 recommendations, and averaging the precisions and recalls at each number of recommendations. As Figure 2 shows, in both cases the recommendation lists generated by the Cosine similarity dominate the recommendation lists generated by the Pearson correlation algorithm in terms of precision. In the Belgian retailer data, Cosine similarity also has better recall than Pearson correlation across the board. On the news click stream data, Cosine similarity has better recall than Pearson correlation for 1, 3, and 5, recommendations. All these comparisons were significant according to a sign test with $p < 0.0001$. Therefore, in these cases, one would select the Cosine algorithm as the most appropriate choice.

This experiment shows that an algorithm that is uniformly better at predicting ratings on ratings data sets is not necessarily better at making recommendations on usage data sets. This suggests that it is possible that an algorithm that is better at predicting ratings could be worse at predicting usage in the same domain as well. An interesting experiment would be, given both ratings and usage data over the same users and items, to see whether algorithms that generate recommendation lists by decreasing order of predicted ratings do as well in the recommendation task over the usage data. Unfortunately, we are unaware of any public data set that contains both types of information. Nevertheless, companies such as Amazon or Netflix collect data both of user purchases and of user ratings over items. These companies can therefore make the appropriate decision for the recommendation task at hand.

It is also possible that websites that support multiple recommendation tasks should use different algorithms for the different tasks. For example, the Netflix website contains a prediction task (e.g., for new releases), and two recommendation tasks—a fixed list of recommendations when adding items to the rental queue, and an unlimited list of recommendations in the "movie you will like"



(a) Belgian retailer recommendations



(b) News click stream recommendations

Figure 2: Comparing recommendations generated by Pearson correlation and Cosine similarity. In both cases, the recommendation list is ordered by decreasing predicted score.

section. It may well be that different algorithms that were trained over different data sets (ratings vs. rentals) may rank differently in different tasks. Deciding on the best recommendation engine based solely on RMSE in the prediction task may lead to worse recommendation lists in the two other cases.

6.3.2 RECOMMENDATION VS. UTILITY MAXIMIZATION

In many retail applications, where the retailer is interested in maximizing profits, people may still train their algorithm on usage data solely, and evaluate them using recommendation oriented metrics, such as precision-recall. For example, even though a retailer website wishes to maximize its profit, it may use a binary data set of items that were bought by users to generate recommendations of the type "people who bought this item also bought ...".

To evaluate the performance of such an approach, we used an item-item recommendation system to generate recommendations for the Ta-Feng data set. Alternatively, one can order the items by expected utility. To compute an estimate of expected utility, we interpreted the normalized predicted score of each recommended item as the probability that the user would actually buy that item. In the case where the recommender predicts numerical ratings, we normalize by the highest possible rating and treat the result as a probability distribution. For example, if the highest rating is 5 and the algorithm predicts a score of 4.5 for a specific user we assume the the probability that the user will use the item is 0.9. We then use the average profit earned from each item to predict the profit that would be obtained from each item if the active user bought it. Multiplying the probability that the user would buy an item buy the profit that would result if the user bought the item yielded the required estimate of expected utility.

We then evaluated the two algorithms by comparing their precision-recall curves, which are shown in Figure 3. The curves were generated by evaluating precision and recall at 1, 3, 5, 10, 25, and 50 recommendations, and averaging the precisions and recalls at each number of recommendations. The averages were computed over 2000 users, and the item-item recommender outperformed the expected profit recommender in terms of both precision and recall at all points with $p < 0.0001$.

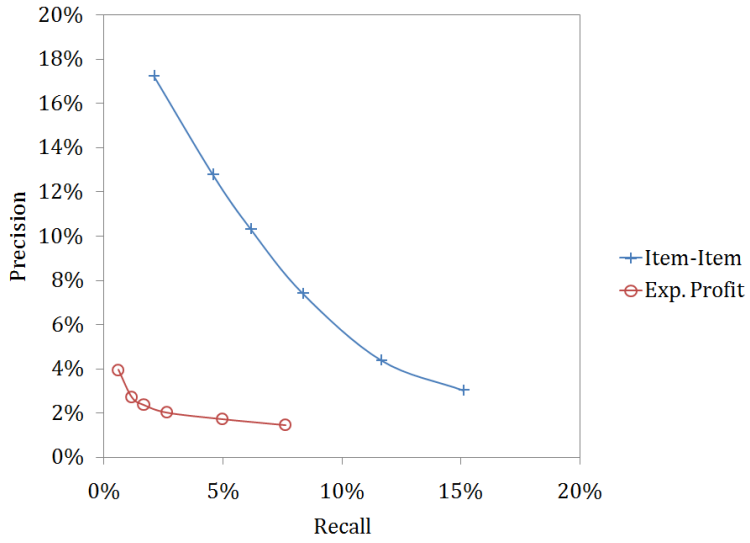


Figure 3: Comparing recommendations generated by the item-item recommender and the expected profit recommender on the Ta-Feng data set.

We then measured an half-life utility score where the utility of a correct recommendation was the profit from selling the correctly recommended item to the user. The results are shown in Table 3.

	Score
Item-Item	0.01
Exp. Profit	0.05

Table 3: Comparing item-item vs. expected utility recommendations on the Ta-Feng data set with the half-life utility score. The utility of a correct recommendation was the profit from selling that item to that user, while the half-life was 5. The trends were similar for other choices of the half-life parameter.

Choosing a recommender based on classification performance would have resulted in an half-life utility score (expected profit) that was 20% of what could have been achieved with the correct choice. This difference is statistically significant with $p < 0.001$.

These results are, of course, not surprising—using a recommender that explicitly attempts to optimize expected profit gives a better expected profit measure. However, occasionally people that are interested in maximizing profit, providing maximum value to the users, or minimizing user effort, use precision-recall to choose a recommendation algorithm.

7. Discussion

Above, we discussed the major considerations that one should make when deciding on the proper evaluation metric for a given task. We now add some discussion, illustrating other conclusions that can be derived, and illuminating some other relevant topics.

7.1 Evaluating Complete Recommender Systems

This survey focuses on the evaluation of recommendation algorithms. However, the success of a recommendation system does not depend solely on the quality of the recommendation algorithm. Such systems typically attempt to modify user behavior which is influenced by many other parameters, most notably, by the user interface. The success of the deployed system in influencing users can be measured through the change in user behavior, such as the number of recommendations that are followed, or the change in revenue.

Decisions about the interface by which users view recommendations are critical to the success of the system. For example, recommendations can be located in different places in the page, can be displayed horizontally or vertically, can be presented through images or text, and so forth. These decisions can make a significant impact, no smaller than the quality of the underlying algorithm, on the success of a system.

When the application is centered around the recommendation system, it is important to select the user interface together with the recommendation algorithm. In other cases, the recommendation system is only a supporting system for the application. For example, an e-commerce website is centered around the item purchases, and a news website is centered around the delivery of news stories. In both cases, a recommender system may be employed to help the users navigate, or to increase sales. It is likely that the recommendations are not the major method for browsing the collection of items.

When the recommender system is only a supporting system, the designer of the application will probably make the decision about the user interface without focusing on positioning recommendations where they have the most influence. In such cases, which we believe to be very common, the developer of the recommender system is constrained by the pre-designed interface, and in many cases can therefore only decide on the best recommendation algorithm, and in some cases perhaps the length of the recommendation list. This paper is targeted at researchers and developers who are making decisions about algorithms, not about the user interface. Designing a good user interface is an interesting and challenging problem, but it is outside the scope of this survey (see, e.g., Pu and Chen 2006).

7.2 Eliciting Utility Functions

A simple way to avoid the need to classify recommendation algorithms, is to assume that we are always optimizing some utility function. This utility function can be the user utility for items (see, e.g., Kumar et al. 1998, Price and Messinger 2005 and Hu and Pu 2009), or it can be the application utility. We can then ask users about their true utility function, or design a utility function that captures the goal of the application, and always choose the algorithm that maximizes this utility.

However, such a view is misleading; eliciting user utilities can be a very difficult task (see, e.g., Braziunas and Boutilier 2005, citealtChajewska and Huang 2008). For example, the value that the user is willing to invest in a laptop depends on a multitude of elements, such as her income, her technical knowledge, the intended use of the laptop and so forth. Indeed, eliciting such functions is the focus of active research, for example by presenting users with forced choices. Therefore, expecting that we will have access to a good estimate of the user utility function is unrealistic—estimating this function may be no easier than coming up with good recommendations.

Furthermore, even when the application designer understands the application utility function, gathering utilities from users may be very difficult. For example, in the Netflix domain, the business model may be to keep the users subscribed. Therefore, the utility of an movie for a user (from the website perspective) is not whether the user enjoys the movie, but rather whether the user will maintain her subscription if the movie is suggested to her. Clearly, most users will not want to answer such questions, and many may not know the answer themselves.

For this reason, when the utility function is unclear, the best we can do is to maximize the number of useful items that we suggest to the user. As such, the recommendation task cannot be viewed as a sub-task for utility optimization.

7.3 Implicit vs. Explicit ratings

Our classification of recommendation tasks sheds some light over another, commonly discussed subject in recommender systems, namely, implicit ratings (Claypool et al., 2001; Oard and Kim, 1998). In many applications, people refer to data that was automatically collected, such as logs of web browsing, or records of product purchases, as an implicit indication for positive opinions over the items that were visited or purchased.

However, this perspective is appropriate only if the task is the prediction task, where we would like to know whether the user will have a positive or negative opinion over certain items. If the task is to recommend more items that the user may buy, given the items that the user has already bought, purchase data becomes an explicit indication. In this case, using ratings that users provide over items is an implicit indication to whether the user will buy the item.

For example, a user may have a positive opinion over many laptop computers, and may rate many laptops highly. However, most people buy only one laptop. In that case, recommending more laptops, based on the co-occurring high ratings, will be inappropriate. However, if we predict the probability of buying a laptop given that another laptop has already been bought, we can expect this probability to be low, and the other laptop will not be recommended.

8. Related Work

In the past, different researchers discussed various topics relevant to the evaluation of recommender systems.

Breese et al. (1998) were probably the first to provide a sound evaluation of a number of recommendation approaches over a collection of different data sets, setting the general framework of evaluating algorithms on more than a single real world data set, and a comparison of several algorithms in identical experiments. The practices that were illustrated in that paper are used in many modern publications.

Herlocker et al. (2004) provide an extensive survey of possible metrics for evaluation. They then compare a set of metrics, concluding that for some pairs of metrics, using both together will give very little additional information compared to using just one.

Another interesting contribution of that paper is a classification of recommendation engines from the user task perspective, namely, what are the reasons and motivations that a user has when interacting with a recommender system. As they are interested in user tasks, and we are interested in the system tasks, our classification is different, yet we share some similar tasks, such as the "recommend some good items" and "recommend all good items" tasks.

Finally, their survey attempted to cover as many evaluation metrics and user task variations as possible, we focus here on the appropriate metrics for the most popular recommendation tasks only.

Mcnee et al. (2003) explain why accuracy metrics alone are insufficient for selecting the correct recommendation algorithm. For example, users may be interested in the serendipity of the recommended items. One way to model serendipity is through a utility function that assigns higher values to "unexpected" suggestions. They also discuss the "usefulness" of recommendations. Many utility functions, such as the inverse log of popularity (Shani et al., 2005) attempt to capture this "usefulness".

Ziegler et al. (2005) focus on another aspect of evaluation—considering the entire list together. This would allow us to consider aspects of a set of recommendations, such as diversification between items in the same list. Our suggested metrics consider only single items, and thus could not be used to evaluate entire lists. It would be interesting to see more evaluation metrics that provide observations over complete lists.

Celma and Herrera (2008) suggest looking at topological properties of the recommendation graph—the graph that connects recommended items. They explain how by looking at the recommendation graph one may understand properties such as the novelty of recommendations. It is still unclear how these properties correlate with the true goal of the recommender system, may it be to optimize revenue or to recommend useful items.

McLaughlin and Herlocker (2004) argue, as we do, that MAE is not appropriate for evaluating recommendation tasks, and that ratings are not necessarily indicative of whether a user is likely to watch a movie. The last claim can be explained by the way we view implicit and explicit ratings.

Some researchers have suggested taking a more holistic approach, and considering the recommendation algorithm within the complete recommendation system. For example, del Olmo and Gaudioso (2008), suggest that systems be evaluated only after deployment, through counting the number of successful recommendations. As we argue above, even in these cases, one is likely to evaluate algorithms offline, to avoid presenting recommendations that have poor quality for users, thus losing their trust.

9. Conclusion

In this paper we discussed how recommendation algorithms should be evaluated in order to select the best algorithm for a specific task from a set of candidates. This is an important step in the research attempt to find better algorithms, as well as in application design where a designer chooses an existing algorithm for their application. As such, many evaluation metrics have been used for algorithm selection in the past.

We review three core tasks of recommendation systems—the prediction task, the recommendation task, and the utility maximization task. Most evaluation metrics are naturally appropriate for one task, but not for the others. We discuss for each task a set of metrics that are most appropriate for selecting the best of the candidate algorithms.

We empirically demonstrate that in some cases two algorithms can be ranked differently by two metrics over the same data set, emphasizing the importance of choosing the appropriate metric for the task, so as not to choose an inferior algorithm.

We also describe the concerns that need to be addressed when designing offline and online experiments. We outline a few important measurements that one must take in addition to the score that the metric provides, as well as other considerations that should be taken into account when designing experiments for recommendation algorithms.

References

D. Bamber. The area above the ordinal dominance graph and the area below the receiver operating characteristic graph. *Journal of Mathematical Psychology*, 12:387–415, 1975.

Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research*, 5, 2004.

F. Bodon. A fast APRIORI implementation. In *The IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2003.

D. Braziunas and C. Boutilier. Local utility elicitation in GAI models. In *Proceedings of the Twenty-first Conference on Uncertainty in Artificial Intelligence*, pages 42–49, Edinburgh, 2005.

J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI: Uncertainty in Artificial Intelligence*, pages 43–52, 1998.

Ò. Celma and P. Herrera. A new approach to evaluating novel recommendations. In *RecSys '08: Proceedings of the 2008 ACM Conference on Recommender Systems*, 2008.

M. Claypool, P. Le, M. Waseda, and D. Brown. Implicit interest indicators. In *Intelligent User Interfaces*, pages 33–40. ACM Press, 2001.

F. Hernández del Olmo and E. Gaudioso. Evaluation of recommender systems: A new approach. *Expert Systems Applications*, 35(3), 2008.

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7, 2006.

R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

C. Goutte and E. Gaussier. A probabilistic interpretation of precision, recall, and F-score, with implication for evaluation. In *ECIR '05: Proceedings of the 27th European Conference on Information Retrieval*, pages 345–359, 2005.

A. Gunawardana and C. Meek. Aggregators and contextual effects in search ad markets. In *WWW Workshop on Targeting and Ranking for Online Advertising*, 2008.

J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1), 2004.

C. N. Hsu, H. H. Chung, and H. S. Huang. Mining skewed and sparse transaction data for personalized shopping recommendation. *Machine Learning*, 57(1-2), 2004.

R. Hu and P. Pu. A comparative user study on rating vs. personality quiz based preference elicitation methods. In *IUI '09: Proceedings of the 13th International Conference on Intelligent User Interfaces*, 2009.

S. L. Huang. Comparision of utility-based recommendation methods. In *The Pacific Asia Conference on Information Systems*, 2008.

C. Kadie, C. Meek, and D. Heckerman. CFW: A collaborative filtering system using posteriors over weights of evidence. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 242–250, San Francisco, CA, 2002. Morgan Kaufmann.

R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne. Controlled experiments on the web: survey and practical guide. *Data Mining and Knowledge Discovery*, 18(1), 2009.

J. A. Konstan, S. M. McNee, C. N. Ziegler, R. Torres, N. Kapoor, and J. Riedl. Lessons on applying automated recommender systems to information-seeking tasks. In *Proceedings of the Twenty-First National Conference on Artifical Intelligence (AAAI)*, 2006.

R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Recommendation systems: A probabilistic analysis. In *FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, 1998.

D. Larocque, J. Nevalainen, and H. Oja. A weighted multivariate sign test for cluster-correlated data. *Biometrika*, 94:267–283, 2007.

G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1), 2003.

M. R. McLaughlin and J. L. Herlocker. A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In *SIGIR '04: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2004.

S. Mcnee, S. K. Lam, C. Guetzlaff, J. A. Konstan, and J. Riedl. Confidence displays and training in recommender systems. In *Proceedings of the 9th IFIP TC13 International Conference on Human Computer Interaction INTERACT*, pages 176–183. IOS Press, 2003.

S. M. McNee, J. Riedl, and J. K. Konstan. Making recommendations better: an analytic model for human-recommender interaction. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, 2006.

M. Montaner, B. López, and J. L. De La Rosa. A taxonomy of recommender agents on the internet. *Artificial Intelligence Review*, 19(4), 2003.

D. Oard and J. Kim. Implicit feedback for recommender systems. In *The AAAI Workshop on Recommender Systems*, pages 81–83, 1998.

B. Price and P. Messinger. Optimal recommendation sets: Covering uncertainty over user preferences. In *National Conference on Artificial Intelligence (AAAI)*, pages 541–548. AAAI Press AAAI Press / The MIT Press, 2005.

P. Pu and L. Chen. Trust building with explanation interfaces. In *IUI '06: Proceedings of the 11th International Conference on Intelligent User Interfaces*, 2006.

P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3), 1997.

C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 1979.

G. Salton. *The SMART Retrieval System—Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.

S. L. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3), 1997.

B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, 2000.

J. B. Schafer, J. Konstan, and J. Riedi. Recommender systems in e-commerce. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, 1999.

A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *SIGIR '02: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2002.

G. Shani, D. Heckerman, and R. I. Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6:1265–1295, 2005.

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society B*, 36(1):111–147, 1974.

E. M. Voorhees. The philosophy of information retrieval evaluation. In *CLEF '01: Revised Papers from the Second Workshop of the Cross-Language Evaluation Forum on Evaluation of Cross-Language Information Retrieval Systems*, 2002a.

E. M. Voorhees. Overview of trec 2002. In *The 11th Text Retrieval Conference (TREC 2002), NIST Special Publication 500-251*, pages 1–15, 2002b.

C. N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW '05: Proceedings of the 14th International Conference on the World Wide Web*, 2005.