Chapter 8
Evaluating Recommendation Systems

Guy Shani and Asela Gunawardana

Abstract *Recommender systems* are now popular both commercially and in the research community, where many approaches have been suggested for providing recommendations. In many cases a system designer that wishes to employ a recommendation system must choose between a set of candidate approaches. A first step towards selecting an appropriate algorithm is to decide which properties of the application to focus upon when making this choice. Indeed, recommendation systems have a variety of properties that may affect user experience, such as accuracy, robustness, scalability, and so forth. In this paper we discuss how to compare recommenders based on a set of properties that are relevant for the application. We focus on comparative studies, where a few algorithms are compared using some evaluation metric, rather than absolute benchmarking of algorithms. We describe experimental settings appropriate for making choices between algorithms. We review three types of experiments, starting with an offline setting, where recommendation approaches are compared without user interaction, then reviewing user studies, where a small group of subjects experiment with the system and report on the experience, and finally describe large scale online experiments, where real user populations interact with the system. In each of these cases we describe types of questions that can be answered, and suggest protocols for experimentation. We also discuss how to draw trustworthy conclusions from the conducted experiments. We then review a large set of properties, and explain how to evaluate systems given relevant properties. We also survey a large set of evaluation metrics in the context of the properties that they evaluate.

Guy Shani
Microsoft Research, One Microsoft Way, Redmond, WA, e-mail: guyshani@microsoft.com

Asela Gunawardana
Microsoft Research, One Microsoft Way, Redmond, WA, e-mail: aselag@microsoft.com

F. Ricci et al. (eds.), *Recommender Systems Handbook*,
DOI 10.1007/978-0-387-85820-3_8,

8.1 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 predict how much they 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 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 goals. Typically, such decisions are based on experiments, comparing the performance of a number of candidate recommenders. The designer can then select the best performing algorithm, given structural constraints such as the type, timeliness and reliability of availability data, allowable memory and CPU footprints. 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).

Initially most recommenders have been evaluated and ranked on their prediction power — their ability to accurately predict the user's choices. However, it is now widely agreed that accurate predictions are crucial but insufficient to deploy a good recommendation engine. In many applications people use a recommendation system for more than an exact anticipation of their tastes. Users may also be interested in discovering new items, in rapidly exploring diverse items, in preserving their privacy, in the fast responses of the system, and many more properties of the interaction with the recommendation engine. We must hence identify the set of properties that may influence the success of a recommender system in the context of a specific application. Then, we can evaluate how the system preforms on these relevant properties.

In this paper we review the process of evaluating a recommendation system. We discuss three different types of experiments; offline, user studies and online experiments.

[1] www.Netflix.com

[2] www.amazon.com

Often it is easiest to perform offline experiments using existing data sets and a protocol that models user behavior to estimate recommender performance measures such as prediction accuracy. A more expensive option is a user study, where a small set of users is asked to perform a set of tasks using the system, typically answering questions afterwards about their experience. Finally, we can run large scale experiments on a deployed system, which we call online experiments. Such experiments evaluate the performance of the recommenders on real users who are oblivious to the conducted experiment. We discuss what can and cannot be evaluated for each of these types of experiments.

We can sometimes evaluate how well the recommender achieves its overall goals. For example, we can check an e-commerce website revenue with and without the recommender system and thereby estimate the value of the system to the website. In other cases, it can also be useful to evaluate how recommenders perform in terms of some specific properties, allowing us to focus on improving properties that fall short. First, one must show that a property is indeed relevant to users and affect their experience. Then, we can design algorithms that improve upon these properties. In improving one property we may reduce the quality of another property, creating a a trade-off between a set of properties. In many cases it is also difficult to say how these trade-offs affect the overall performance of the system, and we have to either run additional experiments to understand this aspect, or use the opinions of domain experts.

This paper focuses on property-directed evaluation of recommender algorithms. We provide an overview of a large set of properties that can be relevant for system success, explaining how candidate recommenders can be ranked with respect to these properties. For each property we discuss the relevant experiment types—offline, user study, and online experiments—and explain how an evaluation can be conducted in each case. We explain the difficulties and outline the pitfalls in evaluating each property. For all these properties we focus on ranking recommenders on that property, assuming that better handling the property will improve user experience.

We also review a set of previous suggestions for evaluating recommendation systems, describing a large set of popular methods and placing them in the context of the properties that they measure. We especially focus on the widely researched accuracy and ranking measurements, describing a large set of evaluation metrics for these properties. For other, less studied properties, we suggest guidelines from which specific measures can be derived. We provide examples of such specific implementations where appropriate.

The rest of the paper is structured as follows. In Section 8.2 we discuss the different experimental settings in which recommender systems can be evaluated, discussing the appropriate use of offline experiments, user studies, and online trials. We also outline considerations that go into making reliable decisions based on these experiments, including generalization and statistical significance of results. In Section 8.3 we describe a large variety of properties of recommendation systems that may impact their performance, as well as metrics for measuring these properties. Finally, we conclude in Section 8.4.

8.2 Experimental Settings

In this section we describe three levels of experiments that can be used in order to compare several recommenders. The discussion below is motivated by 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 [49, 13, 60].

We begin with offline experiments, which are typically the easiest to conduct, as they require no interaction with real users. We then describe user studies, where we ask a small group of subjects to use the system in a controlled environment, and then report on their experience. In such experiments we can collect both quantitative and qualitative information about the systems, but care must be taken to consider various biases in the experimental design. Finally, perhaps the most trustworthy experiment is when the system is used by a pool of real users, typically unaware of the experiment. While in such an experiment we are able to collect only certain types of data, this experimental design is closest to reality.

In all experimental scenarios, it is important to follow a few basic guidelines in general experimental studies:

- **Hypothesis:** before running the experiment we must form an hypothesis. It is important to be concise and restrictive about this hypothesis, and design an experiment that tests the hypothesis. For example, an hypothesis can be that algorithm A better predicts user ratings than algorithm B. In that case, the experiment should test the prediction accuracy, and not other factors.
- **Controlling variables:** when comparing a few candidate algorithms on a certain hypothesis, it is important that all variables that are not tested will stay fixed. For example, suppose that we wish to compare the prediction accuracy of movie ratings of algorithm A and algorithm B, that both use different collaborative filtering models. If we train A on the MovieLens data set, and B on the Netflix data set, and algorithm A presents superior performance, we can not tell whether the performance was due to the superior CF model, or due to the better input data, or both. We therefore must train the algorithms on the same data set (or over unbiased samples from the same data set), or train the same algorithms over the two different data sets, in order to understand the cause of the superior performance.
- **Generalization power:** when drawing conclusions from experiments, we may desire that our conclusions generalize beyond the immediate context of the experiments. When choosing an algorithm for a real application, we may want our conclusions to hold on the deployed system, and generalize beyond our experimental data set. Similarly, when developing new algorithms, we want our conclusions to hold beyond the scope of the specific application or data set that we experimented with. To increase the probability of generalization of the results we must typically experiment with several data sets or applications. It is important to understand the properties of the various data sets that are used.

Generally speaking, the more diverse the data used, the more we can generalize the results.

8.2.1 Offline Experiments

An offline experiment is performed by using a pre-collected data set of users choosing or rating items. Using this data set we can try to simulate the behavior of users that interact with a recommendation system. In doing so, we assume that the user behavior when the data was collected will be similar enough to the user behavior when the recommender system is deployed, so that we can make reliable decisions based on the simulation. Offline experiments are attractive because they require no interaction with real users, and thus allow us to compare a wide range of candidate algorithms at a low cost. The downside of offline experiments is that they can answer a very narrow set of questions, typically questions about the prediction power of an algorithm. In particular, we must assume that users' behavior when interacting with a system including the recommender system chosen will be modeled well by the users' behavior prior to that system's deployment. Thus we cannot directly measure the recommender's influence on user behavior in this setting.

Therefore, the goal of the offline experiments is to filter out inappropriate approaches, leaving a relatively small set of candidate algorithms to be tested by the more costly user studies or online experiments. A typical example of this process is when the parameters of the algorithms are tuned in an offline experiment, and then the algorithm with the best tuned parameters continues to the next phase.

8.2.1.1 Data sets for offline experiments

As the goal of the offline evaluation is to filter algorithms, 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). Sometimes, known biases in the data can be corrected for by techniques such as reweighing data, but correcting biases in the data is often difficult.

Another source of bias may be the data collection itself. For example, users may be more likely to rate items that they have strong opinions on, and some users may

provide many more ratings than others. Thus, the set of items on which explicit ratings are available may be biased by the ratings themselves [38]. Once again, techniques such as resampling or reweighting the test data may be used to attempt to correct such biases.

8.2.1.2 Simulating user behavior

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. In many cases, though, we are restricted by the computational cost of an evaluation protocol, and must make compromises in order to execute the experiment over large data sets.

Ideally, if we have access to time-stamps for user selections, we can simulate what the systems predictions would have been, had it been running at the time the data set was collected. We can begin with no available prior data for computing predictions, and step through user selections in temporal order, attempting to predict each selection and then making that selection available for use in future predictions. For large data sets, a simpler approach is to 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 still be computationally quite expensive.

An even 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 built 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 the sequence in which items are selected is important, not the absolute times when the selections are made. A final alternative is to ignore time. We would first sample a set of test users, then sample the number n_a of items to hide for each user a, and finally sample n_a items to hide. This assumes that the temporal aspects of user selections are unimportant. We may be forced to make this assumption if the timestamps of user actions are not known. 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 (see Chapter 11), given the constraints, 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 only 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 protocols have biases that make them unreliable in predicting the performance of the algorithms online.

8.2.1.3 More complex user modeling

All the protocols that we discuss above make some assumptions concerning the behavior of users, which could be regarded as a user-model for the specific application. While we discuss only very simple user-models it is possible to suggest more complicated models for user behavior [37]. Using advanced user models we can execute simulations of users interactions with the system, thus reducing the need for expensive user studies and online testing. However, care must be made when designing user-models; First, user-modeling is a difficult task, and there is a vast amount of research on the subject (see, e.g. [15]). Second, when the user model is inaccurate, we may optimize a system whose performance in simulation has no correlation with its performance in practice. While it is reasonable to design an algorithm that uses complex user-models to provide recommendations, we should be careful in trusting experiments where algorithms are verified using such complex, difficult to verify user models.

8.2.2 User Studies

Many recommendation approaches rely on the interaction of users with the system (see, e.g., Chapters 23, 13, 6). It is very difficult to create a reliable simulation of users interactions with the system, and thus, offline testing are difficult to conduct. In order to properly evaluate such systems, real user interactions with the system must be collected. Even when offline testing is possible, interactions with real users can still provide additional information about the system performance. In these cases we typically conduct user studies.

A user study is conducted by recruiting a set of test subjects, and asking them to perform several tasks requiring an interaction with the recommendation system. While the subjects perform the tasks, we observe and record their behavior, collecting any number of quantitative measurements, such as what portion of the task was completed, the accuracy of the task results, or the time taken to perform the task. In many cases we can ask qualitative questions before, during, and after the task is completed. Such questions can collect data that is not directly observable, such

as whether the subject enjoyed the user interface, or whether the user perceived the task as easy to complete.

A typical example of such an experiment is to test the influence of a recommendation algorithm on the browsing behavior of news stories. In this example, the subjects are asked to read a set of stories that are interesting to them, in some cases including related story recommendations and in some cases without recommendations. We can then check whether the recommendations are used, and whether people read different stories with and without recommendations. We can collect data such as how many times a recommendation was clicked, and even, in certain cases, track eye movement to see whether a subject looked at a recommendation. Finally, we can ask qualitative questions such as whether the subject thought the recommendations were relevant.

Of course, in many other research areas user studies are a central tool, and thus there is much literature on the proper design of user studies. This section only overviews the basic considerations that should be taken when evaluating a recommender system through a user study, and the interested reader can find much deeper discussions elsewhere (see. e.g. [5]).

8.2.2.1 Advantages and Disadvantages

User studies can perhaps answer the widest set of questions of all three experimental settings that we survey here. Unlike offline experiments this setting allows us to test the behavior of users when interacting with the recommendation system, and the influence of the recommendations on user behavior. In the offline case we typically make assumptions such as "given a relevant recommendation the user is likely to use it" which are tested in the user study. Second, this is the only setting that allows us to collect qualitative data that is often crucial for interpreting the quantitative results. Also, we can typically collect in this setting a large set of quantitative measurements because the users can be closely monitored while performing the tasks.

User studies however have some disadvantages. Primarily, user studies are very expensive to conduct; collecting a large set of subjects and asking them to perform a large enough set of tasks is costly in terms of either user time, if the subjects are volunteers, or in terms of compensation if paid subjects are employed. Therefore, we must typically restrict ourselves to a small set of subjects and a relatively small set of tasks, and cannot test all possible scenarios. Furthermore, each scenario has to be repeated several time in order to make reliable conclusions, further limiting the range of distinct tasks that can be tested.

As these experiments are expensive to conduct we should collect as much data about the user interactions, in the lowest possible granularity. This will allow us later to study the results of the experiment in detail, analyzing considerations that were not obvious prior to the trial. This guideline can help us to reduce the need for successive trials to collect overlooked measurements.

Furthermore, in order to avoid failed experiments, such as applications that malfunction under certain user actions, researchers often execute pilot user studies.

These are small scale experiments, designed not to collect statistical data, but to test the systems for bugs and malfunctions. In some cases, the results of these pilot studies are then used to improve the recommender. If this is the case, then the results of the pilot become "tainted", and should not be used when computing measurements in the final user study.

Another important consideration is that the test subjects must represent as closely as possible the population of users of the real system. For example, if the system is designed to recommend movies, the results of a user study over avid movie fans may not carry to the entire population. This problem is most persistent when the participants of the study are volunteers, as in this case people who are originally more interested in the application may tend to volunteer more readily.

However, even when the subjects represent properly the true population of users, the results can still be biased because they are aware that they are participating in an experiment. For example, it is well known that paid subjects tend to try and satisfy the person or company conducting the experiment. If the subjects are aware of the hypothesis that is tested they may unconsciously provide evidence that supports it. To accommodate that, it is typically better not to disclose the goal of the experiment prior to collecting data. Another, more subtle effect occurs when the payment to subjects takes the form of a complete or partial subsidy of items they select. This may bias the data in cases where final users of the system are not similarly subsidized, as users' choices and preferences may be different when they pay full price.

8.2.2.2 Between vs. Within Subjects

As typically a user study compares a few candidate approaches, each candidate must be tested over the same tasks. To test all candidates we can either compare the candidates *between subjects*, where each subject is assigned to a candidate method and experiments with it, or *within subjects*, where each subject tests a set of candidates on different tasks [20].

Typically, within subjects experiments are more informative, as the superiority of one method cannot be explained by a biased split of users between candidate methods. It is also possible in this setting to ask comparative questions about the different candidates, such as which candidate the subject preferred. However, in these types of tests users are more conscious of the experiment, and hiding the distinctions between candidates is more difficult.

Between subjects experiments, also known as *A-B testing* (All Between), provide a setting that is closer to the real system, as each user experiments with a single system. Such experiments can also test long term effects of using the system, because the user is not required to switch systems. Thus we can test how the user becomes accustomed to the system, and estimate a learning curve of expertise.

8.2.2.3 Variable Counter Balance

As we have noted above, it is important to control all variables that are not specifically tested. However, when a subject is presented with the output of several candidates, as is done in within subject experiments, we must counter balance several variables.

When presenting several results to the subject, the results can be displayed either sequentially, or together. In both cases there are certain biases that we need to correct for. When presenting the results sequentially the previously observed results influence the user opinion of the current results. For example, if the results that were displayed first seem inappropriate, the results displayed afterwards may seem better than they actually are. When presenting two sets of results, there can be certain biases due to location. For example, users from many cultures tend to observe results left to right and top to bottom. Thus, the user may observe the results displayed on top as superior.

A common approach to correct for such untested variables is by using the *Latin square* [5] procedure. This procedure randomizes the order or location of the various results each time, thus canceling out biases due to these untested variables.

8.2.2.4 Questionnaires

User studies allow us to use the powerful questionnaire tool. Prior, during, and after subjects perform their tasks we can ask them questions about their experience. These questions can provide information about properties that are difficult to measure, such as the subject's state of mind, or whether the subject enjoyed the system.

While these questions can provide valuable information, they can also provide misleading information. It is important to ask neutral questions, that do not suggest a "correct" answer. People may also answer untruthfully, for example when they perceive the answer as private, or if they think the true answer may put them in an unflattering position.

Indeed, vast amount of research was conducted in other areas about the art of questionnaire writing, and we refer the readers to that literature (e.g. [46]) for more details.

8.2.3 Online Evaluation

In many realistic recommendation applications 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, or if some utility gathered from users of one system exceeds utility gathered from users of the

other system, then we can conclude that one system is superior to the other, all else being equal.

The real effect of the recommendation system depends 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 recommendations are presented.

Thus, the experiment that provides the strongest evidence as to the true value of the system is an online evaluation, where the system is used by real users that perform real tasks. It is most trustworthy to compare a few systems online, obtaining a ranking of alternatives, rather than absolute numbers that are more difficult to interpret.

For this reason, many real world systems employ an online testing system [32], where multiple algorithms can be compared. Typically, such systems redirect a small percentage of the traffic to different alternative 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 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.

In some cases, such experiments are risky. For example, a test system that provides irrelevant recommendations, may discourage the test users from using the real system ever again. Thus, the experiment can have a negative effect on the system, which may be unacceptable in commercial applications.

For these reasons, it is best to run an online evaluation last, after an extensive offline study provides evidence that the candidate approaches are reasonable, and perhaps after a user study that measures the user's attitude towards the system. This gradual process reduces the risk in causing significant user dissatisfaction.

Online evaluations are unique in that they allow direct measurement of overall system goals, such as long-term profit or user retention. As such, they can be used to understand how these overall goals are affected by system properties such as recommendation accuracy and diversity of recommendations, and to understand the trade-offs between these properties. However, since varying such properties independently is difficult, and comparing many algorithms through online trials is expensive, it can be difficult to gain a complete understanding of these relationships.

8.2.4 Drawing Reliable Conclusions

In any type of experiment it is important that we can be confidant that the candidate recommender 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 in an offline experiments, and the subjects in a user study, to best resemble the online application. Still, there is a possibility that the algorithm that performed best on this test set did so because the experiment was fortuitously suitable for that algorithm. To reduce the possibility of such statistical mishaps, we must perform significance testing on the results.

8.2.4.1 Confidence and p-values

A standard tool for significance testing is 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 some threshold. That is, if the probability that the observed ranking is achieved by chance exceeds the threshold, then the results of the experiment are not deemed significant. Traditionally, people choose $p = 0.05$ as their threshold, which indicates less than 95% confidence. 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.

8.2.4.2 Paired Results

In order to perform a significance test that algorithm A is indeed better than algorithm B, we often use the results of several independent experiments comparing A and B. Indeed, the protocol we have suggested for generating our test data (Section 8.2.1.2) allows us to obtain such a 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 basis.

Given such paired per-user performance measures for algorithms A and B a simple test of significance is the **sign test** [13]. 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 significance level is the probability that A is not truly better than B, and is estimated as the probability of at least n_A out of $n = n_A + n_B$ 0.5-probability Binomial trials succeeding (i.e. n_A out of $n_A + n_B$ fair coin-flips coming up "heads"), and is given by

$$p = (0.5)^n \sum_{i=n_A}^{n} \frac{n!}{i!(n-i)!} \tag{8.1}$$

The sign test is an attractive choice due to its simplicity, and lack of assumptions over the distribution of cases. While tied results are traditionally ignored, Demšar [13] recommends splitting them between A and B, since they provide evidence for the null hypothesis that the algorithms are no different. When $n_A + n_B$ is large, we

can take advantage of large sample theory to approximate (8.1) by a normal distribution. However, this is usually unnecessary with powerful modern computers. Some authors (e.g. [49]) use the term **McNemar's test** to refer to the use of a χ^2 approximation to the two-sided sign test.

Note that sometimes, the sign test may indicate that system A outperforms system B with high probability, even though the average performance of system B is higher than that of system A. This happens in cases where system B occasionally outperforms system A overwhelmingly. Thus, the reason for this seemingly inconsistent result is that the test only examines the probability of one system outperforming the other, without regard to the magnitude of the difference.

The sign test can be extended to cases where we want to know the probability that one system outperforms the other by some amount. For example, suppose that system A is much more resource intensive than system B, and is only worth deploying if it outperforms system B by some amount. We can define "success" in the sign test as A outperforming B by this amount, and find the probability of A not truly outperforming B by this amount as our p value in equation (8.1).

A commonly used test that takes the magnitude of the differences into account is the **paired Student's t-test**, which looks at the average difference between the performance scores of algorithms A and B, normalized by the standard deviation of the score difference. Using this test requires that the differences in scores for different users is comparable, so that averaging these differences is reasonable. For small numbers of users, the validity of the test also depends on the differences being Normally distributed. Demšar [13] points out that this assumption is hard to verify when the number of samples is small and that the t-test is susceptible to outliers. He recommends the use of **Wilcoxon signed rank test**, which like the t-test, uses the magnitude of the differences between algorithms A and B, but without making distributional assumptions on the differences. However, using the Wilcoxon signed rank test still requires that differences between the two systems are comparable between users.

Another way to improve the significance of our conclusions is to use a larger test set. In the offline case, this may require using a smaller training set, which may result in an experimental protocol that is not representative of the amount of training data available after deployment. In the case of user studies, this implies an additional expense. In the case of online testing, increasing the amount of data collected for each algorithm requires either the added expense of a longer trial or the comparison of fewer algorithms.

8.2.4.3 Unpaired Results

The tests described above are suitable for cases where observations are paired. That is, each algorithm is run on each test case, as is often done in offline tests. In online tests, however, it is often the case that users are assigned to one algorithm or the other, so that the two algorithms are not evaluated on the same test cases. The **Mann-**

Whitney test is an extension of the Wilcoxon test to this scenario. Suppose we have n_A results from algorithm A and n_B results from algorithm B.

The performance measures of the two algorithms are pooled and sorted so that the best result is ranked first and the worst last. The ranks of ties are averaged. For example if the second through fifth place tie, they are all assigned a rank of 3.5. The Mann-Whitney test computes the probability of the null hypothesis that n_A randomly chosen results from the total $n_A + n_B$ have at least as good an average rank as the n_A results that came from algorithm A.

This probability can be computed exactly be enumerating all $\frac{(n_A+n_B)!}{n_A!n_B!}$ choices and counting the choices that have at least the required average rank, or can be approximated by repeatedly resampling n_A of the results. When n_A and n_B are both large enough (typically over 5), the distribution of the average rank of n_A results randomly selected from a pool of $n_A + n_B$ under the null hypothesis is well approximated by a Gaussian with mean $\frac{1}{2}(n_A + n_B + 1)$ and standard deviation $\sqrt{\frac{1}{12}\frac{n_A}{n_B}(n_A + n_B + 1)}$. Thus, in this case we can compute the average rank of the n_A results from system A, subtract $\frac{1}{2}(n_A + n_B + 1)$, divide by $\sqrt{\frac{1}{12}\frac{n_A}{n_B}(n_A + n_B + 1)}$, and evaluate the standard Gaussian CDF at this value to get the p value for the test.

8.2.4.4 Multiple tests

Another important consideration, mostly in the offline scenario, 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 10 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 $1 - (1 - 0.05)^{10} = 0.40$. 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. To achieve a confidence $1 - p$ that the chosen algorithm is indeed the best, it must outperform 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, approaches such as **ANOVA** or the **Friedman test for ranking**, which are generalization of the Student's t-test and Wilcoxon's rank test. ANOVA makes strong assumptions about the Normality of the different algorithms' performance measures, and about the re-

lationships of their variances. We refer the reader to [13] for further discussion of these and other tests for ranking multiple algorithms.

A more subtle version of this concern is when a pair of algorithms are compared in a number of ways. For example, two algorithms may be compared using a number of accuracy measures, a number of coverage measures, etc. Even if the two algorithms are identical in all measures, the probability of finding a measure by which one algorithm seems to outperform the other with some significance level increases with the number of measures examined. If the different measures are independent, the Bonferroni correction mentioned above can be used. However, since the measures are often correlated, the Bonferroni correction may be too stringent, and other approaches such as controlling for false discovery rate [2] may be used.

8.2.4.5 Confidence Intervals

Even though we focus here on comparative studies, where one has to choose the most appropriate algorithm out of a set of candidates, it is sometimes desirable to measure the value of some property. For example, an administrator may want to estimate the error in the system predictions, or the net profit that the system is earning. When measuring such quantities it is important to understand the reliability of your estimates. A popular approach for doing this is to compute **confidence intervals**.

For example, one may estimate that the RMSE of a system is expected to be 1.2, and that it will be between 1.1 and 1.35 with probability 0.95. The simplest method for computing confidence intervals is to assume that the quantity of interest is Gaussian distributed, and then estimate its mean and standard deviations from multiple independent observations. When we have many observations, we can dispense with this assumption by computing the distribution of the quantity of interest with a non-parametric method such as a histogram and finding upper and lower bounds such that include the quantity of interest with the desired probability.

8.3 Recommendation System Properties

In this section we survey a range of properties that are commonly considered when deciding which recommendation approach to select. As different applications have different needs, the designer of the system must decide on the important properties to measure for the concrete application at hand. Some of the properties can be traded-off, the most obvious example perhaps is the decline in accuracy when other properties (e.g. diversity) are improved. It is important to understand and evaluate these trade-offs and their effect on the overall performance. However, the proper way of gaining such understanding without intensive online testing or defering to the opinions of domain experts is still an open question.

Furthermore, the effect of many of these properties on the user experience is unclear, and depends on the application. While we can certainly speculate that users

would like diverse recommendations or reported confidence bounds, it is essential to show that this is indeed important in practice. Therefore, when suggesting a method that improves one of this properties, one should also evaluate how changes in this property affects the user experience, either through a user study or through online experimentation.

Such an experiment typically uses a single recommendation method with a tunable parameter that affects the property being considered. For example, we can envision a parameter that controls the diversity of the list of recommendations. Then, subjects should be presented with recommendations based on a variety of values for this parameter, and we should measure the effect of the parameter on the user experience. We should measure here not whether the user noticed the change in the property, but whether the change in property has affected their interaction with the system. As is always the case in user studies, it is preferable that the subjects in a user study and users in an online experiment will not know the goal of the experiment. It is difficult to envision how this procedure could be performed in an offline setting because we need to understand the user response to this parameter.

Once the effects of the specific system properties in affecting the user experience of the application at hand is understood, we can use differences in these properties to select a recommender.

8.3.1 User Preference

As in this paper we are interested in the selection problem, where we need to choose one out of a set of candidate algorithms, an obvious option is to run a user study (within subjects) and ask the participants to choose one of the systems [25]. This evaluation does not restrict the subjects to specific properties, and it is generally easier for humans to make such judgments than to give scores for the experience. Then, we can select the system that had the largest number of votes.

However, aside from the biases in user studies discussed earlier, there are additional concerns that we must be aware of. First, the above scheme assumes that all users are equal, which may not always be true. For example, an e-commerce website may prefer the opinion of users who buy many items to the opinion of users who only buy a single item. We therefore need to further weight the vote by the importance of the user, when applicable. Assigning the right importance weights in a user study may not be easy.

It may also be the case that users who preferred system A, only slightly preferred it, while users who preferred B, had a very low opinion of A. In this case, even if more users preferred A we may still wish to choose B. To measure this we need non-binary answers for the preference question in the user study. Then, the problem of calibrating scores across users arises.

Finally, when we wish to improve a system, it is important to know why people favor one system over the other. Typically, it is easier to understand that when comparing specific properties. Therefore, while user satisfaction is important to mea-

sure, breaking satisfaction into smaller components is helpful to understand the system and improve it.

8.3.2 Prediction Accuracy

Prediction accuracy is by far the most discussed property in the recommendation system literature. At the base of the vast majority of recommender systems lie a prediction engine. This engine may predict user opinions over items (e.g. ratings of movies) or the probability of usage (e.g. purchase).

A basic assumption in a recommender system is that a system that provides more accurate predictions will be preferred by the user. Thus, many researchers set out to find algorithms that provide better predictions.

Prediction accuracy is typically independent of the user interface, and can thus be measured in an offline experiment. Measuring prediction accuracy in a user study measures the accuracy given a recommendation. This is a different concept from the prediction of user behavior without recommendations, and is closer to the true accuracy in the real system.

We discuss here three broad classes of prediction accuracy measures; measuring the accuracy of ratings predictions, measuring the accuracy of usage predictions, and measuring the accuracy of rankings of items.

8.3.2.1 Measuring Ratings Prediction Accuracy

In some applications, such as in the new releases page of the popular Netflix DVD rental service, we wish to predict the rating a user would give to an item (e.g. 1-star through 5-stars). In such cases, we wish to measure the accuracy of the system's predicted ratings.

Root Mean Squared Error (RMSE) is perhaps the most popular metric used in evaluating accuracy of predicted ratings. The system generates predicted ratings $\hat{r}_{ui}$ for a test set $\mathcal{T}$ of user-item pairs (u,i) for which the true ratings r_{ui} are known. Typically, r_{ui} are known because they are hidden in an offline experiment, or because they were obtained through a user study or online experiment. The RMSE between the predicted and actual ratings is given by:

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} (\hat{r}_{ui} - r_{ui})^2} \tag{8.2}$$

Mean Absolute Error (MAE) is a popular alternative, given by

$$\text{MAE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} |\hat{r}_{ui} - r_{ui}|} \tag{8.3}$$

Compared to MAE, RMSE disproportionately penalizes large errors, so that, given a test set with four hidden items RMSE would prefer a system that makes an error of 2 on three ratings and 0 on the fourth to one that makes an error of 3 on one rating and 0 on all three others, while MAE would prefer the second system.

Normalized RMSE (NMRSE) and **Normalized MAE (NMAE)** are versions of RMSE and MAE that have been normalized by the range of the ratings (i.e. $r_{max} - r_{min}$). Since they are simply scaled versions of RMSE and MAE, the resulting ranking of algorithms is the same as the ranking given by the unnormalized measures.

Average RMSE and **Average MAE** adjust for unbalanced test sets. For example, if the test set has an unbalanced distribution of items, the RMSE or MAE obtained from it might be heavily influenced by the error on a few very frequent items. If we need a measure that is representative of the prediction error on any item, it is preferable to compute MAE or RMSE separately for each item and then take the average over all items. Similarly, one can compute a per-user average RMSE or MAE if the test set has an unbalanced user distribution and we wish to understand the prediction error a randomly drawn user might face.

RMSE and MAE depend only on the magnitude of the errors made. In some applications, the semantics of the ratings may be such that the impact of a prediction error does not depend only on its magnitude. In such domains it may be preferable to use a suitable distortion measure $d(\hat{r}, r)$ than squared difference or absolute difference. For example in an application with a 3-star rating system where 1 means "disliked," 2 means "neutral" and 3 means "liked," and where recommending an item the user dislikes is worse that not recommending an item a user likes, a distortion measure with $d(3,1) = 5$, $d(2,1) = 3$, $d(3,2) = 3$, $d(1,2) = 1$, $d(2,3) = 1$, and $d(1,3) = 2$ may be reasonable.

8.3.2.2 Measuring Usage Prediction

In many applications the recommendation system does not predict the user's preferences of items, such as movie ratings, but tries to recommend to users items that they may use. For example, when movies are added to the queue, Netflix suggests a set of movies that may also be interesting, given the added movie. In this case we are interested not in whether the system properly predicts the ratings of these movies but rather whether the system properly predicts that the user will add these movies to the queue (use the items).

In an offline evaluation of usage prediction, we typically have a data set consisting of items each user has used. We then select a test user, hide some of her selections, and ask the recommender to predict a set of items that the user will use. We then have four possible outcomes for the recommended and hidden items, as shown in Table 8.1.

In the offline case, since the data isn't typically collected using the recommender system under evaluation, we are forced to assume that unused items would have not be used even if they had they been recommended — i.e. that they are uninteresting

	Recommended	Not recommended
Used	True-Positive (tp)	False-Negative (fn)
Not used	False-Positive (fp)	True-Negative (tn)

Table 8.1: Classification of the possible result of a recommendation of an item to a user.

or useless to the user. This assumption may be false, such as when the set of unused items contains some interesting items that the user did not select. For example, a user may not have used an item because she was unaware of its existence, but after the recommendation exposed that item the user can decide to select it. In this case the number of false positives is over estimated.

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

$$\textbf{Precision} = \frac{\#\text{tp}}{\#\text{tp} + \#\text{fp}} \tag{8.4}$$

$$\textbf{Recall (True Positive Rate)} = \frac{\#\text{tp}}{\#\text{tp} + \#\text{fn}} \tag{8.5}$$

$$\textbf{False Positive Rate } (1 - \textbf{Specificity}) = \frac{\#\text{fp}}{\#\text{fp} + \#\text{tn}} \tag{8.6}$$

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 applications where the number of recommendations that can be presented to the user is preordained, the most useful measure of interest is **Precision at N**.

In other applications where the number of recommendations that are presented to the user is not preordained, it is 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[3] 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 the 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 (i.e. the false positive rate) may not be

[3] A reference to their origins in signal detection theory.

as relevant as what proportion of the relevant items the system recommended to the user, so a precision-recall curve would be suitable for this application. On the other hand, consider a recommender system that is used for selecting items to be marketed to users, for example by mailing an item to the user who returns it at no cost to themselves if they do not purchase it. In this case, where we are interested in realizing as many potential sales as possible while minimizing marketing costs, ROC curves would be more relevant than precision-recall curves.

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 superior 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.

Measures that summarize the precision recall of ROC curve such as **F-measure** [58] and the **Area Under the ROC Curve (AUC)** [1] 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.

Precision-Recall and ROC for Multiple Users

When evaluating precision-recall or ROC curves for multiple test users, a number of strategies can be employed in aggregating the results, depending on the application at hand.

In applications where a fixed number of recommendations is made to each user (e.g. when a fixed number of headlines are shown to a user visiting a news portal), we can 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 [51]. 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. An ROC curve obtained in this manner is termed a **Customer ROC (CROC) curve** [52].

When different numbers of recommendations can be shown to each user (e.g. when presenting the set of all recommended movies to each user), we can compute ROC or precision-recall curves by aggregating the hidden ratings from the test set into a set of reference user-item pairs, using the recommender system to generate a single ranked list of user-item pairs, picking the top recommendations from the list, and scoring them against the reference set. An ROC curve calculated in this way is termed a **Global ROC (GROC) curve** [52]. Picking an operating point on the resulting curve can result in a different number of recommendations being made to each user.

A final class of applications is where the recommendation process is more interactive, and the user is able to obtain more and more recommendations. This is typical of information retrieval tasks, where the user can keep asking the system for

more recommended documents. In such applications, we 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 [59]. Such a curve can be used to understand the trade-off between precision and recall (or false positives and false negatives) a typical user would face.

8.3.2.3 Ranking Measures

In many cases the application presents to the user a list of recommendations, typically as vertical or horizontal list, imposing a certain natural browsing order. For example, in Netflix, the "movies you'll love" tab, shows a set of categories, and in each category, a list of movies that the system predicts the user to like. These lists may be long and the user may need to continue to additional "pages" until the entire list is browsed. In these applications, we are not interested in predicting an explicit rating, or selecting a set of recommended items, as in the previous sections, but rather in ordering items according to the user's preferences. This task is typically known as ranking of items. There are two approaches for measuring the accuracy of such a ranking. We can try to determine the correct order of a set of items for each user and measure how close a system comes to this correct order, or we can attempt to measure the utility of the system's raking to a user. We first describe these approaches for offline tests, and then describe their applicability to user studies and online tests.

Using a Reference Ranking

In order to evaluate a ranking algorithm with respect to a reference ranking (a correct order), it is first necessary to obtain such a reference. In cases where explicit user ratings of items are available, we can rank the rated items in decreasing order of the ratings, with ties. For example, in the case of Netflix, movies ranked by a user can be ranked in order of decreasing order of rating, with 5-star movies tied, followed by 4-star movies which are themselves tied, etc. In cases where we only have usage data, it may be appropriate to construct a reference ranking where items used by the user are ranked above unused items. However, this is only valid if we know that the user was aware of the unused items, so that we can infer that the user actually preferred the used items to the unused items. Thus, for example, logs from an online music application such as Pandora[4] can be used to construct a reference ranking where tracks that a user listened to are ranked above ones they skipped. Once again, used items should be tied with each other and unused items should be tied with each other.

[4] www.pandora.com

In both kinds of reference rankings above, two items are tied when we have no information about the user's relative ranking of the two. However, a recommender system used in an application such as Netflix's "movies you'll love" needs to strictly rank items with no ties. In such cases, a system under evaluation should not be penalized for ranking one item over another when they are tied in the reference ranking. The **Normalized Distance-based Performance Measure (NDPM)** measure [61] is suitable for such cases. If we have reference rankings r_{ui} and system rankings $\hat{r}_{ui}$ of n_u items i for user u, we can define

$$C^+ = \sum_{ij} \operatorname{sgn}(r_{ui} - r_{uj}) \operatorname{sgn}(\hat{r}_{ui} - \hat{r}_{uj}) \tag{8.7}$$

$$C^- = \sum_{ij} \operatorname{sgn}(r_{ui} - r_{uj}) \operatorname{sgn}(\hat{r}_{uj} - \hat{r}_{ui}) \tag{8.8}$$

$$C^u = \sum_{ij} \operatorname{sgn}^2(r_{ui} - r_{uj}) \tag{8.9}$$

$$C^s = \sum_{ij} \operatorname{sgn}^2(\hat{r}_{ui} - \hat{r}_{uj}) \tag{8.10}$$

$$C^{u0} = C^u - (C^+ + C^-) \tag{8.11}$$

where the sums range over the $\frac{1}{2} n_u (n_u - 1)$ pairs of items. Thus, C^u is the number of pairs of items for which the reference ranking asserts an ordering (i.e. not tied), while C^+ and C^- are the number of these pairs that the system ranking asserts the correct order and the incorrect order respectively. C^{u0} is the number of pairs where the reference ranking does not tie but the system ranking ties. The NDPM is then given by

$$\text{NDPM} = \frac{C^- + 0.5 C^{u0}}{C^u} \tag{8.12}$$

Thus, the NDPM measure gives a perfect score of 0 to systems that correctly predicts every preference relation asserted by the reference. The worst score of 1 is assigned to systems that contradict every reference preference relation. Not predicting a reference preference relation is penalized only half as much as contradicting it. Predicting preferences that the reference does not order (i.e. when we do not know the user's true preference) is not penalized.

In some cases, we may completely know the user's true preferences for some set of items. For example, we may elicit the user's true ordering by presenting the user with binary choices. In this case, when a pair of items are tied in the reference ranking it means that the user is actually indifferent between the items. Thus, a perfect system should not rank one item higher than the other. In such cases, rank correlation measures such as **Spearman's** ρ or **Kendall's** τ [30, 31] can be used. These measures tend to be highly correlated in practice [18]. Kendall's τ is given by

$$\tau = \frac{C^+ - C^-}{\sqrt{C^u}\sqrt{C^s}} \tag{8.13}$$

while Spearman's ρ is given by

$$\rho = \frac{1}{n_u} \frac{\sum_i (r_{i,u} - \bar{r})(\hat{r}_{i,u} - \bar{\hat{r}})}{\sigma(r)\sigma(\hat{r})} \tag{8.14}$$

where $\bar{\cdot}$ and $\sigma(\cdot)$ denote the mean and standard deviation. Note that in the case of ties, each tied item should be assigned the average ranking, so that, e.g., if two items are tied for second and third place, they are assigned a rank of 2.5 [31].

Utility-based ranking

While using a reference ranking scores a ranking on its correlation with some "true" ranking, there are other criteria for deciding on ordering a list of items. One popular alternative is to assume that the utility of a list of recommendations is additive, given by the sum of the utilities of the individual recommendations. The utility of each recommendation is the utility of the recommended item discounted by a factor that depends on its position in the list of recommendations. One example of such a utility is the likelihood that a user will observe a recommendation at position i in the list. It is usually assumed that users scan recommendation lists from the beginning to the end, with the utility of recommendations being discounted more heavily towards the end of the list. The discount can also be interpreted as the probability that a user would observe a recommendation in a particular position in the list, with the utility of the recommendation given that it was observed depending only on the item recommend. Under this interpretation, the probability that a particular position in the recommendation list is observed is assumed to depend only on the position and not on the items that are recommended.

In many applications, the user can use only a single, or a very small set of items, or the recommendation engine is not used as the main browsing tool. In such cases, we can expect the users to observe only a few items of the top of the recommendations list. We can model such applications using a very rapid decay of the positional discount down the list. The **R-Score** metric [8] assumes that the value of recommendations decline exponentially down the ranked list to yield the following score for each user u:

$$R_u = \sum_u \sum_j \frac{\max(r_{ui_j} - d, 0)}{2^{\frac{j-1}{\alpha-1}}} \tag{8.15}$$

where i_j is the item in the jth position, r_{ui} is user u's rating of item i, d is a task dependent neutral ("don't care") rating, and α is a half-life parameter, which controls the exponential decline of the value of positions in the ranked list. In the case of ratings prediction tasks, r_{ui} is the rating given by the user to each item (e.g. 4 stars), and d is the don't care vote (e.g. 3 stars), and the algorithm only gets credit for ranking items with rating above the "don't care" vote higher than d (e.g. 4 or 5 stars). In usage prediction tasks, r_{ui} is typically 1 if u selects i and 0 otherwise, while

d is 0. Using $r_{ui} = -\log(\text{popularity}(i))$ if i is used and 0 otherwise [54] can capture the amount of information in the recommendation. The resulting per-user scores are aggregated using:

$$R = 100 \frac{\sum_u R_u}{\sum_u R_u^*} \tag{8.16}$$

where R_u^* is the score of the best possible ranking for user u.

In other applications the user is expected to read a relatively large portion of the list. In certain types of search, such as the search for legal documents [24], users may look for all relevant items, and would be willing to read large portions of the recommendations list. In such cases, we need a much slower decay of the positional discount. **Normalized Cumulative Discounted Gain (NDCG)** [27] is a measure from information retrieval, where positions are discounted logarithmically. Assuming each user u has a "gain" g_{ui} from being recommended an item i, the average Discounted Cumulative Gain (DCG) for a list of J items is defined as

$$\text{DCG} = \frac{1}{N} \sum_{u=1}^{N} \sum_{j=1}^{J} \frac{g_{ui_j}}{\max(1, \log_b j)} \tag{8.17}$$

where the logarithm base is a free parameter, typically between 2 and 10. A logarithm with base 2 is commonly used to ensure all positions are discounted. NDCG is the normalized version of DCG given by

$$\text{NDCG} = \frac{\text{DCG}}{\text{DCG}^*} \tag{8.18}$$

where DCG^* is the ideal DCG.

We show the two methods here as they were originally presented, but note that the numerator in the two cases contains a utility function that assigns a value for each item. One can replace the original utility functions with a function that is more appropriate to the designed application. A measure closely related to R-score and NDCG is **Average Reciprocal Hit Rank (ARHR)** [14] which is an un-normalized measure that assigns a utility $1/k$ to a successful recommendation at position k. Thus, ARHR decays more slowly than R score but faster than NDCG.

Online evaluation of ranking

In an online experiment designed to evaluate the ranking of the recommendation list, we can look at the interactions of users with the system. When a recommendation list is presented to a user, the user may select a number of items from the list. We can now assume that the user has scanned the list at least as deep as the last selection. That is, if the user has selected items 1, 3, and 10, we can assume that the user has observed items 1 through 10. We can now make another assumption, that the user has found items 1,3, and 10 to be interesting, and items 2,4,5,6,7,8, and 9 to be

uninteresting. In some cases we can have additional information whether the user has observed more items. For example, if the list is spread across several pages, and only 20 results are presented per page, then, in the example above, if the user moved to the second page we can also assume that she has observed results 11 through 20 and had found them to be irrelevant.

In the scenario above, the results of this interaction is a division of the list into 3 parts — the interesting items (1,3,10 in the example above), the uninteresting items (the rest of the items from 1 through 20), and the unknown items (21 till the end of the list). We can now use an appropriate reference ranking metric to score the original list. This can be done in two different ways. First, the reference list can contain the interesting items at the top, then the unknown items, and the uninteresting items at the bottom. This reference list captures the case where the user may only select a small subset of the interesting items, and therefore the unknown items may contain more interesting items. Second, the reference list can contain the interesting items at the top, followed by the uninteresting items, with the unknown items completely ignored. This is useful when making unreasonable preference assumptions, such as that some unknown items are preferred to the uninteresting items, may have negative consequences. In either case, it should be borne in mind that the semantics of the reference ranking are different from the case of offline evaluations. In offline evaluations, we have a single reference ranking which is assumed to be correct, and we measure how much each recommender deviates from this "correct" ranking. In the online case, the reference ranking is assumed to be the ranking that the user would have preferred given that were presented with the recommender's ranking. In the offline case, we assume that there is one correct ranking, while in the online case we allow for the possibility of multiple correct rankings.

In the case of utility ranking, we can evaluate a list based on the sum of the utilities of the selected items. Lists that place interesting items with high utility close to the beginning of the list, will hence be preferred to lists that place these interesting items down the list, because we expect that in the latter case, the user will often not observe these interesting items at all, generating no utility for the recommender.

8.3.3 Coverage

As the prediction accuracy of a recommendation system, especially in collaborative filtering systems, in many cases grows with the amount of data, some algorithms may provide recommendations with high quality, but only for a small portion of the items where they have huge amounts of data. The term coverage can refer to several distinct properties of the system that we discuss below.

8.3.3.1 Item Space Coverage

Most commonly, the term coverage refers to the proportion of items that the recommendation system can recommend. This is often referred to as **catalog coverage**. The simplest measure of catalog coverage is the percentage of all items that can ever be recommended. This measure can be computed in many cases directly given the algorithm and the input data set.

A more useful measure is the percentage of all items that are recommended to users during an experiment, either offline, online, or a user study. In some cases it may be desirable to weight the items, for example, by their popularity or utility. Then, we may agree not to be able to recommend some items which are very rarely used anyhow, but ignoring high profile items may be less tolerable.

Another measure of catalog coverage is the **sales diversity** [16], which measures how unequally different items are chosen by users when a particular recommender system is used. If each item i accounts for a proportion $p(i)$ of user choices, the **Gini Index** is given by:

$$G = \frac{1}{n-1} \sum_{j=1}^{n} (2j - n - 1) p(i_j) \tag{8.19}$$

where $i_1, \cdots i_n$ is the list of items ordered according to increasing $p(i)$. The index is 0 when all items are chosen equally often, and 1 when a single item is always chosen. The Gini index of the number of times each item is recommended could also be used. Another measure of distributional inequality is the **Shannon Entropy**:

$$H = -\sum_{i=1}^{n} p(i) \log p(i) \tag{8.20}$$

The entropy is 0 when a single item is always chosen or recommended, and $\log n$ when n items are chosen or recommended equally often.

8.3.3.2 User Space Coverage

Coverage can also be the proportion of users or user interactions for which the system can recommend items. In many applications the recommender may not provide recommendations for some users due to, e.g. low confidence in the accuracy of predictions for that user. In such cases we may prefer recommenders that can provide recommendations for a wider range of users. Clearly, such recommenders should be evaluated on the trade-off between coverage and accuracy.

Coverage here can be measured by the richness of the user profile required to make a recommendation. For example, in the collaborative filtering case this could be measured as the number of items that a user must rate before receiving recommendations. This measurement can be typically evaluated in offline experiments.

8.3.3.3 Cold Start

Another, related set of issues are the well known cold start problems — the performance of the system on new items and on new users. Cold start can be considered as a sub problem of coverage because it measures the system coverage over a specific set of items and users. In addition to measuring how large the pool of cold start items or users are, it may also be important to measure system accuracy for these users and items.

Focusing on cold start items, we can use a threshold to decide on the set of cold items. For example, we can decide that cold items are only items with no ratings or usage evidence [52], or items that exist in the system for less than a certain amount of time (e.g., a day), or items that have less than a predefined evidence amount (e.g., less than 10 ratings). Perhaps a more generic way is to consider the "coldness" of an item using either the amount of time it exists in the system or the amount of data gathered for it. Then, we can credit the system more for properly predicting colder items, and less for the hot items that are predicted.

It may be possible that a system better recommends cold items at the price of a reduced accuracy for hotter items. This may be desirable due to other considerations such as novelty and serendipity that are discussed later. Still, when computing the system accuracy on cold items it may be wise to evaluate whether there is a trade-off with the entire system accuracy.

8.3.4 Confidence

Confidence in the recommendation can be defined as the system's trust in its recommendations or predictions [128, 22]. As we have noted above, collaborative filtering recommenders tend to improve their accuracy as the amount of data over items grows. Similarly, the confidence in the predicted property typically also grows with the amount of data.

In many cases the user can benefit from observing these confidence scores [22]. When the system reports a low confidence in a recommended item, the user may tend to further research the item before making a decision. For example, if a system recommends a movie with very high confidence, and another movie with the same rating but a lower confidence, the user may add the first movie immediately to the watching queue, but may further read the plot synopsis for the second movie, and perhaps a few movie reviews before deciding to watch it.

Perhaps the most common measurement of confidence is the probability that the predicted value is indeed true, or the interval around the predicted value where a predefined portion, e.g. 95% of the true values lie. For example, a recommender may accurately rate a movie as a 4 star movie with probability 0.85, or have 95% of the actual ratings lie within -1 and $+\frac{1}{2}$ of the predicted 4 stars. The most general method of confidence is to provide a complete distribution over possible outcomes [40].

Given two recommenders that perform similarly on other relevant properties, such as prediction accuracy, is can be desirable to choose the one that can provide valid confidence estimates. In this case, given two recommenders with, say, identical accuracy, that report confidence bounds in the same way, we will prefer the recommender that better estimates its confidence bounds.

Standard confidence bounds, such as the ones above, can be directly evaluated in regular offline trials, much the same way as we estimate prediction accuracy. We can design for each specific confidence type a score that measures how close the method confidence estimate is to the true error in prediction. This procedure cannot be applied when the algorithms do not agree on the confidence method, because some confidence methods are weaker and therefore easier to estimate. In such a case a more accurate estimate of a weaker confidence metric does not imply a better recommender.

Example 8.1. Recommenders A and B both report confidence intervals over possible movie ratings. We train A and B over a confidence threshold, ranging of 95%. For each trained model, we run A and B on offline data, hiding a part of the user ratings and requesting each algorithm to predict the missing ratings. Each algorithm produces, along with the predicted rating, a confidence interval. We compute A_+ and A_-, the number of times that the predicted rating of algorithm A was within and outside the confidence interval (respectively), and do the same for B. Then we compute the true confidence of each algorithm using $\frac{A_+}{A_-+A_+} = 0.97$ and $\frac{B_+}{A_-+A_+} = 0.94$. The result indicates that A is over conservative, and computes intervals that are too large, while B is liberal and computes intervals that are too small. As we do not require the intervals to be conservative, we prefer B because its estimated intervals are closer to the requested 95% confidence.

Another application of confidence bounds is in filtering recommended items where the confidence in the predicted value is below some threshold. In this scenario we assume that the recommender is allowed not to predict a score for all values, as is typically the case when presenting top n recommendations. We can hence design an experiment around this filtering procedure by comparing the accuracy of two recommenders after their results were filtered by removing low confidence items. In such experiments we can compute a curve, estimating the prediction accuracy for each portion of filtered items, or for different filtering thresholds. This evaluation procedure does not require both algorithms to agree on the confidence method.

While user studies and online experiments can study the effect of reporting confidence on the user experience, it is difficult to see how these types of tests can be used to provide further evidence as to the accuracy of the confidence estimate.

8.3.5 Trust

While confidence is the system trust in its ratings, in trust we refer here to the user's trust in the system recommendation[5]. For example, it may be beneficial for the system to recommend a few items that the user already knows and likes. This way, even though the user gains no value from this recommendation, she observes that the system provides reasonable recommendations, which may increase her trust in the system recommendations for unknown items. Another common way of enhancing trust in the system is to explain the recommendations that the system provides (see Chapter 15). Trust in the systems is also called the credibility of the system.

If we do not restrict ourselves to a single method of gaining trust, such as the one suggested above, the obvious method for evaluating user trust is by asking users whether the system recommendations are reasonable in a user study [22, 12, 11, 47]. In an online test one could associate the number of recommendations that were followed with the trust in the recommender, assuming that higher trust in the recommender would lead to more recommendations being used. Alternatively, we could also assume that trust in the system is correlated with repeated users, as users who trust the system will return to it when performing future tasks. However, such measurements may not separate well other factors of user satisfaction, and may not be accurate. It is unclear how to measure trust in an offline experiment, because trust is built through an interaction between the system and a user.

8.3.6 Novelty

Novel recommendations are recommendations for items that the user did not know about [33]. In applications that require novel recommendation, an obvious and easy to implement approach is to filter out items that the user already rated or used. However, in many cases users will not report all the items they have used in the past. Thus, this simple method is insufficient to filter out all items that the user already knows.

While we can obviously measure novelty in a user study, by asking users whether they were already familiar with a recommended item [9, 28], we can also gain some understanding of a system's novelty through offline experiments. For such an experiment we could split the data set on time, i.e. hide all the user ratings that occurred after a specific point in time. In addition, we can hide some ratings that occurred prior to that time, simulating the items that the user is familiar with, but did not report ratings for. When recommending, the system is rewarded for each item that was recommended and rated after the split time, but would be punished for each item that was recommended but rated prior to the split time.

[5] Not to be confused with trust in the social network research, used to measure how much a user believes another user. Some literature on recommender systems uses such trust measurements to filter similar users [39] and Chapter 20.

To implement the above procedure we must carefully model the hiding process such that it would resemble the true preference discovery process that occurs in the real system. In some cases the set of rated items is not a uniform sample of the set of all items the user is familiar with, and such bias should be acknowledged and handled if possible. For example, if we believe that the user will provide more ratings about special items, but less ratings for popular items, then the hiding process should tend to hide more popular items.

In using this measure of novelty, it is important to control for accuracy, as irrelevant recommendations may be new to the user, but still worthless. One approach would be to consider novelty only among the relevant items [63].

Example 8.2. We wish to evaluate the novelty of a set of movie recommenders in an offline test. As we believe that users of our system rate movies after they watch them, we split the user ratings in a sequential manner. For each test user profile we choose a cutoff point randomly along the time-based sequence of movie ratings, hiding all movies after a certain point in the sequence.

User studies on our system showed that people tend not to report ratings of movies that they did not feel strongly about, but occasionally also do not report a rating of a movie that they liked or disliked strongly. Therefore, we hide a rating of a movie prior to the cutoff point with probability $1 - \frac{|r-3|}{2}$ where $r \in \{1,2,3,4,5\}$ is the rating of the movie, and 3 is the neutral rating. We would like to avoid predicting these movies with hidden ratings because the user already knows about them.

Then, for each user, each recommender produces a list of 5 recommendations, and we compute precision only over items after the cutoff point. That is, the recommenders get no credit for recommending movies with hidden ratings that occurred prior to the cutoff point. In this experiment the algorithm with the highest precision score is preferred.

Another method for evaluating novel recommendations uses the above assumption that popular items are less likely to be novel. Thus, novelty can be taken into account be using an accuracy metric where the system does not get the same credit for correctly predicting popular items as it does when it correctly predicts non-popular items [53]. Ziegler *et al.* [64] and Celma and Herrera [9] also give accuracy measures that take popularity into account.

Finally, we can evaluate the amount of new information in a recommendation together with the relevance of the recommended item. For example, when item ratings are available, we can multiply the hidden rating by some information measurement of the recommended item (such as the conditional entropy given the user profile) to produce a novelty score.

8.3.7 Serendipity

Serendipity is a measure of how surprising the successful recommendations are. For example, if the user has rated positively many movies where a certain star actor

appears, recommending the new movie of that actor may be novel, because the user may not know of it, but is hardly surprising. Of course, random recommendations may be very surprising, and we therefore need to balance serendipity with accuracy.

One can think of serendipity as the amount of relevant information that is new to the user in a recommendation. For example, if following a successful movie recommendation the user learns of a new actor that she likes, this can be considered as serendipitous. In information retrieval, where novelty typically refers to the new information contained in the document (and is thus close to our definition of serendipity), Zhang *et al*. [63] suggested to manually label pairs of documents as redundant. Then, they compared algorithms on avoiding recommending redundant documents. Such methods are applicable to recommender systems when some meta-data over items, such as content information, is available.

To avoid human labeling, we could design a distance measurement between items based on content (see Chapter 3). Then, we can score a successful recommendation by its distance from a set of previously rated items in a collaborative filtering system, or from the user profile in a content-based recommender [62]. Thus, we are rewarding the system for successful recommendations that are far from the user profile.

Example 8.3. In a book recommendation application, we would like to recommend books from authors that the reader is less familiar with. We therefore design a distance metric between a book b and a set of books B (the books that the user has previously read); Let $c_{B,w}$ be the number of books by writer w in B. Let $c_B = \max_w c_{B,w}$ the maximal number of books from a single writer in B. Let $d(b,B) = \frac{1+c_B-c_{B,w(b)}}{1+c_B}$, where $w(b)$ is the writer of book b.

We now run an offline experiment to evaluate which of the candidate algorithms generates more serendipitous recommendations. We split each test user profile — set of books that the user has read — into sets of observed books B_i^o and hidden books B_i^h. We use B_i^o as the input for each recommender, and request a list of 5 recommendations. For each hidden book $b \in B_i^h$ that appeared in the recommendation list for user i, the recommender receives a score of $d(b,B_i^o)$. Thus the recommender is getting more credit for recommending books from writers that the reader has read less often. In this experiment the recommender that received a higher score is selected for the application.

One can also think of serendipity as deviation from the "natural" prediction [44]. That is, given a prediction engine that has a high accuracy, the recommendations that it issues are "obvious". Therefore, we will give higher serendipity scores to successful recommendations that the prediction engine would deem unlikely.

We can evaluate the serendipity of a recommender in a user study by asking the users to mark the recommendations that they find unexpected. Then, we can also see whether the user followed these recommendations, which would make them unexpected and successful and therefore serendipitous. In an online study, we can assume that our distance metric is correct and evaluate only how distance from the user profile affected the probability that a user will follow the recommendation. It is

important to check the effect of serendipity over time, because users might at first be intrigued by the unexpected recommendations and try them out. If after following the suggestion they discover that the recommendations are inappropriate, they may stop following them in the future, or stop using the recommendation engine at all.

8.3.8 Diversity

Diversity is generally defined as the opposite of similarity. In some cases suggesting a set of similar items may not be as useful for the user, because it may take longer to explore the range of items. Consider for example a recommendation for a vacation [55], where the system should recommend vacation packages. Presenting a list with 5 recommendations, all for the same location, varying only on the choice of hotel, or the selection of attraction, may not be as useful as suggesting 5 different locations. The user can view the various recommended locations and request more details on a subset of the locations that are appropriate to her.

The most explored method for measuring diversity uses item-item similarity, typically based on item content, as in Section 8.3.7. Then, we could measure the diversity of a list based on the sum, average, min, or max distance between item pairs, or measure the value of adding each item to the recommendation list as the new item's diversity from the items already in the list [64, 6]. The item-item similarity measurement used in evaluation can be different from the similarity measurement used by the algorithm that computes the recommendation lists. For example, we can use for evaluation a costly metric that produces more accurate results than fast approximate methods that are more suitable for online computations.

As diversity may come at the expanse of other properties, such as accuracy [62], we can compute curves to evaluate the decrease in accuracy vs. the increase in diversity.

Example 8.4. In a book recommendation application, we are interested in presenting the user with a diverse set of recommendations, with minimal impact to accuracy. We use $d(b,B)$ from Example 8.3 as the distance metric. Given candidate recommenders, each with a tunable parameter that controls the diversity of the recommendations, we train each algorithm over a range of values for the diversity parameters. For each trained model, we now compute a precision score, and a diversity score as follows; we take each recommendation list that an algorithm produces, and compute the distance of each item from the rest of the list, averaging the result to obtain a diversity score. We now plot the precision-diversity curves of the recommenders in a graph, and select the algorithm with the dominating curve.

In recommenders that assist in information search (see Chapter 18), we can assume that more diverse recommendations will result in shorter search interactions [55]. We could use this in an online experiment measuring interaction sequence length as a proxy for diversification. As is always the case in online testing, shorter sessions may be due to other factors of the system, and to validate this claim it is

useful to experiment with different diversity thresholds using the same prediction engine before comparing different recommenders.

8.3.9 Utility

Many e-commerce websites employ a recommendation system in order to improve their revenue by, e.g., enhancing cross-sell. In such cases the recommendation engine can be judged by the revenue that it generates for the website [54]. In general, we can define various types of utility functions that the recommender tries to optimize. For such recommenders, measuring the utility, or the expected utility of the recommendations may be more significant than measuring the accuracy of recommendations. It is also possible to view many of the other properties, such as diversity or serendipity, as different types of utility functions, over single items or over lists. In this paper, however, we define utility as the value that either the system or the user gains from a recommendation.

Utility can be measured cleanly from the perspective of the recommendation engine or the recommender system owner. Care must be taken, though, when measuring the utility that the user receives from the recommendations. First, user utilities or preferences are difficult to capture and model, and considerable research has focused on this problem [7, 21, 48]. Second, it is unclear how to aggregate user utilities across users for computing a score for a recommender. For example, it is tempting to use money as a utility thus selecting a recommender that minimizes user cost. However, under the diminishing returns assumption [56], the same amount of money does not have the same utility for people with different income levels. Therefore, the average cost per purchase, for example, is not a reasonable aggregation across users.

In an application where users rate items, it is also possible to use the ratings as a utility measurement [8]. For example, in movie ratings, where a 5 star movie is considered an excellent movie, we can assume that a recommending a 5 star movie has a higher utility for the user than recommending a movie that the user will rate with 4 stars. As users may interpret ratings differently, user ratings should be normalized before aggregating across users.

While we typically only assign positive utilities to successful recommendations, we can also assign negative utilities to unsuccessful recommendations. For example, if some recommended item offends the user, then we should punish the system for recommending it by assigning a negative utility. We can also add a cost to each recommendation, perhaps based on the position of the recommended item in the list, and subtract it from the utility of the item.

For any utility function, the standard evaluation of the recommender is to compute the expected utility of a recommendation. In the case where the recommender is trying to predict only a single item, such as when we evaluate the system on time-based splits and try to predict only the next item in the sequence, the value of a correct recommendation should simply be the utility of the item. In the task where

the recommender predicts n items we can use the sum of the utilities of the correct recommendations in the list. When negative utilities for failed recommendations are used, then the sum is over all recommendations, successful or failed. We can also integrate utilities into ranking measurements, as discussed in Section 8.3.2.3. Finally, we can normalize the resulting score using the maximal possible utility given the optimal recommendation list.

Evaluating utility in user studies and online is easy in the case of recommender utility. If the utility we optimize for is the revenue of the website, measuring the change in revenue between users of various recommenders is simple. When we try to optimize user utilities the online evaluation becomes harder, because users typically find it challenging to assign utilities to outcomes. In many cases, however, users can say whether they prefer one outcome to another. Therefore, we can try to elicit the user preferences [26] in order to rank the candidate methods.

8.3.10 Risk

In some cases a recommendation may be associated with a potential risk. For example, when recommending stocks for purchase, users may wish to be risk-averse, preferring stocks that have a lower expected growth, but also a lower risk of collapsing. On the other hand, users may be risk-seeking, preferring stocks that have a potentially high, even if less likely, profit. In such cases we may wish to evaluate not only the (expected) value generated from a recommendation, but also to minimize the risk.

The standard way to evaluate risk sensitive systems is by considering not just the expected utility, but also the utility variance. For example, we may use a parameter q and compare two systems on $E[X] + q \cdot Var(X)$. When q is positive, this approach prefers risk-seeking (also called bold [41]) recommenders, and when q is negative, the system prefers risk-averse recommenders.

8.3.11 Robustness

Robustness (see Chapter 25) is the stability of the recommendation in the presence of fake information [45], typically inserted on purpose in order to influence the recommendations. As more people rely on recommender systems to guide them through the item space, influencing the system to change the rating of an item may be profitable to an interested party. For example, an owner of an hotel may wish to boost the rating for their hotel. This can be done by injecting fake user profiles that rate the hotel positively, or by injecting fake users that rate the competitors negatively.

Such attempts to influence the recommendation are typically called attacks [43, 35]. Coordinated attacks occur when a malicious user intentionally queries the data

set or injects fake information in order to learn some private information of some users. In evaluating such systems, it is important to provide a complete description of the attack protocol, as the sensitivity of the system typically varies from one protocol to another.

In general, creating a system that is immune to any type of attack is unrealistic. An attacker with an ability to inject an infinite amount of information can, in most cases, manipulate a recommendation in an arbitrary way. It is therefore more useful to estimate the cost of influencing a recommendation, which is typically measured by the amount of injected information. While it is desirable to theoretically analyze the cost of modifying a rating, it is not always possible. In these cases, we can simulate a set of attacks by introducing fake information into the system data set, empirically measuring average cost of a successful attack [36, 10].

As opposed to other evaluation criteria discussed here, it is hard to envision executing an attack on a real system as an online experiment. It may be fruitful, however, to analyze the real data collected in the online system to identify actual attacks that are executed against the system.

Another type of robustness is the stability of the system under extreme conditions, such as a large number of requests. While less discussed, such robustness is very important to system administrators, who must avoid system malfunction. In many cases system robustness is related to the infrastructure, such as the database software, or to the hardware specifications, and is related to scalability (Section 8.3.14).

8.3.12 Privacy

In a collaborative filtering system, a user willingly discloses his preferences over items to the system in the hope of getting useful recommendations. However, it is important for most users that their preferences stay private, that is, that no third party can use the recommendation system to learn something about the preferences of a specific user.

For example, consider the case where a user who is interested in the wonderful, yet rare art of growing Bahamian orchids has bought a book titled "The Divorce Organizer and Planner". The spouse of that user, looking for a present, upon browsing the book "The Bahamian and Caribbean Species (Cattleyas and Their Relatives)" may get a recommendation of the type "people who bought this book also bought" for the divorce organizer, thus revealing sensitive private information.

It is generally considered inappropriate for a recommendation system to disclose private information even for a single user. For this reason analysis of privacy tends to focus on a worst case scenario, illustrating theoretical cases under which users private information may be revealed. Other researchers [17] compare algorithms by evaluating the portion of users whose private information was compromised. The assumption in such studies is that complete privacy is not realistic and that therefore we must compromise on minimizing the privacy breaches.

Another alternative is to define different levels of privacy, such as k-identity [17], and compare algorithms sensitivity to privacy breaches under varying levels of privacy.

Privacy may also come at the expense of the accuracy of the recommendations. Therefore, it is important to analyze this trade-off carefully. Perhaps the most informative experiment is when a privacy modification has been added to an algorithm, and the accuracy (or any other trade-off property) can be evaluated with or without the modification [42].

8.3.13 Adaptivity

Real recommendation systems may operate in a setting where the item collection changes rapidly, or where trends in interest over items may shift. Perhaps the most obvious example of such systems is the recommendation of news items or related stories in online newspapers [19]. In this scenario stories may be interesting only over a short period of time, afterwards becoming outdated. When an unexpected news event occurs, such as the tsunami disaster, people become interested in articles that may not have been interesting otherwise, such as a relatively old article explaining the tsunami phenomenon. While this problem is similar to the cold-start problem, it is different because it may be that old items that were not regarded as interesting in the past suddenly become interesting.

This type of adaptation can be evaluated offline by analyzing the amount of information needed before an item is recommended. If we model the recommendation process in a sequential manner, we can record, even in an offline test, the amount of evidence that is needed before the algorithm recommends a story. It is likely that an algorithm can be adjusted to recommend items faster once they become interesting, by sacrificing some prediction accuracy. We can compare two algorithms by evaluating a possible trade-off between accuracy and the speed of the shift in trends.

Another type of adaptivity is the rate by which the system adapts to a user's personal preferences [37], or to changes in user profile [34]. For example, when users rate an item, they expect the set of recommendations to change. If the recommendations stay fixed, users may assume that their rating effort is wasted, and may not agree to provide more ratings. As with the shift in trends evaluation, we can again evaluate in an offline experiment the changes in the recommendation list after adding more information to the user profile such as new ratings. We can evaluate an algorithm by measuring the difference between the recommendation lists before and after the new information was added. The Gini index and Shannon entropy measures discussed in Section 8.3.3 can be used to measure the variability of recommendations made to a user as the user profile changes.

8.3.14 Scalability

As recommender systems are designed to help users navigate in large collections of items, one of the goals of the designers of such systems is to scale up to real data sets. As such, it is often the case that algorithms trade other properties, such as accuracy or coverage, for providing rapid results even for huge data sets consisting of millions of items (e.g. [12]).

With the growth of the data set, many algorithms are either slowed down or require additional resources such as computation power or memory. One standard approach in computer science research is to evaluate the computational complexity of an algorithm in terms of time or space requirements (as done, e.g., in [29, 4]). In many cases, however, the complexity of two algorithms is either identical, or could be reduced by changing some parameters, such as the complexity of the model, or the sample size. Therefore, to understand the scalability of the system it is also useful to report the consumption of system resources over large data sets.

Scalability is typically measured by experimenting with growing data sets, showing how the speed and resource consumption behave as the task scales up (see, e.g. [19]). It is important to measure the compromises that scalability dictates. For example, if the accuracy of the algorithm is lower than other candidates that only operate on relatively small data sets, one must show over small data sets the difference in accuracy. Such measurements can provide valuable information both on the potential performance of recommender systems in general for the specific task, and on future directions to explore.

As recommender systems are expected in many cases to provide rapid recommendations online, it is also important to measure how fast does the system provides recommendations [23, 50]. One such measurement is the throughput of the system, i.e., the number of recommendations that the system can provide per second. We could also measure the latency (also called response time) — the required time for making a recommendation online.

8.4 Conclusion

In this paper we discussed how recommendation algorithms could be evaluated in order to select the best algorithm 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 describe the concerns that need to be addressed when designing offline and online experiments and user studies. 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.

We specify a set of properties that are sometimes discussed as important for the recommendation system. For each such property we suggest an experiment that can be used to rank recommenders with regards to that property. For less explored properties, we restrict ourselves to generic descriptions that could be applied to various manifestations of that property. Specific procedures that can be practically implemented can then be developed for the specific property manifestation based on our generic guidelines.

References

1. Bamber, D.: The area above the ordinal dominance graph and the area below the receiver operating characteristic graph. Journal of Mathematical Psychology **12**, 387–415 (1975)
2. benjamini: Controlling the false discovery rate:a practical and powerful approach to multiple testing. J. R. Statist. Soc. B **57**(1), 289–300 (1995)
3. Bonhard, P., Harries, C., McCarthy, J., Sasse, M.A.: Accounting for taste: using profile similarity to improve recommender systems. In: CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems, pp. 1057–1066. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1124772.1124930
4. Boutilier, C., Zemel, R.S.: Online queries for collaborative filtering. In: In Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics (2002)
5. Box, G.E.P., Hunter, W.G., Hunter, J.S.: Statistics for Experimenters. Wiley, New York (1978)
6. Bradley, K., Smyth, B.: Improving recommendation diversity. In: Twelfth Irish Conference on Artificial Intelligence and Cognitive Science, pp. 85–94 (2001)
7. Braziunas, D., Boutilier, C.: Local utility elicitation in GAI models. In: Proceedings of the Twenty-first Conference on Uncertainty in Artificial Intelligence, pp. 42–49. Edinburgh (2005)
8. Breese, J.S., Heckerman, D., Kadie, C.M.: Empirical analysis of predictive algorithms for collaborative filtering. In: UAI, pp. 43–52 (1998)
9. Celma, O., Herrera, P.: A new approach to evaluating novel recommendations. In: RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems, pp. 179–186. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1454008.1454038
10. Chirita, P.A., Nejdl, W., Zamfir, C.: Preventing shilling attacks in online recommender systems. In: WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management, pp. 67–74. ACM, New York, NY, USA (2005). DOI http://doi.acm.org/10.1145/1097047.1097061
11. Cramer, H., Evers, V., Ramlal, S., Someren, M., Rutledge, L., Stash, N., Aroyo, L., Wielinga, B.: The effects of transparency on trust in and acceptance of a content-based art recommender. User Modeling and User-Adapted Interaction **18**(5), 455–496 (2008). DOI http://dx.doi.org/10.1007/s11257-008-9051-3
12. Das, A.S., Datar, M., Garg, A., Rajaram, S.: Google news personalization: scalable online collaborative filtering. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, pp. 271–280. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1242572.1242610
13. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
14. Deshpande, M., Karypis, G.: Item-based top-N recommendation algorithms. ACM Transactions on Information Systems **22**(1), 143–177 (2004)
15. Fischer, G.: User modeling in human-computer interaction. User Model. User-Adapt. Interact. **11**(1-2), 65–86 (2001)

16. Fleder, D.M., Hosanagar, K.: Recommender systems and their impact on sales diversity. In: EC '07: Proceedings of the 8th ACM conference on Electronic commerce, pp. 192–199. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1250910.1250939
17. Frankowski, D., Cosley, D., Sen, S., Terveen, L., Riedl, J.: You are what you say: privacy risks of public mentions. In: SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 565–572. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1148170.1148267
18. Fredricks, G.A., Nelsen, R.B.: On the relationship between spearman's rho and kendall's tau for pairs of continuous random variables. Journal of Statistical Planning and Inference **137**(7), 2143–2150 (2007)
19. George, T.: A scalable collaborative filtering framework based on co-clustering. In: Fifth IEEE International Conference on Data Mining, pp. 625–628 (2005)
20. Greenwald, A.G.: Within-subjects designs: To use or not to use? Psychological Bulletin **83**, 216–229 (1976)
21. Haddawy, P., Ha, V., Restificar, A., Geisler, B., Miyamoto, J.: Preference elicitation via theory refinement. Journal of Machine Learning Research **4**, 2003 (2002)
22. Herlocker, J.L., Konstan, J.A., Riedl, J.T.: Explaining collaborative filtering recommendations. In: CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work, pp. 241–250. ACM, New York, NY, USA (2000). DOI http://doi.acm.org/10.1145/358916.358995
23. Herlocker, J.L., Konstan, J.A., Riedl, J.T.: An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. Inf. Retr. **5**(4), 287–310 (2002). DOI http://dx.doi.org/10.1023/A:1020443909834
24. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. **22**(1), 5–53 (2004). DOI http://doi.acm.org/10.1145/963770.963772
25. Hijikata, Y., Shimizu, T., Nishida, S.: Discovery-oriented collaborative filtering for improving user satisfaction. In: IUI '09: Proceedings of the 13th international conference on Intelligent user interfaces, pp. 67–76. ACM, New York, NY, USA (2009). DOI http://doi.acm.org/10.1145/1502650.1502663
26. Hu, R., Pu, P.: A comparative user study on rating vs. personality quiz based preference elicitation methods. In: IUI Ó9: Proceedings of the 13th international conference on Intelligent user interfaces, pp. 367–372. ACM, New York, NY, USA (2009). DOI http://doi.acm.org/10.1145/1502650.1502702
27. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of ir techniques. ACM Trans. Inf. Syst. **20**(4), 422–446 (2002). DOI http://doi.acm.org/10.1145/582415.582418
28. Jones, N., Pu, P.: User technology adoption issues in recommender systems. In: Networking and Electronic Conference (2007)
29. Karypis, G.: Evaluation of item-based top-n recommendation algorithms. In: CIKM '01: Proceedings of the tenth international conference on Information and knowledge management, pp. 247–254. ACM, New York, NY, USA (2001). DOI http://doi.acm.org/10.1145/502585.502627
30. Kendall, M.G.: A new measure of rank correlation. Biometrika **30**(1–2), 81–93 (1938)
31. Kendall, M.G.: The treatment of ties in ranking problems. Biometrika **33**(3), 239–251 (1945)
32. Kohavi, R., Longbotham, R., Sommerfield, D., Henne, R.M.: Controlled experiments on the web: survey and practical guide. Data Min. Knowl. Discov. **18**(1), 140–181 (2009). DOI http://dx.doi.org/10.1007/s10618-008-0114-1
33. Konstan, J.A., McNee, S.M., Ziegler, C.N., Torres, R., Kapoor, N., Riedl, J.: Lessons on applying automated recommender systems to information-seeking tasks. In: AAAI (2006)
34. Koychev, I., Schwab, I.: Adaptation to drifting user's interests. In: In Proceedings of ECML2000 Workshop: Machine Learning in New Information Age, pp. 39–46 (2000)
35. Lam, S.K., Frankowski, D., Riedl, J.: Do you trust your recommendations? an exploration of security and privacy issues in recommender systems. In: In Proceedings of the 2006 Interna-

tional Conference on Emerging Trends in Information and Communication Security (ETRICS (2006)
36. Lam, S.K., Riedl, J.: Shilling recommender systems for fun and profit. In: WWW '04: Proceedings of the 13th international conference on World Wide Web, pp. 393–402. ACM, New York, NY, USA (2004). DOI http://doi.acm.org/10.1145/988672.988726
37. Mahmood, T., Ricci, F.: Learning and adaptivity in interactive recommender systems. In: ICEC '07: Proceedings of the ninth international conference on Electronic commerce, pp. 75–84. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1282100.1282114
38. Marlin, B.M., Zemel, R.S., Roweis, S., Slaney, M.: Collaborative filtering and the missing at random assumption. In: Proceedings of the 23rd COnference on Uncertainity in Artificial Intelligence (2007)
39. Massa, P., Bhattacharjee, B.: Using trust in recommender systems: An experimental analysis. In: In Proceedings of iTrust2004 International Conference, pp. 221–235 (2004)
40. McLaughlin, M.R., Herlocker, J.L.: 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, pp. 329–336. ACM, New York, NY, USA (2004). DOI http://doi.acm.org/10.1145/1008992.1009050
41. McNee, S.M., Riedl, J., Konstan, J.A.: Making recommendations better: an analytic model for human-recommender interaction. In: CHI '06: CHI '06 extended abstracts on Human factors in computing systems, pp. 1103–1108. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1125451.1125660
42. McSherry, F., Mironov, I.: Differentially private recommender systems: building privacy into the netflix prize contenders. In: KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 627–636. ACM, New York, NY, USA (2009). DOI http://doi.acm.org/10.1145/1557019.1557090
43. Mobasher, B., Burke, R., Bhaumik, R., Williams, C.: Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. ACM Trans. Internet Technol. **7**(4), 23 (2007). DOI http://doi.acm.org/10.1145/1278366.1278372
44. Murakami, T., Mori, K., Orihara, R.: Metrics for evaluating the serendipity of recommendation lists. New Frontiers in Artificial Intelligence **4914**, 40–46 (2008)
45. O'Mahony, M., Hurley, N., Kushmerick, N., Silvestre, G.: Collaborative recommendation: A robustness analysis. ACM Trans. Internet Technol. **4**(4), 344–377 (2004). DOI http://doi.acm.org/10.1145/1031114.1031116
46. Pfleeger, S.L., Kitchenham, B.A.: Principles of survey research. SIGSOFT Softw. Eng. Notes **26**(6), 16–18 (2001). DOI http://doi.acm.org/10.1145/505532.505535
47. Pu, P., Chen, L.: Trust building with explanation interfaces. In: IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces, pp. 93–100. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1111449.1111475
48. Queiroz, S.: Adaptive preference elicitation for top-k recommendation tasks using gai-networks. In: AIAP'07: Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference, pp. 579–584. ACTA Press, Anaheim, CA, USA (2007)
49. Salzberg, S.L.: On comparing classifiers: Pitfalls toavoid and a recommended approach. Data Min. Knowl. Discov. **1**(3), 317–328 (1997). DOI http://dx.doi.org/10.1023/A:1009752403260
50. Sarwar, B., Karypis, G., Konstan, J., Reidl, J.: Item-based collaborative filtering recommendation algorithms. In: WWW '01: Proceedings of the 10th international conference on World Wide Web, pp. 285–295. ACM, New York, NY, USA (2001). DOI http://doi.acm.org/10.1145/371920.372071
51. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Analysis of recommendation algorithms for e-commerce. In: EC '00: Proceedings of the 2nd ACM conference on Electronic commerce, pp. 158–167. ACM, New York, NY, USA (2000). DOI http://doi.acm.org/10.1145/352871.352887

52. Schein, A.I., Popescul, A., Ungar, L.H., Pennock, D.M.: 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, pp. 253–260. ACM, New York, NY, USA (2002). DOI http://doi.acm.org/10.1145/564376.564421
53. Shani, G., Chickering, D.M., Meek, C.: Mining recommendations from the web. In: RecSys '08: Proceedings of the 2008 ACM Conference on Recommender Systems, pp. 35–42 (2008)
54. Shani, G., Heckerman, D., Brafman, R.I.: An mdp-based recommender system. Journal of Machine Learning Research **6**, 1265–1295 (2005)
55. Smyth, B., McClave, P.: Similarity vs. diversity. In: ICCBR, pp. 347–361 (2001)
56. Spillman, W., Lang, E.: The Law of Diminishing Returns. World Book Company (1924)
57. Swearingen, K., Sinha, R.: Beyond algorithms: An hci perspective on recommender systems. In: ACM SIGIR 2001 Workshop on Recommender Systems (2001)
58. Van Rijsbergen, C.J.: Information Retrieval. Butterworth-Heinemann, Newton, MA, USA (1979). URL `http://portal.acm.org/citation.cfm?id=539927`
59. Voorhees, E.M.: Overview of trec 2002. In: In Proceedings of the 11th Text Retrieval Conference (TREC 2002), NIST Special Publication 500-251, pp. 1–15 (2002)
60. Voorhees, E.M.: 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, pp. 355–370. Springer-Verlag, London, UK (2002)
61. Yao, Y.Y.: Measuring retrieval effectiveness based on user preference of documents. J. Amer. Soc. Inf. Sys **46**(2), 133–145 (1995)
62. Zhang, M., Hurley, N.: Avoiding monotony: improving the diversity of recommendation lists. In: RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems, pp. 123–130. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1454008.1454030
63. Zhang, Y., Callan, J., Minka, T.: Novelty and redundancy detection in adaptive filtering. In: SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 81–88. ACM, New York, NY, USA (2002). DOI http://doi.acm.org/10.1145/564376.564393
64. Ziegler, C.N., McNee, S.M., Konstan, J.A., Lausen, G.: Improving recommendation lists through topic diversification. In: WWW '05: Proceedings of the 14th international conference on World Wide Web, pp. 22–32. ACM, New York, NY, USA (2005). DOI http://doi.acm.org/10.1145/1060745.1060754