Chapter 12 in Recommendation Systems in Software Engineering (c) Springer 2014

Dimensions and Metrics for Evaluating Recommendation Systems

Iman Avazpour, Teerat Pitakrat, Lars Grunske and John Grundy

Abstract recommendation systems support users and developers of various computer and software systems to overcome information overload, perform information discovery tasks and approximate computation, among others. They have recently become popular and have attracted a wide variety of application scenarios from business process modelling to source code manipulation. Due to this wide variety of application domains, different approaches and metrics have been adopted for their evaluation. In this chapter, we review a range of evaluation metrics and measures as well as some approaches used for evaluating recommendation systems. The metrics presented in this chapter are grouped under sixteen different dimensions, e.g., correctness, novelty, coverage. We review these metrics according to the dimensions to which they correspond. A brief overview of approaches to comprehensive evaluation using collections of recommendation system dimensions and associated metrics is presented. We also provide suggestions for key future research and practice directions.

Iman Avazpour
Faculty of ICT, Centre for Computing and Engineering Software and Systems (SUCCESS), Swinburne University of Technology, Hawthorn, Victoria 3122, Australia e-mail: iavazpour@swin.edu.au

Teerat Pitakrat
Institute of Software Technology, Universität Stuttgart, Universitätsstraße 38, D-70569 Stuttgart, Germany e-mail: teerat.pitakrat@informatik.uni-stuttgart.de

Lars Grunske
Institute of Software Technology, Universität Stuttgart, Universitätsstraße 38, D-70569 Stuttgart, Germany e-mail: lars.grunske@informatik.uni-stuttgart.de

John Grundy
Faculty of ICT, Centre for Computing and Engineering Software and Systems (SUCCESS), Swinburne University of Technology, Hawthorn, Victoria 3122, Australia e-mail: jgrundy@swin.edu.au

1 Introduction

Due to the complexity of today's software systems, modern software development environments provide recommendation systems for various tasks. These ease the developers' decisions or warn them about the implications of their decisions. Examples are code completion, refactoring support or enhanced search capabilities during specific maintenance activities. In recent years, research has produced a variety of these recommendation systems and some of them have similar intentions and functionalities [23, 58]. One obvious question is, therefore, how can we assess quality and how can we benchmark different recommendation systems?

In this chapter, we provide a practical guide to the commonly used quantitative evaluation techniques used to compare recommendation systems. As a first step, we have identified a set of dimensions, e.g., the correctness or diversity of the results that may serve as a basis for an evaluation of a recommendation system. The different dimensions will be explained in detail and different metrics are presented to measure and quantify each dimension. Furthermore, we explore interrelationships between dimensions and present a guide showing how to use the dimensions in an individual recommendation system validation.

The rest of the chapter is organized as follows: Section 2 introduces the evaluation dimensions for recommendation systems and presents common metrics for them. Section 3 explores relationships between the different dimensions. Section 4 provides a description of some evaluation approaches and their practical application and implications. Finally, conclusions are drawn in Section 5.

2 Dimensions

The multi-faceted characteristics of recommendation systems lead us to consider multiple dimensions for recommender evaluation. Just one dimension and metric for evaluating the wide variety of recommendation systems and application domains is far too simplistic to obtain a nuanced evaluation of the approach as applied to the domain.

In this chapter, we investigate a variety of dimensions that may be used to play a significant role in evaluating a recommendation system. We list these dimensions below according to our view of their relative evaluative importance, along with the characteristics that each dimension is used to measure. Some of these dimensions describe qualitative characteristics while others are more quantitative. Below we list the key dimensions we describe in detail in this chapter:

- **Correctness** - how close are recommendations to a set of recommendations assumed to be correct?
- **Coverage** - to what extent does the recommendation system cover a set of items or user space?
- **Diversity** - how diverse (dissimilar) are the recommended items in a list?

- **Trustworthiness** - how trustworthy are the recommendations?
- **Recommender confidence** - how confident is the recommendation system in its recommendations?
- **Novelty** - how successful is the recommendation system in recommending items that are new or unknown to users?
- **Serendipity** - to what extent has the system succeeded in providing surprising yet beneficial recommendations?
- **Utility** - what is the value gained from this recommendation for users?
- **Risk** - how much user risk is associated in accepting each recommendation?
- **Robustness** - how tolerant is the recommendation system to bias or false information?
- **Learning rate** - how fast can the system incorporate new information to update its recommendation list?
- **Usability** - how usable is the recommendation system? Will it be easy for users to adopt it in an appropriate way?
- **Scalability** - how scalable is the system with respect to number of users, underlying data size and algorithm performance?
- **Stability** - how consistent are the recommendations over a period of time?
- **Privacy** - are there any risks to user privacy?
- **User preference** - how do users perceive the recommendation system?

We have grouped these dimensions into four broad categories, depending on different aspects of the recommendation system they address. The categories we use are: *Recommendation-centric*, *User-centric*, *System-centric* and *Delivery-centric*. Table 1 summarizes how each of the above dimensions can be grouped inside each category.

Recommendation-centric dimensions primarily assess the recommendations generated by the recommendation system itself: their coverage, correctness, diversity and level of confidence in the produced recommendations. On the other hand, user-centric dimensions allow us to assess the degrees to which the recommendation system under evaluation fulfils its target end-user needs. This includes how trustworthy are the recommendations produced, degree of novelty, whether serendipitous recommendations are a feature, the overall utility of the recommendations from the users' perspective, and risks associated with the recommendations produced, again from the users' perspective. System-centric dimensions in contrast principally provide ways to gauge the recommendation system itself, rather than the recommendations or user perspective. These include assessment of the robustness of the recommendation system, its learning rate given new data, its scalability given data size, its stability under data change, and degree of privacy support in the context of shared recommendation system datasets. Finally, delivery-centric dimensions primarily focus of the recommendation system in the context of use, including its usability (broadly assessed) and support for user configuration and preferences.

The following subsections describe each of these dimensions in detail.

Table 1: Categorization of dimensions

Recommendation-centric	**User-centric**
Correctness	Trustworthiness
Coverage	Novelty
Diversity	Serendipity
Recommender confidence	Utility
	Risk
System-centric	**Delivery-centric**
Robustness	Usability
Learning rate	User preference
Scalability	
Stability	
Privacy	

2.1 Correctness

In order to be of real value, recommendation systems must provide useful results that are close to users' interests, intentions or applications without overwhelming them with unwanted results. A key measure of this is the *correctness* of the set of recommendations produced. Correctness provides a measure of how close the recommendations given to a user are to a set of expected predefined, or assumed *correct*, recommendations. This pre-defined set of correct recommendations is sometimes referred to as the *gold standard*. The correctness of a recommendation may refer to its alignment with a benchmark (e.g., each recommended link is in the predefined set of correct links), or how well it adheres to desired qualities (e.g., increase in developer productivity).

Depending on the type of recommendations the system is generating, different methods can be used for measuring correctness. A recommender might predict how users rate an item, the order (ranking) of most interesting to least interesting items for a user in a list, or which item (or list of items) is of interest to the user. In the following subsections, we describe most frequently used metrics for evaluating recommendation approaches for correctness in each scenario.

2.1.1 Predicting user ratings

If the recommendations produced are intended to predict how users rate items of interest then Root Mean Squared Error (RMSE) or Mean Absolute Error (MAE) metrics are often used (examples are [6, 33, 42, 65, 73]). When calculating RMSE, the difference between actual user ratings and predicted ratings (often called *Residual*) should be determined. If r_{ui} is the actual rating of user u for item i, and $\hat{r}_{ui}$ is the predicted value, $(\hat{r}_{ui} - r_{ui})$ will determine the residual of the two ratings. Depending on whether the recommendation system has over or under estimated the rating, residuals can be positive or negative. Assuming N is the number of all rat-

ings, RMSE can be calculated by squaring the residuals, averaging the squares and taking the square root.

$$\textit{Root Mean Squared Error} = \sqrt{\frac{\Sigma_{(u,i)\in T}(\hat{r}_{ui} - r_{ui})^2}{N}} \tag{1}$$

MAE, on the other hand, measures the average absolute deviation of predicted ratings from user ratings.

$$\textit{Mean Absolute Error} = \frac{\Sigma_{(u,i)\in T} \mid \hat{r}_{ui} - r_{ui} \mid}{N} \tag{2}$$

Where T is the test set of user item pairs (u,i). All individual residuals in MAE are equally weighted while in RMSE large errors get penalized more than small errors. This is because the errors are squared before they are averaged. Therefore, if large errors are undesirable, RMSE is a more suitable metric than MAE. Lower values of both RMSE and MAE indicate greater correctness. RMSE is generally larger than or equal to the MAE. If both metrics are equal, then all errors have the same magnitude.

Both RMSE and MAE can be normalized according to the rating range to represent scaled versions of themselves.

$$\textit{normalized RMSE} = \frac{RMSE}{r_{\max} - r_{\min}} \tag{3}$$

$$\textit{normalized MAE} = \frac{MAE}{r_{\max} - r_{\min}} \tag{4}$$

If the items to be tested represent an unbalanced distribution, RMSE and MAE can be used in averaged form, depending on the evaluation (e.g., per-user or per-item). If the RMSE of each item can be calculated separately, then the average of all calculated RMSEs represents the Average RMSE of the recommendation system.

Example ▷ Consider the problem of ranking Java files returned by a recommendation system for code search. Assume three files are recommended to a user with predicted ratings of 3, 5, 5 in a 1 to 5 scale scoring system while the actual ratings provided by the user are 4, 3, 5 respectively. The above metrics can be calculated as follows.

$$RMSE = \sqrt{\frac{(3-4)^2 + (5-3)^2 + (5-5)^2}{3}} \approx 1.291 \tag{5}$$

$$MAE = \frac{(3-4) + (5-3) + (5-5)}{3} \approx 0.334 \tag{6}$$

$$\textit{normalized RMSE} = \frac{RMSE}{5-1} \approx 0.323 \tag{7}$$

$$normalized\ MAE = \frac{MAE}{5-1} \approx 0.08 \tag{8}$$

◁

2.1.2 Ranking items

Ranking measures are used when an ordered list of recommendations is presented to users according to the their preferences. This order can be as the most important, or 'most relevant', items at the top and the least important, or 'least relevant' items at the bottom. For example, when recommending links between architecture documents and code artifacts in a source code traceability recommendation system, the most closely related links should be shown first. Similarly, when recommending code snippets for reuse from a source code repository in a code reuse recommendation system, the code snippet most appropriate to the current reuse context should be shown first.

When checking for correctness of ranking measures, if a reference ranking (benchmark) is available, the correctness of the ranking can be measured by Normalized Distance-based Performance Measure (NDPM) [78].The value returned by NDPM is between 0 and 1 with any acceptable ranking having a distance of 0. A ranking farthest away from an ideal ranking would have a normalized distance of 1.

NDPM penalizes a contradicting prediction twice as much as when it does not predict an item in the ranking. It also does not penalize the system for ranking one item over another when they have ties. Having a tie in some situations, however, indicates that the value of the items having tie is equal to the user. Therefore, ranking one item higher than the other in a tie will produce inaccurate ranking. In situations where ties between recommended items are to be considered, rank correlation measures, such as Spearmans ρ or Kendall's τ can be used [29, 30].

For some cases, the position of recommended items in the list is important for the application of the recommendation. For example, in a software documentation retrieval environment, since all documentation artifacts are not of equal relevance to their users, highly relevant documents, or document components, should be identified and ranked first for presentation [27]. Therefore, the correctness of an item in the ranking list should be weighted by its position in the ranking. A frequently used metric for measuring ranking correctness, considering item ranking position, is Normalized Discounted Cumulative Gain (NDCG). It is calculated based on measuring Discounted Cumulative Gain (DCG) and then comparing that to the ideal ranking. DCG measures the correctness of a ranked list based on the relevance of items discounted by their position in the list. Higher values of NDCG indicate better ranked lists and therefore better correctness. Various approaches have been introduced to optimize NDCG and ranking measures. Examples of these approaches can be found in Weimer et al. [77] and Le and Smola [40].

2.1.3 Recommending interesting items

If a recommendation system is providing the items that users may like to use, a common approach to evaluate it is to use classification metrics like precision, recall, accuracy and false positive rate. These metrics have been used excessively across different domains [15, 17, 41, 43, 47, 79] and classify produced recommendations into groups indicated by Table 2. Once categories are defined, metrics will be calculated according to the following formulae:

Table 2: Categorisation of all possible recommendations.

	Recommended	Not Recommended	Total
Used	True-Positive (TP)	False-Negative (FN)	Total Used
Not Used	False-Positive (FP)	True-Negative (TN)	Total Not Used
Total	Total Recommended	Total Not Recommended	Total (T)

$$Precision = \frac{TP}{TP+FP} \tag{9}$$

$$Recall\ (True\ Positive\ Rate) = \frac{TP}{TP+FN} \tag{10}$$

$$False\ Positive\ Rate = \frac{FP}{FP+TN} \tag{11}$$

$$Specifity\ (True\ Negative\ Rate) = \frac{TN}{FP+TN} = 1 - False\ Positive\ Rate \tag{12}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{13}$$

When testing for these metrics off-line and on test data, a common assumption is that items that the user has not selected are uninteresting, or useless, to other users. This assumption can often be incorrect [70]. A user might not select an item because they are not aware of such an item. Therefore, there can be a bias in the categories defined by Table 2. Also, there exists an important trade-off between these metrics when measuring correctness. For instance, allowing for a longer list of recommendations improves recall but is likely to reduce precision. Improving precision often worsens recall [62]. F-measure is introduced as a measure of the harmonic mean of precision and recall according to Equation 14.

$$F\text{-}Measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{14}$$

It is also noteworthy to mention the cost associated with identification of False-Positives (FP) and False-Negatives (FN). For example, it could be relatively easier to identify FP for a user. If this is the case, calculating recall would be less costly and hence more preferred than precision. F-measure assumes equal cost for both FP and FN as defined by Equation 14.

Sometimes it is desirable to provide multiple recommendations to users. In this case, these metrics can be altered to provide correctness measured for the number of N items being provided to user. For example, consider a code completion recommender that can recommend hundreds of items while the user is typing program code. Showing one item at a time would be too limited, similarly showing all recommendations would not be useful. If for each recommendation five items are shown to the user, to calculate precision of this code completion recommender for example, precision at 5 can be used.

If using recommendations over a range of recommendation list lengths, plotting the precision to recall (Precision-Recall curve) or true-positive rate to false-positive rate (Receiver Operating Characteristics or ROC) can be used [25]. 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 emphasizes the items that are not preferred but are recommended.

Example ▷ Assuming an API function list contains 100 items in total, and 20 of them are of interest to a certain user in an API reuse recommendation system. Given the user is presented with a list of ten recommended items, with six being of interest and four otherwise, the precision, recall and F-measure metrics can be calculated as follows:

$$TP = 6,\ FP = 4,\ FN = 14,\ TN = 76 \tag{15}$$

$$\textit{Precision} = \frac{6}{6+4} = 0.6 \tag{16}$$

$$\textit{Recall (True Positive Rate)} = \frac{6}{6+14} = 0.3 \tag{17}$$

$$\textit{False Positive Rate} = \frac{4}{4+76} = 0.05 \tag{18}$$

$$\textit{Specifity (True Negative Rate)} = \frac{76}{4+76} = 0.95 \tag{19}$$

$$\textit{Accuracy} = \frac{6+76}{6+76+4+14} = 0.82 \tag{20}$$

$$\textit{F-Measure} = 2 \cdot \frac{0.6 \cdot 0.3}{0.6+0.3} = 0.4 \tag{21}$$

◁

2.2 Coverage

Recommendation systems make recommendations by searching available information spaces. This recommendation is not always possible, for example when new items or users are introduced, or insufficient data is available for particular items or users. *Coverage* refers to the proportion of available information (items, users) that recommendations can be made for.

Consider a code maintenance recommendation system which guides developers on where to look in a large code base to apply modifications (e.g., [59]). If such a recommender is not capable of covering the whole code base at hand, developers might not be able to find the actual artifact that requires alteration. Hence, the information overload problem and complexity of finding faults in the code base will still exist to a greater or lesser degree. Sometimes this is acceptable, such as when alternative techniques, like visualization, can assist users. Sometimes this is unacceptable, for example when the search space is too large for developers or important parts of the code base remain unsearched, thus hindering maintenance effort.

Coverage usually refers to *catalogue coverage* (item-space coverage) or *prediction coverage* (user-space coverage) [25]. Catalogue coverage is the proportion of available items that the recommendation system recommends to users. Prediction coverage, on the other hand, refers to the proportion of users or user interactions that the recommendation system is able to generate predictions for.

A straightforward way to measure catalogue coverage is by calculating the proportion of items able to be recommended in a single recommendation session where multiple recommender algorithms would be executed a number of times. Therefore, if the set of items recommended to a user over a particular recommendation session is S_r and S_a is the set of all available items, catalogue coverage can be calculated by the following formula:

$$Catalogue\ Coverage = \frac{|\ S_r\ |}{|\ S_a\ |} \tag{22}$$

The items available for recommendation may not all be of interest to a user. Consider a recommendation system that finds relevant expertise to perform a collaborative software engineering task (e.g., [50]). In such a system, if users are looking for expertise in file processing for a Java-based project, recommending an expert in Prolog will not be useful and should be filtered out. Ge et al. propose weighted catalogue coverage for balancing the decrease in recommender coverage by usefulness for users [19]. As a result Equation 22 will be updated to:

$$Weighted\ Catalogue\ Coverage = \frac{|\ S_r \cap S_s\ |}{|\ S_s\ |} \tag{23}$$

where S_s is the set of items that are considered useful to users.

Similar to catalogue coverage, prediction coverage can be calculated by measuring the proportion of users that prediction can be made for (S_p) to a set of available users (S_u).

$$Prediction\ Coverage = \frac{|\ S_p\ |}{|\ S_u\ |} \tag{24}$$

Accordingly, by considering the usefulness of recommended items for the user as a function *f(x)* we have:

$$Weighted\ Prediction\ Coverage = \frac{\Sigma_{i \in S_p} f(i)}{\Sigma_{j \in S_u} f(j)} \tag{25}$$

Ge et al. suggest using *correctness* and *novelty* metrics to calculate usefulness function *f(x)* in Equation 25 and set of useful items S_s in Equation 23 [19].

Situations where a new item is added to the system and sufficient information (like ratings by other users for that item) does not exist is referred to as *Cold start* problem. Cold start can also refer to situations where new users have joined the system and their preferences are not yet known. For example, consider a recommendation system that recommends solutions to fixing a bug similar to DebugAdvisor [5]. In such a recommender the developer submits a query describing the defect. The system then searches for bug descriptions, functions or people that can help the developer fix the bug. If the bug, or a similar bug, has not been previously reported, there is no guarantee that the returned results will help resolve the situation. Similarly, if the system has been newly implemented in a development environment with few bug reports or code repositories, the recommendation would not be very helpful.

Cold start is seen more often in collaborative filtering recommenders as they rely heavily on input from users. Therefore, these recommenders can be used in conjunction with other non-collaborative techniques. Such a hybrid mechanism was proposed by Schein et al. [66]. They used two variations of ROC curve to evaluate their method, namely Global ROC (GROC) and Customer ROC (CROC). GROC was used to measure performance when the recommender is allowed to recommend more often to some users than others. CROC was used to measure performance when the system was constrained to recommend the same number of items to each user [66].

2.3 Diversity

In some cases, having similar items in a recommendation list does not add value from the users' perspectives. The recommendations will seem redundant and it takes longer for users to explore the item space. For example, in an API recommendation system, showing two APIs with the same non-functional characteristics may not be useful unless it helps users gain confidence in the recommendation system. Showing two APIs with diverse performance, memory overheads and providers, for example, could be more desirable for the programmer.

A recommendation list should display some degree of diversity in the presented items. A recent case study on recommending documents to users showed that users

prefer a system providing document diversity. This allows them to get a more complete map of the information [11].

Diversity could be also considered to be the opposite of similarity. If items presented to users are too similar, they do not present diverse items and so may not be of interest. Thus, Smyth and McClave defined diversity in a set of items, $c_1..c_n$, as the average dissimilarity between all pairs of items in the itemset [72]. They introduce the following formula for measuring diversity:

$$Diversity\ (c_1..c_n) = \frac{\Sigma_{i=1..n}\Sigma_{j=i..n}(1 - Similarity(c_i, c_j))}{\frac{n}{2} * (n-1)} \tag{26}$$

Where *similarity* is calculated by the weighted-sum metric below for item c and target query t:

$$Similarity\ (t\ ,\ c) = \frac{\Sigma_{i=1..n}\omega_i * sim(t_i, c_i)}{\Sigma_{i=1..n}\omega_i} \tag{27}$$

Sim can be a similarity heuristic based on sum, average, minimum or maximum distance between item pairs, and ω is the associated weight. Higher values in Equation 26 indicate more diversity and consecutively less similarity. In a fixed size recommendation list, improving diversity results in sacrificing similarity. Therefore, a strategy that optimizes this similarity-diversity trade-off is often beneficial. Thus, a quality metric was introduced to combine both diversity and similarity [72].

$$Quality\ (t,c,R) = Similarity(t,c) * RelDiversity(c,R) \tag{28}$$

This basically specifies that the quality of item c is proportional to its similarity with the current target query t, and to the diversity of c relative to those items so far selected $R = \{r_1 \ldots r_m\}$. As a result a variation of Equation 26 can be provided as *Relative Diversity*.

$$RelDiversity\ (t,R) = \begin{cases} 0 & , if\ R = \{\} \\ \frac{\Sigma_{i=1..m}(1 - Similarity(c,r_i))}{m} & , otherwise \end{cases} \tag{29}$$

To measure diversity in a recommendation list, an alternative approach is to compute the distance of each item from the rest of the list and average the result to obtain a diversity score. For such an average, however, a random recommender may also produce diverse recommendations. Therefore, this needs to be accompanied by some measure of precision. Plotting precision-diversity curves helps in selecting the algorithm with the dominating curve [70]. Having correctness metrics combined with diversity has an added advantage as correctness metrics do not take into account the entire recommendation list. Instead, they consider the correctness of individual items. For instance, the intra-list similarity metric can help to improve the process of topic diversification for recommendation lists [80]. In this way, the returned lists can be checked for intra-list similarity and altered to either increase or decrease the diversity of items on that list as desired or required. Increasing diversity this way has been shown to perform worse than unchanged lists, according to correctness measures, but users preferred the altered lists [44].

Diversity of rating predictions can be measured by well-known diversity measures being used in ensemble learning [36]. These approaches try to increase diversity for returned classification of individual learning algorithms in order to improve the overall performance of ensemble learning. For example, Q-Statistics can be used to find diversity between two recommender algorithms. Q-statistics is based on a confusion matrix concept. It confronts two classifiers, based on binary evaluation strategy, of correctly classified versus incorrectly classified. The confusion matrix displays the overlap of those itemsets. Then using Q-statistic measures, diversity of the two recommender algorithms can be measured. Kille and Albayrak used this approach and introduced a *difficulty* measure which helps with personalizing recommendations per user [31]. They measured a user's difficulty by means of the diversity of rating predictions (RMSE) and item rankings (NDCG), and used diversity metrics by pair wise Q-Statistics to fit the item ranking scenario [31].

Lathia et al. introduced a measure of diversity for recommendations in two lists of varying lengths [39]. In their approach, given two sets L_1 and L_2, the items of L_2 that are not in L_1 are first determined as their set theoretic difference. Then, the diversity between the two lists (at depth N) is defined as the size of their set theoretic difference over N. This way, diversity returns 0 if the two lists are the same, and 1 if the two lists are completely different at depth N.

2.4 Trustworthiness

A recommendation system is expected to provide trustworthy suggestions to its users. It has been shown that perceived usefulness correlated most highly with *good* and *useful* recommendations [74]. If the system is continuously producing incorrect recommendations, users' trust in the recommender will be lost. Lack of trustworthiness will encourage users to ignore recommendations and so decrease the usefulness of the recommendation system. For example, in an IDE being used for a refactoring scenario, a wrong suggestion made by the refactoring task recommender may adversely impact large amounts of application code. If users of such a refactoring recommendation system use a faulty recommendation and experience the consequences, they will be less likely to choose among its recommendations again.

Some users will not build trust in the recommendations unless they see a well-known item, or an item they were already aware of, being recommended [74]. Also, explanations regarding how the system comes up with its recommendations can encourage users to use them and build trust [71, 76].

A common approach to measure trust is asking users whether the recommendations are reasonable in a user study [7, 14, 24]. Depending on the usage scenario of the recommendation system, it might be possible to check how frequently users use recommendations to gain understanding of their trust [52]. For example, in a code reuse recommender, how often the user selects and applies one of the recommended code snippets. Or similarly, how often do users select recommendations of a code completion recommender.

2.5 Recommender confidence

Recommender confidence is the certainty the system has in its own recommendations or predictions. In online scenarios, it is possible to calculate recommender confidence by observing environmental variables. For example, a refactoring recommendation system can build confidence scores by observing how frequently users use and apply suggested refactoring recommendations to their application.

Some prediction models can be used in calculating confidence scores. For example, Bell et al. used a neighborhood-aware similarity model that considers similarities between items and users for generating recommendations [6]. In their model, a recommendation that maximizes the similarity between the item being recommended and similar items, and the user this recommendation is to be represented to and similar users, defines the most suitable recommendation. They showed how such a metric can help identify most suitable recommendations, according to RMSE of the predicted rating and the user's true rating [6].

Cheetham and Price provided an approach for calculating confidence in Case-Based Reasoning (CBR) systems [13]. They proposed to identify multiple indicators such as 'sum of similarities for retrieved cases with best solution' or 'similarity of the single most similar case with best solution'. Once possible indicators are defined, their effect on CBR process was determined using 'leave-one-out' testing. Finally, they used Quinlan's C4.5 algorithm on the 'leave-one-out' test results to identify indicators that are best at determining confidence [13, 55].

Recommender confidence scores can be used in the form of confidence intervals (e.g. in [60]) or by the probability that the predicted value is true [70]. Also, they have been used in hybrid recommendation systems for switching between recommender algorithms [8].

2.6 Novelty

A *novel* recommender recommends items that the users did not know about. *Novelty* is very much related to the emotional response of users to a recommendation, as a result, it is a difficult dimension to measure [44].

A possible approach for building a novel recommender is to remove items that the user has already rated or used before in a recommendation list. If this information is available, novelty of the recommender can be measured easily by comparing recommendations against already used or rated recommendations. This requires keeping user profiles so that it is possible to know which user chose and rated which items. User profiles can then be used to calculate the set of familiar items. For example, CodeBroker is a development environment that promotes reuse by enabling software developers to reuse available components [79]. It integrates a *user model* for capturing methods that the developer already knows and thus does not need to be recommended again.

An alternative approach for measuring novelty is to count the number of popular items that have been recommended [70]. This metric is based on the assumption that highly rated and popular items are likely to be known to users and therefore not novel [48]. A good measure for novelty might be to look more generally at how well a recommendation system made the user aware of previously unknown items that subsequently turn out to be useful in context [25].

2.7 Serendipity

Serendipity by definition is 'the occurrence and development of events by chance in a happy or beneficial way' [69]. In the context of recommendation systems this has been referred to as an unexpected and fortuitous recommendation [44]. Serendipity and novelty are different considering the fact that there is an element of correctness present in serendipity which prevents random recommenders from being serendipitous. Novel unexpected items may, or may not, turn out to be serendipitous. While a random recommender may be novel, if a surprising recommendation does not have any utility to the user it will not be classified as serendipitous, but rather as erroneous and distracting. Therefore it is required that correctness and serendipity be balanced and considered together [70].

Like novelty, to have a serendipitous recommender, similar recommendations should be avoided since their expected appearance in the list will generally not benefit the user [44]. Therefore, user profiles or an automatic or manual labelling of pairs of similar items can help filter out similar items. The definition of this similarity, however, should be dependent on the context in which the recommender is being used. For example, an API recommender presenting completely unusable APIs in the current code context is highly unlikely to promote serendipitous reuse. A document recommender, showing unlikely but still possibly related artifacts in a traceability recommender, may very well present with serendipitously useful artifacts.

Ratability is a feature defined in accordance to serendipity and mostly regarded in machine learning approaches. Given the system has some understanding of the user profile, ratability of a recommended item to a user is the probability that the item will be the next item the user will consume [44]. It is assumed that items with higher ratability are the items that the user has not consumed yet but is likely to use in future, or the items the user has consumed but has not been added to the user profile [45]. In other words, ratability defines the *obviousness* of a 'user rating an item'. Since machine-learning approaches calculate the probability of the item being chosen next, if the recommendation system is using a leave-one-out approach to train the learning procedure, it is possible to calculate the ratability based on that probability.

2.8 Utility

Utility is the value that the system or user gains from a recommendation. For example, Parseweb helps developers find sequences of method calls on objects of a specific type. This helps to match that object with a specific method sequence [75]. In that context, the evaluation can be based on the amount of time saved for finding such a method sequence using recommendations. Therefore, the value of a correct recommendation is based on the utility of that item. A possible evaluation in this context is to consider utility from a cost/benefit ratio analysis [25].

It is noteworthy that precision cannot measure the true usefulness of a recommendation. For example, recommending an already well-known and used API call, document link, code snippet, data map or algorithm will increase precision but has very low utility [48] since such an item will probably already be known to the user. On the other hand, for memory intensive applications, it is sometimes beneficial to recommend well-known items. Thus, it is fair to align the recommender evaluation framework with utility measures in real world applications rather than perhaps over-alignment for correctness.

Depending on the application domain of the recommendation system, the utility of a recommendation can be specified by the user (e.g., in user-defined ratings) or computed by the application itself (e.g., profit-based utility function) [1]. The utility might be calculated by observing subsequent actions of the user, for example, interacting with the recommendation or using recommended items.

For some applications, the position of a recommendation in a list is a deciding factor. For example, Rascal uses a recommender agent to track usage histories of a group of developers and recommends components that are expected to be needed by individual developers [43]. The components that are believed to be most useful to current developers will appear first in the recommendation list. If we assume that there is a higher chance for developers to choose a recommendation among top recommended items rather than exploring the whole list, the utility of each recommendation is then the utility of the recommended item in relation to its position in the list of recommendations [70].

2.9 Risk

Depending on where the recommendation system is being used and what its application domain is, the recommendations can be associated with various potential risks. For example, recommending a list of movies to watch is usually less risky than recommending refactoring solutions in complex coding situations (unless the movies might include inappropriate material for some audiences). Therefore, high-risk recommendation systems must obey a set of constraints on a valid solution. This is because false-positive recommendations are less tolerable and users must be more convinced to use a recommendation [9].

Consequently users may approach risk differently. For example, different users might be prepared to tolerate different levels of Risk. One user might prefer using a component which is no longer maintained but has all required features. Another user might prefer a component that has less features but is under heavy development. In such cases, a standard way to evaluate risk is to consider utility variance in conjunction with the measures of utility and parameterise the degree of risk users are seeking in the evaluation [70].

Another aspect of risk involves privacy. If the system is working according to user profiles, collecting information from users to create that profile introduces the risk of breaching users' privacy [56]. Therefore, it should be ensured that users are aware and willing to take that risk. For example, when recommending developers based on expertise for outsourcing tasks, many other factors will also need to be considered. Privacy will be more discussed in Section 2.15.

2.10 Robustness

Robustness is the ability of a recommendation system to tolerate false information intentionally provided by malicious users or, more commonly, to tolerate mistaken information accidentally provided by users. Mistakes made by users may include asking recommender to analyze documents in incorrect formats, mistakenly rating items, making mistakes in the user profile specification and using the recommender in the wrong context or for the wrong tasks.

In order to evaluate the robustness of a system against attacks, O'Mahony et al. compared prediction ratings before and after false information is provided and analyzed the prediction shift that reflects how the prediction changed afterwards [53]. The prediction shift of item i (Δ_i) and its average ($\overline{\Delta}$) can be defined as

$$\Delta_i = \sum_{u \in U} \frac{\hat{\hat{r}}_{ui} - \hat{r}_{ui}}{|U|} \tag{30}$$

$$\overline{\Delta} = \sum_{i \in I} \frac{\Delta_i}{|I|} \tag{31}$$

where $\hat{r}$ and $\hat{\hat{r}}$ are the predicted ratings before and after false information, respectively, U is a set of users, and I is a set of items.

A large shift, however, may not always affect performance of the system if the false information does not alter the items recommended to users. This situation may occur if actual rating of particular items are ranked so low that the mistakes still cannot push them to the top of recommended items. Many studies, e.g., [49, 63, 68], have also discussed and employed other metrics, including *average hit ratio* and *average rank*, to evaluate robustness. Average hit ratio measures how effective the misleading information is to push items into a recommended list while average rank measures the drop of item ratings outside the recommended list. Hit ratio and rank

for item i, and their averages can be defined using following equations.

$$HitRatio_i = \sum_{u \in U} \frac{H_{ui}}{|U|} \tag{32}$$

$$Rank_i = \sum_{u \in U} \frac{Rank_{ui}}{|U|} \tag{33}$$

$$\overline{HitRatio} = \sum_{i \in I} \frac{HitRatio_i}{|I|} \tag{34}$$

$$\overline{Rank} = \sum_{i \in I} \frac{Rank_i}{|I|} \tag{35}$$

where H_{ui} is 1 if item i appears in the list of recommended items of user u and 0 otherwise. $Rank_{ui}$ is the position of item i in the unrecommended list of user u sorted in a descending order.

2.11 Learning rate

Learning rate is the speed at which a recommendation system learns new information or trends and updates the recommended item list accordingly. A system with high learning rate will be able to adapt to new user preferences or interests of existing users to provide useful recommendations within a short period of learning time. For example, an API recommendation system may have a high learning rate if every time a user rates a recommended item the ranking index and calculations are immediately updated. In comparison, a code recommendation system may have a low learning rate if the indexing of the code repository can only be undertaken sporadically due to high overheads.

Although a fast learning rate can cope with quick shifts in trends, it may also give up some prediction correctness since the new trend that the system recommends might not perfectly match a user's interests. A slow learning rate can also affect the system utility if it fails to catch up with trends and cannot provide a new set of useful recommendations.

The evaluation of learning rate can be done by measuring (1) the time that takes the system to regain its prediction correctness when user interests drift, the time to reach a certain level of correctness for new users or (2) the prediction correctness that the system can achieve within a limited learning time. Koychev and Schwab measured and plotted the prediction correctness of a recommendation system over time and assessed how fast their algorithm adapted to changes [34]. To evaluate the learning rate for new users,Rashid et al. evaluated different algorithms that learn user preferences during the sign-up process [57]. Each algorithm presents users with a list of initial items to be rated and learns from the given ratings. After the sign-up

process and the learning phase is completed, predictions for other items are made and the accuracies of the algorithms are measured and compared.

2.12 Usability

In order for recommendation systems to be effective, their target end users must be able to use them in appropriate ways. They must also adhere to the general principles of usability. They must be effective, efficient and provide some degree of satisfaction for their target end users [51].

Recommendation systems typically manifest in some way via a user interface. The contents presented by this user interface plays an important role in acceptance of the recommendation [54]. This user interface may simply be a suggestion to the user in-situ in the containing application. More commonly, a list of recommendations, often ranked, is provided to the user on demand. Additionally, many recommendation systems require configuration parameters, user preferences and some form of user profile to be specified. All of these interfaces greatly impact on the usability of the recommendation system as a whole. For example, presenting the user with an overwhelmingly large list of unranked or improperly ordered items is ineffective and inefficient. Presenting the user with very complicated or hard to understand information is also ineffective and impacts satisfaction. Satisfaction and efficiency are reduced if users are not allowed to interact with recommended items, for example go to target document adversely, or if the system is slow in producing a set of recommendations. These factors of recommendation systems are generally evaluated through user studies [54, 71, 74].

2.13 Scalability

One of the most important goals of a recommendation system is to provide on-line recommendations for users to navigate through a collection of items. When the system scales up to the point where there are thousands of components, bug reports, or software experts to be recommended, the system must be able to process and make each recommendation within a reasonable amount of time. If the system cannot handle a large amount of data, other dimensions will have to be compromised. For instance, the algorithm might generate recommendations based on only a subset of items instead of using the whole database. This reduces the processing time but consequently also reduces its coverage and correctness. Many examples exist of recommendation systems that work well on small data sets but struggle with large item sets or large numbers of users. These include most early API and code recommenders, many existing code or database search and rank result recommenders and complex design or code refactoring recommenders.

The scalability problem can be divided into two parts: (1) the training time of the recommendation algorithm and (2) the performance of the system or throughput when working with a large item database. The time that is required to train the algorithm can be evaluated by training different algorithms with the same dataset or by training them until they reach the same level of prediction correctness [20, 28]. The performance of the system can be evaluated in terms of throughput — the number of recommendations that the system can generate per second [16, 22, 64]. Performance (in terms of number of recommendations) can also adversely impact the usability of the recommendation system as response time may become too slow to be effective for its users.

2.14 Stability

Stability refers to the prediction consistency of the recommendation system over a period of time, assuming that new ratings or items added during that period are in agreement with the ones already existing in the system. A stable recommender can help increase user trust as users will be presented with consistent predictions. The prediction that changes and fluctuates frequently can cause confusion to the users and, consequently, distrust in the system.

Stability can be measured by comparing a prediction at a certain point in time with a point when new ratings are added. Adomavicius and Zhang carried out a stability evaluation by training the recommendation algorithm with the existing ratings and making a first prediction [2, 3]. After new ratings during the next period are added, the algorithm is retrained with this new dataset. It then makes a second prediction. Similar to robustness, Equations 30 and 31 can be used to calculate the prediction shift after a new set of ratings are added.

2.15 Privacy

Recommendation systems often record and log user interaction into historical user pro-files. This helps personalize recommendations and improve understanding of user needs. Recording this information introduces a potential threat to users' privacy. Therefore, some users might request their personal data to be kept private and not disclosed. To secure data, some approaches have proposed cryptographic solutions, or removing the single trusted party having access to the collected data (see, e.g., [4, 12]). Despite these efforts, it has been demonstrated that it is possible to infer user histories by passively observing a recommender's recommendations [10].

Indeed, introducing a metric for measuring privacy is a difficult task. A feasible approach is to measure how much information has been disclosed to third parties as used in web browsing scenarios [35]. The *Differential Privacy* measure is a privacy definition based on similar principles [18]. It indicates that the output of a computa-

tion should not let inference know about any record's presence in, or absence from, the computation's input. This is calculated as follows.

Consider a randomized function K with its input as the data set and its output as the released information. Also consider data sets D_1 and D_2 differing on at most one element. Then function K satisfies differential privacy (ε) for all $S \subseteq Range(K)$ if:

$$Pr[K(D1) \in S] \leq \exp(\varepsilon) \times Pr[K(D2) \in S] \tag{36}$$

In the context of recommendation systems, however, privacy should be measured in conjunction with correctness since keeping information from the system, or third party recommendation system, has a direct effect on correctness of the recommendation system. This difference can be shown by plotting correctness against the options available for preserving privacy. For example, McSherry and Mironov demonstrated their privacy preserving application by plotting RMSE versus Differential Privacy [46].

There are still open questions and areas to explore regarding how privacy can affect recommendation systems and how to measure its effects [38]. Consider multi-user and multi-organizational situations such as open source applications where API, bug triage, code reuse, document/code trace and expertise recommenders may share repositories. Capturing user recommender interactions may enhance recommender performance for all of these domains, however, exposing the recommended items, user ratings and recommender queries all have the potential to seriously compromise developer and organizational privacy.

2.16 User Preferences

We have presented a number of measures to evaluate the performance of recommendation systems. The bottom line of any recommendation system evaluation is the perception of the users of that system. Therefore, depending on application domain, an effective evaluation scenario could be to provide recommendations regarding the selection of algorithms and ask users which one they prefer. Moreover, it has been shown that some metrics (although useful for comparison) are not good measures of user preference. For example, what MAE measures and what really matters to users contrast since, due to the decision supportive nature of recommendation systems, the exact predicted value is of far less importance to a user than the fact that an item is recommended [38]. A number of recent document/code link recovery recommenders incorporate concurrently used algorithms that generate multiple sets of recommendations that can be presented either separately or combined. Many systems allow users to configure the presentation of results, ranking scales, filters on results, number of results provided and relative weightings of multiple item features.

It should be taken into consideration, however, that user preferences are not binary values. Users might prefer one algorithm to another [70]. Therefore, if testing user preferences regarding a group of algorithms, a non-binary measure should be

used before the scores are calibrated [32]. Also, new users should be separated in the evaluation from more experienced users. New users may need to establish trust and rapport with a recommender before taking advantage of the recommendations it offers. Therefore, they might benefit from an algorithm which generates highly ratable items [44].

3 Relation between dimensions

To have an effective evaluation, relationships between dimensions should also be considered. These relationships describe whether changing a dimension affects other dimensions. We have captured these relationships in Table 3, depicting the relationships between dimensions for overall performance of the recommendation system. Each cell in this table depicts relationships between one dimension when compared to another. If changes to a dimension are in accordance with another dimension, i.e., if improving that dimension improves the other, it has been shown by $\odot$. If a dimension tends to adversely impact another, it is shown as a $\times$. Dimensions that tend to be independent are shown with blank cells. Below we summarize some of these recommender dimension interrelations that are not already mentioned in previous sections.

Table 3: Relation between metrics. $\odot$ indicates direct relation, while $\times$ indicates adverse relation.

Metric	User preference	Correctness	Coverage	Confidence	Trustworthiness	Novelty	Serendipity	Diversity	Utility	Risk	Robustness	Privacy	Usability	Stability	Scalability	Learning rate
User preference	-	$\times$	$\odot$		$\odot$				$\odot$				$\odot$	$\odot$		
Correctness	$\times$	-	$\odot$		$\odot$	$\times$	$\times$	$\times$	$\odot$	$\odot$	$\times$	$\times$			$\odot$	$\odot$
Coverage	$\odot$	$\odot$	-			$\odot$	$\odot$	$\odot$	$\odot$			$\times$			$\odot$	
Confidence				-												
Trustworthiness	$\odot$	$\odot$			-					$\odot$	$\odot$		$\odot$	$\odot$		
Novelty		$\times$	$\odot$			-	$\odot$	$\odot$		$\times$			$\odot$			
Serendipity		$\times$	$\odot$			$\odot$	-	$\odot$		$\times$			$\odot$			
Diversity		$\times$	$\odot$			$\odot$	$\odot$	-		$\times$						
Utility	$\odot$	$\odot$	$\odot$						-	$\odot$						$\odot$
Risk		$\odot$			$\odot$	$\times$	$\times$	$\times$	$\odot$	-	$\odot$	$\odot$				
Robustness		$\times$			$\odot$					$\odot$	-			$\odot$		$\times$
Privacy		$\times$	$\times$							$\odot$		-				
Usability	$\odot$				$\odot$	$\odot$	$\odot$						-			
Stability	$\odot$				$\odot$						$\odot$			-		
Scalability		$\odot$	$\odot$												-	
Learning rate		$\odot$							$\odot$		$\times$					-

Coverage can directly affect correctness since the more data available for generating recommendations, the more meaningful the recommendations are. Hence correctness increases [21]. Coverage is also closely related to serendipity. Not every increase in coverage increases serendipity, however, an increase in serendipity will lead to higher catalogue coverage. On the other hand, more correctness dictates more constraints and therefore decreases serendipity [19]. The same is true for risk, i.e., if recommendations are being used in high risk environments, more constraints should be considered. This decreases serendipity, novelty and diversity but increases correctness, trust and utility.

High usability increases the amount of trust users have in the recommended system, especially when recommendations are transparent and accompanied by explanations. Improving privacy forces recommendation systems to hide some user data and hence affects the correctness of the recommendation.

Novel recommendations are generally recommendations that are not known to the user. It is not always a requirement for a novel recommendation to be accurate. Improving novelty by introducing randomness may decrease correctness. Also, improving novelty by omitting well-known items will affect correctness. Therefore, increasing novelty may decrease correctness. The same is true for diversity.

Scalability and learning rate directly affect correctness since improving them allows faster adaptation of new items and users, thus resulting in better correctness. Improving Scalability at the same time also improves coverage.

Improving robustness prevents mistaken information from affecting recommendations and hence improves user trust [37]. It will, however, result in true recommendations being adopted more slowly, therefore reducing short time correctness.

It is noteworthy that from the metrics presented in this table, risk could have been categorized separately. Regardless of how the recommendation system performs, risks involved with the application are the same, i.e., although having a better performing recommendation system helps minimising the risk associated with 'selecting a recommendation', it does not change the fact that risks for that particular application exist in general.

The true relationships between metrics is beyond a two dimensional table. For example, improving coverage directly improves correctness and increasing novelty might decrease the effect of long tail and hence improve coverage. Thus, improving novelty can be considered to indirectly improve correctness that is in contrast with our table, since it has been mentioned that improving novelty may decrease correctness. Therefore, a better framework or standard for understanding these relationships is a necessity and should be considered for future research.

4 Evaluation approaches and frameworks

Table 4 summarizes the set of evaluation metrics and techniques dimensions described earlier according to their corresponding dimension and type(s). Some of the dimensions are qualitative assessments while others are quantitative.

Table 4: Summary of metrics.

Dimension	Metric/Technique	Type(s)
Correctness	Ratings: Root Mean Square Error (RMSE), Normalized RMSE (NRMSE), Mean Absolute Error (MAE), Normalized MAE (NMAE) Ranking: Normalized Distance-based Performance Measure (NDPM), Spearman correlation, Kendall correlation, Normalized Discounted Cumulative Gain (NDCG) Classification: Precision, Recall, False Positive Rate, Specifity, F-Measure, Reciver Operating Characteristics (ROC)	Quantitative
Coverage	Catalogue Coverage, Weighted Catalogue Coverage, Prediction Coverage, Weighted Prediction Coverage	Quantitative
Diversity	Diversity Measure, Relative Diversity, Precision-Diversity Curve, Q-Statistics, Set theoretic difference of recommendation lists	Quantitative
Trustworthiness	User studies	Qualitative
Confidence	Neighborhood-aware similarity model, Similarity indicators	Qualitative/ Quantitative
Novelty	Comparing recommendation list and user profiles, Counting popular items	Qualitative/ Quantitative
Serendipity	Comparing recommendation list and user profiles, ratability	Qualitative/ Quantitative
Utility	Profit based utility function, study user intention, user study	Qualitative/ Quantitative
Risk	Depending on application and user preference	Qualitative
Robustness	Prediction shift, average hit ratio, average rank	Quantitative
Learning rate	Correctness over time	Quantitative
Usability	User studies (survey, observation, monitoring)	Qualitative/ Quantitative
Scalability	Training time, recommendation throughput	Quantitative
Stability	Prediction shift	Quantitative
Privacy	Differential privacy, RMSE vs. Differential privacy curve	Qualitative/ Quantitative
User preference	User studies	Qualitative/ Quantitative

The most basic evaluation of a recommendation system is to use just one or two metrics covering one or two dimensions. For example, one may choose to evaluate and compare a recommender using correctness and diversity dimensions. When possible, the selected dimensions can be plotted to allow better analysis. The selection of dimensions can be according to a particular recommender application. As mentioned in Section 3, however, there is always a trade-off present between the dimensions of a recommendation system that should be considered when evaluating the effectiveness of recommendation systems. Also, the multi-faceted characteristics of these systems, and unavailability of a standard framework for evaluation, and in many case suitable performance benchmarks, has directly affected the evaluation of

different systems by dimensions. In addition, many metrics require significant time and effort to properly design experiments, capture results and analyse results. Availability of end users, suitable datasets, suitable reference benchmarks, and multiple implementations of different approaches are all often challenging issues.

However, some new approaches are beginning to emerge to help developers and users decide between different recommender algorithms and systems. An example of this is an approach that helps users define which metrics can be used for evaluation of the recommendation system at hand [67]. It proposes to consider evaluation goals to ensure the selection of an appropriate metric. An analysis of a collection of correctness metrics is provided as evidence regarding how different goals can affect the outcome of the evaluation [67].

Hernández del Olmo and Gaudioso propose an objective-based framework for the standardization of recommendation system evaluations [26]. Their framework is based on the concept that a recommendation system is composed of interactive (*guide*) and non-interactive subsystems (*filter*). The *guide* decides when and how each recommendation is to be shown to users. The *filter* selects interesting items to recommend. Accordingly, a performance metric P has been introduced as the quantification of the final performance of a recommendation system over a set of sessions. P is defined as the number of selected relevant recommendations that have been followed by the user over a recommendation session [26].

A more recent approach introduced a multi-faceted model for recommender evaluation that proposes evaluation along three axis - users, technical constraints and business models [61]. This approach considers users, technical and business aspects together and evaluates the recommender accordingly. However, considerable further work is needed to enable detailed evaluation of recommendation system against many of the potential metrics itemised in Table 4.

5 Conclusion

In this chapter, we have presented and explained a range of common metrics used for the evaluation of recommendation systems in software engineering. Based on a review of current literature, we derived a set of dimensions that are used to evaluate individual recommendation systems or in comparing it against the current state of the art. For the dimensions, we have provided a description as well as a set of commonly used metrics and explored relationships between the dimensions.

We hope that our classification and description of this range of available evaluation metrics will hep other researchers to develop better recommendation systems. We also hope that our taxonomy will be used to improve the validation of newly developed recommendation systems and clearly show in specific ways how a new recommendation system is better than the current state of the art. Finally, the content of this chapter can be used by practitioners in understanding the evaluation criteria for recommendation systems. This can thus improve their decisions when selecting a specific recommendation system for a software development project.

References

1. Gediminas Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734 – 749, june 2005. ISSN 1041-4347.
2. Gediminas Adomavicius and Jingjing Zhang. On the stability of recommendation algorithms. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 47–54, New York, NY, USA, 2010. ACM.
3. Gediminas Adomavicius and Jingjing Zhang. Iterative smoothing technique for improving stability of recommender systems. In *Workshop on Recommendation Utility Evaluation: Beyond RMSE (RUE 2011)*, page 3, 2012.
4. Esma Aïmeur, Gilles Brassard, José Manuel Fernandez, and Flavien Serge Mani Onana. Alambic: a privacy-preserving recommender system for electronic commerce. *Int. J. Inf. Sec*, 7(5):307–334, 2008.
5. B. Ashok, Joseph Joy, Hongkang Liang, Sriram K. Rajamani, Gopal Srinivasa, and Vipindeep Vangala. Debugadvisor: a recommender system for debugging. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC/FSE '09, pages 373–382, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-001-2.
6. Robert Bell, Yehuda Koren, and Chris Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 95–104, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-609-7.
7. Philip Bonhard, Clare Harries, John McCarthy, and M. Angela Sasse. Accounting for taste: using profile similarity to improve recommender systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 1057–1066, New York, NY, USA, 2006. ACM. ISBN 1-59593-372-7.
8. Robin Burke. Hybrid web recommender systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 377–408. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-72078-2.
9. Robin Burke and Maryam Ramezani. Matching recommendation technologies and domains. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 367–386. Springer US, 2011. ISBN 978-0-387-85819-7.
10. J.A. Calandrino, A. Kilzer, A. Narayanan, E.W. Felten, and V. Shmatikov. "you might also like:" privacy risks of collaborative filtering. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 231 –246, may 2011.
11. L. Candillier, M. Chevalier, D. Dudognon, and J. Mothe. Diversity in recommender systems: bridging the gap between users and systems. In *CENTRIC 2011, The Fourth International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services*, pages 48–53, 2011.
12. J. Canny. Collaborative filtering with privacy. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 45 – 57, 2002.
13. William Cheetham and Joseph Price. Measures of solution accuracy in case-based reasoning systems. In Peter Funk and PedroA. Gonzlez Calero, editors, *Advances in Case-Based Reasoning*, volume 3155 of *Lecture Notes in Computer Science*, pages 106–118. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22882-0.
14. Henriette Cramer, Vanessa Evers, Satyan Ramlal, Maarten Someren, Lloyd Rutledge, Natalia Stash, Lora Aroyo, and Bob Wielinga. 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, November 2008. ISSN 0924-1868.
15. Davor Cubranic, Gail C. Murphy, Janice Singer, and Kellogg S. Booth. Hipikat: A project memory for software development. *IEEE Trans. Softw. Eng.*, 31(6):446–465, June 2005. ISSN 0098-5589.

16. Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 271–280, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-654-7.
17. Xavier Dolques, Aymen Dogui, Jean-Rémy Falleri, Marianne Huchard, Clémentine Nebut, and François Pfister. Easing model transformation learning with automatically aligned examples. In *Proceedings of the 7th European conference on Modelling foundations and applications*, ECMFA'11, pages 189–204, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-21469-1.
18. Cynthia Dwork. Differential privacy: A survey of results. In Manindra Agrawal, Dingzhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation*, volume 4978 of *Lecture Notes in Computer Science*, pages 1–19. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-79227-7.
19. Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 257–260, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-906-0.
20. T. George and S. Merugu. A scalable collaborative filtering framework based on co-clustering. In *Data Mining, Fifth IEEE International Conference on*, page 4 pp., nov. 2005.
21. Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, pages 439–446, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence. ISBN 0-262-51106-1.
22. Peng Han, Bo Xie, Fan Yang, and Ruimin Shen. A scalable p2p recommender system based on distributed collaborative filtering. *Expert Systems with Applications*, 27(2):203 – 210, 2004. ISSN 0957-4174.
23. Hans-Jörg Happel and Walid Maalej. Potentials and challenges of recommendation systems for software development. In *Proceedings of the 2008 international workshop on Recommendation systems for software engineering*, RSSE '08, pages 11–15, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-228-3.
24. Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, CSCW '00, pages 241–250, New York, NY, USA, 2000. ACM. ISBN 1-58113-222-0.
25. Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, January 2004. ISSN 1046-8188.
26. Félix Hernández del Olmo and Elena Gaudioso. Evaluation of recommender systems: A new approach. *Expert Syst. Appl.*, 35(3):790–804, October 2008. ISSN 0957-4174.
27. Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, October 2002. ISSN 1046-8188.
28. George Karypis. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the tenth international conference on Information and knowledge management*, CIKM '01, pages 247–254, New York, NY, USA, 2001. ACM. ISBN 1-58113-436-3.
29. M.G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.
30. M.G. Kendall. The treatment of ties in ranking problems. *Biometrika*, pages 239–251, 1945.
31. B. Kille and S. Albayrak. Modeling difficulty in recommender systems. In *Workshop on Recommendation Utility Evaluation: Beyond RMSE (RUE 2011)*, page 30, 2012.
32. Barbara A. Kitchenham and Shari Lawrence Pfleeger. Principles of survey research: part 3: constructing a survey instrument. *SIGSOFT Softw. Eng. Notes*, 27(2):20–24, March 2002. ISSN 0163-5948.

33. Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. ISSN 0018-9162.
34. I. Koychev and I. Schwab. Adaptation to drifting user's interests. In *Proceedings of ECML2000 Workshop: Machine Learning in New Information Age*, pages 39–46, 2000.
35. Balachander Krishnamurthy, Delfina Malandrino, and Craig E. Wills. Measuring privacy loss and the impact of privacy protection in web browsing. In *Proceedings of the 3rd symposium on Usable privacy and security*, SOUPS '07, pages 52–63, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-801-5.
36. Ludmila I. Kuncheva and Christopher J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Mach. Learn.*, 51(2):181–207, May 2003. ISSN 0885-6125.
37. Shyong K. Lam and John Riedl. Shilling recommender systems for fun and profit. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 393–402, New York, NY, USA, 2004. ACM. ISBN 1-58113-844-X.
38. Shyong-Tony Lam, Dan Frankowski, and John Riedl. Do you trust your recommendations? an exploration of security and privacy issues in recommender systems. In *Emerging Trends in Information and Communication Security*, volume 3995 of *Lecture Notes in Computer Science*, pages 14–29. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-34640-1.
39. Neal Lathia, Stephen Hailes, Licia Capra, and Xavier Amatriain. Temporal diversity in recommender systems. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 210–217, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0153-4.
40. Q Le and A Smola. Direct optimization of ranking measures. *arXiv preprint arXiv:0704.3359*, 2007.
41. Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Trans. Softw. Eng. Methodol.*, 16(4), September 2007. ISSN 1049-331X.
42. Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, RecSys '07, pages 17–24, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-730–8.
43. Frank Mccarey, Mel Ó. Cinnéide, and Nicholas Kushmerick. Rascal: A recommender agent for agile reuse. *Artif. Intell. Rev.*, 24(3-4):253–276, November 2005. ISSN 0269-2821.
44. Sean M. McNee, John Riedl, and Joseph A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 1097–1101, New York, NY, USA, 2006. ACM. ISBN 1-59593-298-4.
45. S.M. McNee. *Meeting user information needs in recommender systems*. PhD thesis, University of Minnesota, 2006.
46. Frank McSherry and Ilya Mironov. Differentially private recommender systems: building privacy into the net. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 627–636, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9.
47. Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 117–, Washington, DC, USA, 2002. IEEE Computer Society.
48. F. Meyer, F. Fessant, F. Clérot, and E. Gaussier. Toward a new protocol to evaluate recommender systems. In *Workshop on Recommendation Utility Evaluation: Beyond RMSE (RUE 2012)*, pages 9–14, New York, NY, USA, 2012. ACM.
49. Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Trans. Internet Technol.*, 7(4), October 2007. ISSN 1533-5399.

50. Audris Mockus and James D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 503–512, New York, NY, USA, 2002. ACM. ISBN 1-58113-472-X.
51. J. Nielsen and J.A.T. Hackos. *Usability engineering*, volume 125184069. Academic press San Diego, 1993.
52. John O'Donovan and Barry Smyth. Trust in recommender systems. In *Proceedings of the 10th international conference on Intelligent user interfaces*, IUI '05, pages 167–174, New York, NY, USA, 2005. ACM. ISBN 1-58113-894-6.
53. Michael O'Mahony, Neil Hurley, Nicholas Kushmerick, and Guénolé Silvestre. Collaborative recommendation: A robustness analysis. *ACM Trans. Internet Technol.*, 4(4):344–377, November 2004. ISSN 1533-5399.
54. A. Ant Ozok, Quyin Fan, and Anthony F. Norcio. Design guidelines for effective recommender system interfaces based on a usability criteria conceptual model: results from a college student population. *Behav. Inf. Technol.*, 29(1):57–83, January 2010. ISSN 0144-929X. doi: 10.1080/01449290903004012. URL http://dx.doi.org/10.1080/01449290903004012.
55. J.R. Quinlan. *C4.5: programs for machine learning*, volume 1. Morgan kaufmann, 1993.
56. Naren Ramakrishnan, Benjamin J. Keller, Batul J. Mirza, Ananth Y. Grama, and George Karypis. Privacy risks in recommender systems. *IEEE Internet Computing*, 5(6):54–62, November 2001. ISSN 1089-7801.
57. A.M. Rashid, I. Albert, D. Cosley, S.K. Lam, S.M. McNee, J.A. Konstan, and J. Riedl. Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134. ACM, 2002.
58. M. Robillard, R. Walker, and T. Zimmermann. Recommendation systems for software engineering. *Software, IEEE*, 27(4):80 –86, july-aug. 2010. ISSN 0740-7459.
59. Martin P. Robillard. Topology analysis of software dependencies. *ACM Trans. Softw. Eng. Methodol.*, 17(4):1–36, August 2008. ISSN 1049-331X.
60. Neil Rubens, Dain Kaplan, and Masashi Sugiyama. Active learning in recommender systems. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 735–767. Springer US, 2011. ISBN 978-0-387-85819-7.
61. Alan Said, Domonkos Tikk, Klara Stumpf, Yue Shi, Martha A. Larson, and Paolo Cremonesi. Recommender systems evaluation: A 3d benchmark. In *Workshop on Recommendation Utility Evaluation: Beyond RMSE*, page 21, 2012.
62. Felix Salfner, Maren Lenk, and Miroslaw Malek. A survey of online failure prediction methods. *ACM Comput. Surv.*, 42(3):10:1–10:42, March 2010. ISSN 0360-0300.
63. J. J. Sandvig, Bamshad Mobasher, and Robin Burke. Robustness of collaborative recommendation based on association rule mining. In *Proceedings of the 2007 ACM conference on Recommender systems*, RecSys '07, pages 105–112, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-730–8.
64. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, DTIC Document, 2000.
65. Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM. ISBN 1-58113-348-0.
66. Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, pages 253–260, New York, NY, USA, 2002. ACM. ISBN 1-58113-561-0.
67. G. Schroder, M. Thiele, and W. Lehner. Setting goals and choosing metrics for recommender system evaluation. In *User-Centric Evaluation of Recommender Systems and Their Interfaces (UCERSTI2 - RECSYS)*, 2011.

68. C.E. Seminario and D.C. Wilson. Robustness and accuracy tradeoffs for recommender systems under attack. In *Proceedings of the 25th Florida Artificial Intelligence Research Society Conference (FLAIRS-25)*, 2012.
69. Serendipity. *Oxford British & World English dictionary*. Oxford University Press, 2013. URL www.oxforddictionaries.com.
70. Guy Shani and Asela Gunawardana. Evaluating recommendation systems. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 257–297. Springer US, 2011. ISBN 978-0-387-85819-7.
71. Rashmi Sinha and Kirsten Swearingen. The role of transparency in recommender systems. In *CHI '02 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '02, pages 830–831, New York, NY, USA, 2002. ACM. ISBN 1-58113-454-1.
72. Barry Smyth and Paul McClave. Similarity vs. diversity. In DavidW. Aha and Ian Watson, editors, *Case-Based Reasoning Research and Development*, volume 2080 of *Lecture Notes in Computer Science*, pages 347–361. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-42358-4.
73. Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009, January 2009. ISSN 1687-7470.
74. K. Swearingen and R. Sinha. Beyond algorithms: An hci perspective on recommender systems. In *ACM SIGIR 2001 Workshop on Recommender Systems*, page 11, 2001.
75. Suresh Thummalapenta and Tao Xie. Parseweb: a programmer assistant for reusing open source code on the web. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ASE '07, pages 204–213, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-882-4.
76. N. Tintarev and J. Masthoff. A survey of explanations in recommender systems. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 801 –810, april 2007. doi: 10.1109/ICDEW.2007.4401070.
77. M. Weimer, A. Karatzoglou, Q. Le, A. Smola, et al. Cofirank-maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems 20 21st Annual Conference on Neural Information Processing Systems 2007*, pages 222–230, 2007.
78. Y. Y. Yao. Measuring retrieval effectiveness based on user preference of documents. *J. Am. Soc. Inf. Sci.*, 46(2):133–145, March 1995. ISSN 0002-8231.
79. Yunwen Ye and Gerhard Fischer. Reuse-conducive development environments. *Automated Software Engg.*, 12(2):199–235, April 2005. ISSN 0928-8910.
80. Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 22–32, New York, NY, USA, 2005. ACM. ISBN 1-59593-046-9.