Proceedings of the

Workshop on Recommendation Utility Evaluation: Beyond RMSE (RUE 2011)

held at the

6th ACM International Conference on Recommender Systems (RecSys 2012)

9 September 2012

Dublin, Ireland

Edited by

Xavier Amatriain[1], Pablo Castells[2], Arjen de Vries[3],
Christian Posse[4], Harald Steck[1]

[1] Netflix, USA
[2] Universidad Autónoma de Madrid, Spain
[3] Centrum Wiskunde & Informatica, Netherlands
[4] Linkedin, USA

Preface

Introduction

Measuring the error in predicting held-out user rating values has been by far the dominant offline evaluation methodology in the Recommender Systems (RS) literature. Yet there seems to be a general consensus in the community that this criterion alone is far from being enough or even adequate to assess the practical effectiveness of a recommender system in matching user needs. The end users of recommendations receive lists of items rather than rating values, whereby recommendation accuracy metrics –as surrogates of the evaluated task– should target the quality of the item selection, rather than the numeric system scores that determine this selection. Furthermore, as far as the order of recommended items determines the set of elements that the user will actually consider for consumption, effectiveness assessment methodologies should target item rankings. For this reason, metrics and methodologies from the Information Retrieval (IR) field –where ranking evaluation has been studied and standardized for decades– have started to be adopted by the RS community. Gaps remain between the methodological formalization of tasks in both fields though, which result in divergences in the adoption of IR methodologies for RS, hindering the interpretation and comparability of empirical observations by different authors.

On the other hand, there is a growing realization that accuracy is only one among several relevant dimensions of recommendation effectiveness. The value of novelty, for instance, has been recognized as a key dimension of recommendation utility for users in real scenarios, in-as-much as the purpose of recommendation is inherently linked to discovery in many application domains. Closely related to novelty, diversity is also a desirable quality to enrich the user's experience and enhance his array of relevant choices. Novelty and diversity are generally positive for businesses as well, by favoring the diversity of sales and helping leverage revenues from market niches. As a matter of business performance enhancement, the value added by recommendation can be measured more directly in terms of on-line click-through rate, conversion rate, sales order size increase, returning customers, increased revenue, etc. On the other hand, web portals and social networks commonly face multiple objective optimization problems related to user engagement, requiring appropriate evaluation methodologies for optimizing along the entire recommendation funnel, from the initial click to the real user engagement in subsequent downstream utilities. Other potentially relevant dimensions of effective recommendations for consumers and providers may include confidence, coverage, risk, cost, robustness, etc.

While the need for further extension, formalization, clarification and standardization of evaluation methodologies is recognized in the community, this need is still unmet to a large extent. When engaging in evaluation work, researchers and practitioners are still often faced with experimental design questions for which there are currently not always precise and consensual answers. Room re-mains for further methodological development and convergence, which motivated the RUE 2012 workshop

The ACM RecSys 2012 International Workshop on "Recommendation Utility Evaluation: Beyond RMSE" (RUE 2012) gathered researchers and practitioners interested in developing better, clearer, and/or more complete evaluation methodologies for recommender systems –or just seeking clear guidelines for their experimental needs. The workshop provided an informal setting for exchanging and discussing ideas, sharing experiences and viewpoints. RUE sought to identify and better understand the current gaps in recommender system evaluation methodologies, help lay directions for progress in addressing them, and contribute to the consolidation and convergence of experimental methods and practice.

Scope and topics

The accepted papers and the discussions held at the workshop addressed, among others, the following topics:

- Recommendation quality dimensions.
  - Effective accuracy, ranking quality.
  - Novelty, diversity, unexpectedness, serendipity.
  - Utility, gain, cost, risk, benefit.
  - Robustness, confidence, coverage, usability, etc.
- Matching metrics to tasks, needs, and goals.
  - User satisfaction, user perception, human factors.
  - Business-oriented evaluation.
  - Multiple objective optimization, user engagement.
  - Quality of service, quality of experience.
- Evaluation methodology and experimental design.
  - Definition and evaluation of new metrics, studies of existing ones.
  - Adaptation of methodologies from related fields: IR, Machine Learning, HCI, etc.
  - Evaluation theory.
- Practical aspects of evaluation.
  - Offline and online experimental approaches.
  - Simulation-based evaluation.
  - Datasets and benchmarks.
  - Validation of metrics.

Specific questions raised and addressed by the workshop included, among others, the following:

- What are the unmet needs and challenges for evaluation in the RS field? What changes would we like to see? How could we speed up progress?
- What relevant recommendation utility and quality dimensions should be cared for? How can they be captured and measured?
- How can metrics be more clearly and/or formally related to the task, contexts and goals for which a recommender application is deployed?
- How should IR metrics be applied to recommendation tasks? What aspects require adjustment or further clarification? What further disciplines should we draw from (HCI, Machine Learning, etc.)?
- What biases and noise should experimental design typically watch for?
- Can we predict the success of a recommendation algorithm with our offline experiments? What offline metrics correlate better and under which conditions?
- What are the outreach and limitations of offline evaluation? How can online and offline experiments complement each other?
- What type of public datasets and benchmarks would we want to have available, and how can they be built?
- How can the recommendation effect be traced on business outcomes?
- How should the academic evaluation methodologies improve their relevance and usefulness for industrial settings?
- How do we envision the evaluation of recommender systems in the future?

Submissions and Programme

The workshop received 18 submissions, of which 11 were accepted (61%), including 3 full technical papers, 4 position papers, and 4 technical papers presented as posters. The workshop opened with a keynote talk by Carlos Gómez-Uribe, from Netflix, and included several open discussion sessions. We briefly summarize here the presented works and held discussions.

The keynote talk, entitled "Challenges and Limitations in the Offline and Online Evaluation of Recommender Systems: A Netflix Case Study", provided a comprehensive, inside view of the evaluation of recommendation technologies in one of the major players in the recommender system industry. Gómez-Uribe explained and discussed how offline and online (A/B testing) phases, business metrics (cancellation rate, subscriber streaming), long-term vs. short-term performance measures are handled in an online businesses heavily relying on recommendation technologies.

The papers presented after this cover a wide spectrum of topics, encompassing most of the aspects put forward in the intended workshop scope. In the full technical papers section, G. Adomavicius and J. Zhang address a new quality dimension, namely recommendation stability, defined as the consistency of recommendations over small incremental changes in the input data. A method is proposed to enhance the stability (at the same time as the accuracy) of an arbitrary recommender by means of an iterative approach where the system is fed back samples of its output. F. Meyer et al. propose the distinction of four functions in user activity where a recommender system may assist: decision, comparison, discovery, and exploration. The authors suggest associating a specific evaluation metric to each of these dimensions, and a structured evaluation procedure (including the metrics computation) in offline experiments. M. Habibi and A. Popescu-Belis present a crowdsourcing approach to evaluate the accuracy of a filtering system which automatically links documents to human speech. The study addresses such issues as worker's reliability assessment, inter-worker agreement, and evaluation stability.

In the position papers section, A. Said, D. Tikk et al. present a conceptual framework where evaluation considers three dimensions: the business model, the user requirements, and technical constraints, corresponding to the view of the three broad types of stakeholders involved in a recommender application, respectively: vendors, consumers, and service providers. S. Clerger-Tamayo, J. M. Fernández-Luna and J. F. Huete propose a generalization of MAE where the error in rating predictions can be weighted in order to focus the evaluation on specific cases of the user-item space, and identify conditions where recommendation is suboptimal. O. Başkaya and T. Aytekin study the correspondence between rating-based and content-based inter-item similarity, which is a relevant issue for metrics that are based on a generic item similarity function (such as the average intra-list dissimilarity for diversity evaluation). B. Kille considers the fact that different users may not be equally easy to provide accurate recommendations for, and proposes measures to assess user difficulty in this perspective.

In the posters section, W. L. De Mello Neto and A. Nowé contrast offline and online evaluation in the context of recommendation approaches leveraging social network information, considering such issues as transparency and computational complexity, besides recommendation accuracy. K. Oku and F. Hattori present an approach to enhance recommendation serendipity by mixing features from different items as a seed to produce new recommendations. C. E. Seminario and D. C. Wilson report a wide empirical study with the Mahout open source library, focusing on accuracy and coverage, the tradeoffs between such dimensions, and the variations resulting from functional enhancements introduced by the authors. L. Peska and P. Vojtas present an experimental study on a travel agency website, where the extended use of implicit evidence from user interaction as input for recommendation is tested, using clickthrough rate and conversion rate as the primary recommendation performance metrics.

Different issues were addressed in the open discussion sessions during the workshop. A prominently recurrent one was the gap between offline and online evaluation. Academic research is strongly focused on offline experiments using rating prediction error or IR metrics, whereas businesses rely on live A/B testing with real customers using business metrics such as CTR, conversion rate, cancellation rate (equivalently, returning customers), or revenue increase (in its different measurable forms). With sales and profitability as the obvious common baseline denominator, the designation of specific core metrics among these seems use-case dependent. Some businesses, such as Netflix, nonetheless report using offline testing as well, as a preliminary phase prior to online experimentation. Business-oriented evaluation goals typically require longer-term evaluation cycles, where the effects of a feature (e.g. on customer loyalty) can only be measured over an extended period of time (several months). Short-term indicators (such as CTR) are commonly used nonetheless to complement these to some extent.

Some of the pointed out hurdles hindering the connection between academia and industry in this area include the often discussed difficulty for academic researchers to access real-world large-scale datasets, or the availability of a feasible procedure where algorithms from academia and data from vendors might get in contact, while meeting the requirements and constraints of the involved stakeholders (data and algorithm ownership, end-user's privacy, etc.). Public evaluation campaigns such as the Netflix Prize, the CAMRa challenges, the plista Contest, were discussed as very positive moves in this direction, each with their own limitations. Further alternatives were discussed for setting up some form of evaluation platform where systems could be compared not just for accuracy, but for scalability (response time). Paolo Cremonesi discussed also an initiative, currently in perspective, which is aiming in this direction. The open API provided by Mendeley to its data was also described as an available opportunity for researchers to test their algorithms on massive data. Crowdsourcing approaches were furthermore mentioned as an intermediate option, available to researchers, between offline evaluation and full-scale experiments on real application data.

There seemed to be a general consensus on the inadequacy of RMSE as a proxy for user satisfaction, or any proper view on recommendation utility in general. This was expressed from many perspectives: from conceptual rationales (the recommender's task, the user's goals and interaction paradigm with recommendations in real applications) to experiences –offline and online evidence in formal or informal case studies– shared by many participants in the workshop. Several other general concerns were identified regarding the adequacy of different metrics, such as the fact that different contexts may require different metrics, e.g. navigational recommendation may focus on accuracy and diversity, whereas discovery-oriented recommendation may emphasize novelty and serendipity. It is also a common case in industrial contexts that technical requirements and business constraints may have to be traded off with evaluation needs and may override other concerns and observations in A/B testing. Beyond RMSE, the general opinion seemed yet to be that researchers in academia should still focus on generic metrics rather than too specific business-oriented metrics and constraints.

The interest of the workshop theme was underlined, beyond the RUE workshop itself, by the pervading presence of evaluation as an explicit object of research and discussion in the RecSys conference programme, clearly identified as an open area where further work is needed.

Acknowledgments

The organizers would like to thank the Program Committee members for their high-quality and timely evaluation of the submissions; the RecSys 2012 organizers (Neil J. Hurley and his team) and workshop chairs (Jill Freyne and Pearl Pu) for their support in the organization of this workshop; the keynote speaker (Carlos Gómez-Uribe), all the authors and presenters, for their contribution to a high-quality workshop program; and all participants for such fruitful discussions and valuable ideas as were exchanged during the workshop. Thanks are due to all such contributions which made RUE 2012 a successful venue.

Xavier Amatriain
Pablo Castells
Arjen de Vries
Christian Posse
Harald Steck

Organizing Committee

Xavier Amatriain	Netflix, USA
Pablo Castells	Universidad Autónoma de Madrid, Spain
Arjen de Vries	Centrum Wiskunde & Informatica, Netherlands
Christian Posse	Linkedin, USA
Harald Steck	Netflix, USA

Program Committee

Gediminas Adomavicius	University of Minnesota, USA
Alejandro Bellogín	Universidad Autónoma de Madrid, Spain
Iván Cantador	Universidad Autónoma de Madrid, Spain
Licia Capra	University College London, UK
Òscar Celma	Gracenote, USA
Charles Clarke	University of Waterloo, Canada
Paolo Cremonesi	Politecnico di Milano, Italy
Juan Manuel Fernández-Luna	Universidad de Granada, Spain
Pankaj Gupta	Twitter, USA
Juan F. Huete	Universidad de Granada, Spain
Dietmar Jannach	University of Dortmund, Germany
Jaap Kamps	University of Amsterdam, Netherlands
Neal Lathia	University College London, UK
Jérôme Picault	Bell Labs, Alcatel-Lucent, France
Filip Radlinski	Microsoft, Canada
Francesco Ricci	Free University of Bozen-Bolzano, Italy
Fabrizio Silvestri	Consiglio Nazionale delle Ricerche, Italy
David Vallet	Universidad Autónoma de Madrid, Spain
Paulo Villegas	Telefónica R&D, Spain
Jun Wang	University College London, UK
Yi Zhang	University of California, Santa Cruz, USA

Table of Contents

Keynote talk

Carlos Gómez-Uribe
Challenges and Limitations in the Offline and Online Evaluation of Recommender Systems: A Netflix Case Study 1

Full technical papers

Gediminas Adomavicius and Jingjing Zhang
Iterative Smoothing Technique for Improving the Stability of Recommender Systems 3

Frank Meyer, Françoise Fessant, Fabrice Clérot and Eric Gaussier
Toward a New Protocol to Evaluate Recommender Systems 9

Maryam Habibi and Andrei Popescu-Belis
Using Crowdsourcing to Compare Document Recommendation Strategies for Conversations 15

Position papers

Alan Said, Domonkos Tikk, Klara Stumpf, Yue Shi, Martha Larson and Paolo Cremonesi
Recommender Systems Evaluation: A 3D Benchmark 21

Sergio Cleger-Tamayo, Juan M. Fernández-Luna and Juan F. Huete
On the Use of Weighted Mean Absolute Error in Recommender Systems 24

Osman Başkaya and Tevfik Aytekin
How Similar is Rating Similarity to Content Similarity? 27

Benjamin Kille
Modeling Difficulty in Recommender Systems 30

Posters

Wolney Leal De Mello Neto and Ann Nowé
Insights on Social Recommender Systems 33

Kenta Oku and Fumio Hattori
User Evaluation of Fusion-based Recommender Systems for Serendipity-oriented Recommendation 39

Carlos E. Seminario and David C. Wilson
Case Study Evaluation of Mahout as a Recommender Platform 45

Ladislav Peska and Peter Vojtas
Evaluating the Importance of Various Implicit Factors in E-commerce 51

Challenges and Limitations in the Offline and Online Evaluation of Recommender Systems: A Netflix Case Study

Carlos Gomez-Uribe
Netflix, USA
cgomez@netflix.com

ABSTRACT

The typical use case of recommendation systems is suggesting items such as videos, songs or articles to users. Evaluating a recommender system is critical to the process of improving it. In theory the best judges of the quality and effectiveness of a recommender system are the users themselves, e.g., ideal metrics can describe the intensity and frequency of a user's interaction with the system over the long term. In practice, however, despite the wide adoption of consumer science based on online A/B testing for the evaluation and comparison of different recommender systems, user-derived measurements are often noisy, slow, non-repeatable, and sensitive to a myriad of potential confounders. Furthermore, conducting large-scale user experiments for researchers in academia is often impossible. A complementary offline approach can be used to quickly evaluate and optimize new recommender systems on historical user-generated data. Yet these offline measurements need not translate directly onto the sought-after online results, such as increases in user engagement. This talk will describe the blend of offline and online experimentation we use at Netflix to improve upon our recommendation systems, and will discuss some key challenges and limitations of these approaches that are broadly relevant to the recommender systems field.

Iterative Smoothing Technique for Improving Stability of Recommender Systems

Gediminas Adomavicius
University of Minnesota
gedas@umn.edu

Jingjing Zhang
Indiana University
jjzhang@indiana.edu

ABSTRACT

We focus on the measure of recommendation stability, which reflects the consistency of recommender system predictions. Stability is a desired property of recommendation algorithms and has important implications on users' trust and acceptance of recommendations. Prior research has reported that some popular recommendation algorithms can suffer from a high degree of instability. In this study we propose a scalable, general-purpose iterative smoothing approach that can be used in conjunction with different traditional recommendation algorithms to improve their stability. Our experimental results on real-world rating data demonstrate that the proposed approach can achieve substantially higher stability as compared to the original recommendation algorithms. Importantly, the proposed approach not only does not sacrifice the predictive accuracy in order to improve recommendation stability, but is actually able to provide additional accuracy improvements at the same time.

1. INTRODUCTION

Recommender systems represent technologies that assist users in finding a set of interesting or relevant items [1]. In order to provide good recommendations, recommender systems employ users' feedback on consumed items. This input can include explicitly provided feedback in the form of ratings or tags, as well as feedback that can be implicitly inferred by monitoring users' behavior such as browsing, linking, or buying patterns. The most common approach to modeling users' preferences for items is via numeric *ratings*. The recommendation algorithm then analyzes patterns of users' past ratings and predicts users' preference ratings for new, not yet consumed items. Once ratings for the new items are estimated, the item(s) with the highest estimated rating(s) can be recommended to the user.

In the recommender systems literature, evaluating performance of recommendation algorithms has always been a key issue, and recommendation accuracy has been the major focus in developing evaluation metrics [11,23]. As a result, much of the research in the recommender systems area has focused on proposing new techniques to enhance the *accuracy* of recommendation algorithms in predicting what users will like, as exemplified by the recent $1M Netflix prize competition. Prediction accuracy metrics typically compare the rating values estimated by a recommendation algorithm against the actual rating values and reflect the closeness of the system's predictions to users' true ratings. In addition to recommendation accuracy, researchers have proposed a number of alternative types of measures, including recommendation coverage, diversity, novelty, serendipity, and several others, to evaluate the performance of recommender systems [11,23]. Of special interest to us is the recently introduced measure of recommendation *stability* [2], which reflects the level of consistency among the predictions made by the system.

According to the definition, stability is the consistent agreement of predictions made on the same items by the same algorithm, when any new incoming ratings are in complete agreement to system's prior estimations [2]. As has been discussed in prior work, stability is an important and desired property of recommender systems, and has a number of potential implications related to users' trust and acceptance of such systems [2].

While providing stable and consistent recommendations is important in many contexts, prior research has demonstrated that some popular collaborative filtering recommendation algorithms can suffer from high degree of instability [2]. This is particularly true for the widely used item- and user-based nearest-neighbor collaborative filtering approaches. It has also been shown that stability does not necessarily correlate with predictive accuracy [2], i.e., different recommendation algorithms can exhibit different levels of stability, even though they may have similar prediction accuracy. Thus, maximizing accuracy may not necessarily help to improve stability, and vice versa. For instance, a simple heuristic that predicts any unknown user rating as an average of all known ratings of that user is perfectly stable [2]; however, in most real-world settings this heuristic is outperformed by more sophisticated recommendation algorithms in terms of predictive accuracy. Therefore, the main objective of this study is to develop an approach that can improve stability of recommendation algorithms without sacrificing their accuracy.

In this paper, we propose a general *iterative smoothing* approach to improve stability of any given recommendation technique. The approach serves as a *meta-algorithm*, i.e., it can be used in conjunction with any traditional recommendation technique. Accordingly, the paper evaluates the performance of the proposed approach in conjunction with a number of popular and widely-used recommendation algorithms in terms of their stability as well as accuracy on several real-world movie rating datasets. The results show that this meta-algorithmic approach provides substantial improvements in recommendation stability as compared to the original recommendation algorithms, while providing some additional accuracy benefits as well.

2. RELATED WORK

Based on how unknown ratings are predicted, recommendation techniques can be classified into three general categories: content-based, collaborative filtering, and hybrid [1,3]. Among different recommendation approaches, collaborative filtering techniques have been most widely used, largely because they are domain independent, require minimal, if any, information about user and item features, yet can still achieve accurate predictions [13,19].

In a typical setting of collaborative filtering recommender systems, users' preferences for items are modeled via numeric ratings. Thus, the recommendation problem is reduced to the problem of estimating ratings for the items that have not been seen by a user, and this estimation is usually based on the other available ratings given by this and/or other users. More formally, given a set of users U and a set of items I, the entire user-item space is denoted as $S = U \times I$. Let R_{ui} represent rating that user u gave to item i, where R_{ui} is typically known only for a limited subset of all possible (u,i) pairs. Let D be the set of known ratings, and $S\backslash D$ be the set of unknown ratings. Therefore, the recommendation task is to estimate unknown R_{ui} values for all $(u,i) \in S\backslash D$ pairs, given U, I, and known $R(u,i)$ values for $(u,i) \in D$.

As mentioned earlier, predictive accuracy has been a major focus in recommender systems literature. One of the most widely used predictive accuracy metrics for recommender systems is *root mean squared error* (RMSE), which we will use in this paper to report the recommendation accuracy results. However, there is a growing understanding that good recommendation accuracy alone may not give users a satisfying experience using the recommender systems, and that some other (complementary) measures are also important in evaluating the effectiveness of the system [23]. Our focus in this paper is one of these important complementary measures, recommendation *stability* [2].

Recommendation stability measures the inherent consistency among the different predictions made by the system. Consider an example where the system makes predictions for two movies, i_1 and i_2, for user u. Let's denote the two rating predictions as $R^*(u,i_1)$ and $R^*(u,i_2)$. Let's assume that prediction $R^*(u,i_1)$ is precisely accurate and, after user u consumes item i_1, it gets added to the system as part of the known ratings, i.e., $R(u,i_1) = R^*(u,i_1)$. The recommendation algorithm re-computes all other predictions in light of the new data. Would this change the value of the other prediction, $R^*(u,i_2)$, and to what extent, even though the newly incoming rating data was exactly as predicted by the system? In other words, two predictions $R^*(u,i_1)$ and $R^*(u,i_2)$ of the same recommender system can be viewed as "inconsistent" with each other, if adding one of them to the training data for the system changes the other prediction. As discussed in [2], the degree of the change in predictions reflects the (in)stability of the recommendation algorithm.

Specifically, the stability of a recommender system is defined as the extent to which the system's predictions for the same items stay the same/similar (i.e., are stable), when any and all new incoming ratings submitted to the system are in complete agreement with system's prior predictions. Based on this idea, a two-phase approach (illustrated in Figure 1) for computing stability of a recommendation algorithm has been introduced in prior literature [2]. In Phase 1, given a set of known ratings, a predictive model is built, and predictions for all unknown ratings are made. Then, a random subset of system's predictions is added to the original set of known ratings as hypothetical new incoming ratings. In Phase 2, based on expanded training data, a second predictive model is built using the same recommendation technique, and predictions on remaining unknown ratings are made. Stability is then measured by comparing the two sets of predictions to compute their root mean squared difference, which is called *root mean squared shift* (RMSS), and is computed in a similar fashion as RMSE:

$$\text{RMSS} = \sqrt{\sum\nolimits_{(u,i)\in P_1 \cap P_2} (P_1(u,i) - P_2(u,i))^2 \,/\, |P_1 \cap P_2|}$$

where P_1 and P_2 are the predictions made in Phase 1 and 2, respectively, i.e., RMSS captures the shift in predictions made by the same recommendation algorithm with new ratings that are in complete agreement with algorithm's own prior estimations.

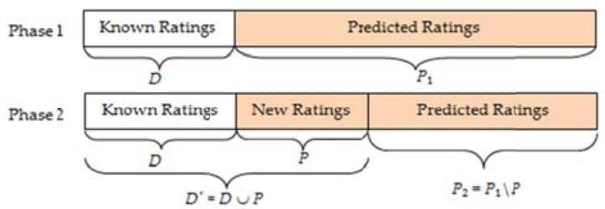


Figure 1. Illustration of stability computation, adapted from [2].

Providing consistent predictions has important implications on users' trust and acceptance of recommendations. In the consumer psychology literature, it has been shown that online advice-giving agents with greater fluctuations in past opinions are considered less informative than those with a more uniform pattern of opinions [9], and advice inconsistent with past recommendations is considered less helpful than consistent advice [7,24]. Inconsistent recommendations may be discredited by users and, as a result, the decrease in users' trust will further reduce their perceptions of the recommender system's competence [12,17]. In contexts where consistency is vital, the instability of a recommender system is likely to have a negative impact on users' acceptance and, thus, harm the success of the system.

Prior research has provided a comprehensive investigation into the stability of popular recommendation algorithms [2] and showed that some widely used algorithms can be highly unstable. For example, it has been shown that RMSS for user-based collaborative filtering approach can be as high as 0.47 on the Netflix movie rating data, meaning that on average *every* predicted rating will shift by 0.47 stars (i.e., by about a half-star) after adding to the training data some new ratings that are identical to the system's current predictions [2]. This is a very significant shift in prediction, considering the length of the rating scale for the dataset is only 4 stars (i.e., ratings go from 1 to 5). Thus, techniques for stability improvement are needed. In this paper, we propose a general-purpose meta-algorithmic approach that can be used to improve stability of traditional recommendation algorithms.

3. ITERATIVE SMOOTHING APPROACH

3.1 General Idea

High instability results from predictions that are inconsistent with each other. We propose an iterative smoothing approach, which involves multiple iterations for repeatedly and collectively adjusting the rating predictions of a recommendation algorithm based on its other predictions and, thus, is explicitly aimed at improving consistency of predicted ratings. The key idea of iterative smoothing is that the rating predictions computed during current iteration will be fed back into the data to predict other ratings in subsequent iterations.

Figure 2 provides an overview of the proposed iterative smoothing algorithm, and Figure 3 gives a high-level illustration of the overall process. Given rating space S and training set D where ratings are known, predictions on unknown ratings $S\backslash D$ are first estimated using some standard recommendation algorithm T. These predictions are denoted as P_0. The procedure of iterative smoothing then starts to iteratively adjust estimations for each rating in $S\backslash D$ based on all other ratings in the rating space S (i.e., both known as well as predicted) in order to proactively improve consistency between different predicted ratings.

More specifically, during the k-th iteration (for any k = 1, 2, …), for each unknown user-item pair $(u,i) \in S\backslash D$, a model $f_{k,u,i}$ is built based on training dataset $D_{k,u,i}$. This dataset is constructed to contain all known ratings combined with all predictions on user-item pairs in S\D computed in the previous (k-1) iteration, except for the prediction on (u,i) as indicated in Figure 2. As the result of the k-th iteration, each model $f_{k,u,i}$ produces a new prediction for (u,i) that is stored as $P_k(u,i)$, i.e.:

$$P_k(u,t) = \begin{cases} R(u,i), & for\ (u,i) \in D \\ f_{k,u,i}(u,i), & for\ (u,i) \in S \setminus D \end{cases}$$

Here $R(u,i)$ represents the known rating that user u gave item i, and $f_{k,u,i}$ is the prediction model built based on $D_{k,u,i}$. Figure 4 illustrates the process within each iteration of the iterative smoothing approach. The set of predictions made in the current iteration is compared with predictions made in the previous iteration to compute the deviation between the two sets (measured in root mean squared difference). The iterative smoothing process stops either after a fixed, pre-determined number of iterations or when predictions on unknown ratings do not change.

By definition, in each iteration, a separate model is built for every user-item pair with unknown rating. Thus, in total, $|S\setminus D| \cdot K$ models need to be constructed over the course of K iterations, each using $|S|–1$ ratings as a training dataset. If $t(x)$ is the time needed to build a predictive model on data sample of size x using recommendation algorithm T, then the time complexity of the iterative smoothing approach is $O(|S\setminus D| \cdot K \cdot t(|S|))$.

```
Iterative Smoothing Algorithm:
Inputs:   known ratings data $D$, # of iterations $K$, algorithm $T$
Process:
1. Build model $f_0$ on known ratings $D$ using some standard
   recommendation algorithm $T$, i.e., $f_0 \leftarrow T(D)$
2. Apply model $f_0$ to compute predictions $P_0$ for unknown
   ratings $S\setminus D$, i.e., $P_0(u,i) = f_0(u,i)$ for $(u,i) \in S\setminus D$
3. For each iteration $k \in \{1, \ldots, K\}$
     For each unknown rating pair $(u, i) \in S\setminus D$
       a. Construct dataset $D_{k,u,i}$ by including all known ratings $D$
          and all predicted ratings $P_{k-1}$ from the previous iteration,
          except for rating $P_{k-1}(u,i)$, i.e.,
              $D_{k,u,i} = D \cup P_{k-1} \setminus \{P_{k-1}(u,i)\}$
       b. Build model $f_{k,u,i}$ on dataset $D_{k,u,i}$ using $T$, i.e.,
              $f_{k,u,i} \leftarrow T(D_{k,u,i})$
       c. Make prediction on $(u, i)$ and store in $P_k$, i.e.,
              $P_k(u, i) = f_{k,u,i}(u, i)$
4. Output predictions made in the final iteration $P_K$
Output: $P_K$
```

Figure 2. Iterative smoothing approach.

In other words, the complexity of iterative smoothing algorithm is proportional to the size of unknown ratings in the rating space. This computational requirement is likely to be prohibitively expensive for many real-world scenarios, i.e., anytime the user-item rating space is large and rating data is sparse. For example, the Movielens 100K dataset is a publicly available movie rating dataset [10], which is considered to be relatively small. This dataset contains 100,000 movie ratings from 943 users on 1682 movies. Thus, the total number of possible ratings is about 1.6M (i.e., 1682 x 943), of which 1.5M is unknown. If we apply iterative smoothing algorithm on the Movielens 100K dataset, in total 1.5 million predictive models (each built on 1.6 million ratings) would need to be constructed within each iteration. Therefore, applying the full iterative smoothing approach may not be feasible even for datasets that are not very large. In order to overcome this complexity issue, in the next section we propose a variation of the iterative smoothing algorithm that offers significant scalability improvements while retaining most of the stability benefits (as will be shown in the Results section).

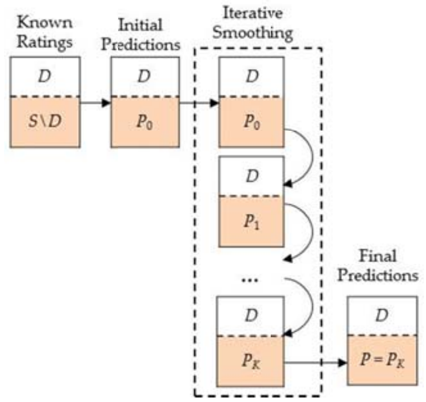


Figure 3. Illustration of the general iterative smoothing process.

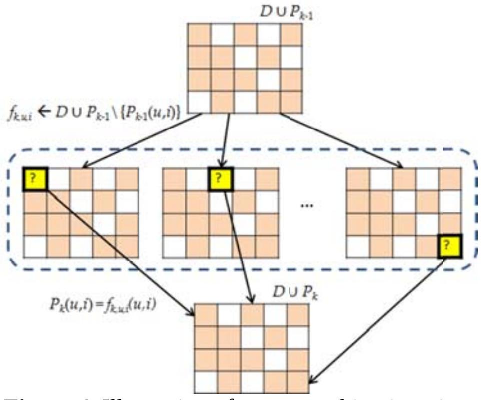

$D \cup P_{k-1}$
$f_{k,u,i} \leftarrow D \cup P_{k-1} \setminus \{P_{k-1}(u,i)\}$
?
?
...
?
$P_k(u,i) = f_{k,u,i}(u,i)$
$D \cup P_k$

Figure 4. Illustration of one smoothing iteration.

3.2 Scalable Iterative Smoothing

We propose a variation of iterative smoothing approach that substantially simplifies the original approach and significantly reduces its computational requirements.

Figure 5 provides an overview of the proposed scalable iterative smoothing algorithm. Similarly to the original iterative smoothing approach, the proposed scalable version also involves multiple iterations for repeatedly adjusting estimations for each unknown rating. However, in each iteration, instead of building one model for each unknown rating, *only one single* model is built upon all known ratings and predicted ratings in previous iteration.

Specifically, during k-th iteration (for any k = 1, 2, …), model f_k is built based on training dataset D_k which contains all known ratings D combined with all predictions P_{k-1} on user-item pairs in $S\setminus D$ computed in the previous iteration. $P_k(u,i)$ denotes the predicted rating for (u,i) in the k iteration by the collective inference approach, defined as follows:

$$P_k(u,t) = \begin{cases} R(u,i), & for(u,i) \in D \\ f_k(u,i), & for\ (u,i) \in S \setminus D \end{cases}$$

Here $R(u,i)$ represents the known rating that user u gave item i. As the result of the k-th iteration, for each $(u,i) \in S\setminus D$, the model f_k produces a new prediction for (u,i) that is stored as $P_k(u,i)$. Similarly to the original approach, new predictions are compared with predictions made in the previous iteration and the procedure ends either after a fixed number of iterations or when predictions on unknown ratings converge (i.e., do not change).

The key difference between this simplified variation and the original iterative smoothing algorithm is the number of predictive models built within each iteration k. The original algorithm builds a separate model for every unknown rating in the rating space, in order to properly adjust the predicted rating $P_k(u,i)$ using *all other* ratings from previous iteration, i.e., all ratings from D as well all ratings $P_{k-1}(u',i')$ where $(u',i') \neq (u,i)$.

```
Scalable Iterative Smoothing Algorithm:
Inputs: known ratings data $D$, # of iterations $K$, algorithm $T$
Process:
1. Build model $f_0$ on known ratings $D$ using some standard recommendation algorithm $T$, i.e., $f_0 \leftarrow T(D)$
2. Apply model $f_0$ to compute predictions $P_0$ for unknown ratings $S\backslash D$, i.e., $P_0(u,i) = f_0(u,i)$ for $(u,i) \in S\backslash D$
3. For each iteration $k \in \{1, \ldots, K\}$
    a. Construct dataset $D_k$ by including all known ratings $D$ and all predicted ratings $P_{k-1}$ from the previous iteration, i.e.,
           $D_k = D \cup P_{k-1}$
    b. Build model $f_k$ on dataset $D_k$ using $T$, i.e.,
           $f_k \leftarrow T(D_k)$
    c. For each unknown rating pair $(u, i) \in S\backslash D$, make prediction on $(u, i)$ and store in $P_k$, i.e.,
           $P_k(u, i) = f_k(u, i)$
4. Output predictions made in the final iteration $P_K$
Output: $P_K$
```

Figure 5. Overview of scalable iterative smoothing approach.

In contrast, the simplified algorithm builds only one predictive model in each iteration, based on the entire rating matrix. In other words, predicted rating $P_k(u,i)$ is adjusted using *all* ratings from previous iteration, i.e., all ratings from D as well as from P_{k-1}, including $P_{k-1}(u,i)$. Thus, in the simplified algorithm, for any given rating prediction $P_1(u,i)$ in the first iteration, the predictive model is built on a rating data (i.e., D_1) that only differs from the rating data used in the original algorithm by one additional rating (i.e., $D_1\backslash\{P_0(u,i)\}$). Because the influence of one additional rating is often subtle, especially when entire rating space is large (i.e., in settings with large numbers of users and items), the single overall model build in the simplified algorithm should produce outcomes similar to the ones produced by individual models built in the original algorithm, especially in the first iteration. While the difference between the original and simplified versions of the iterative smoothing may slowly increase as the number of iterations grows, the simplified approach still provides significant performance improvements (both in stability and accuracy), as demonstrated by the experimental results later in the paper.

Moreover, the runtime complexity of the simplified algorithm is much lower, making it much more practical from the scalability perspective. In particular, as only one overall model is built on all available ratings (i.e., dataset of size $|S|$) within each iteration, in total K models are constructed over the course of K iterations. Thus, the time complexity of the simplified variation is $O(K \cdot t(|S|))$. Comparing this to the complexity of the original algorithm, $O(|S\backslash D| \cdot K \cdot t(|S|))$, the scalable heuristic offers huge computational improvements (i.e., by roughly $|S\backslash D|$ times). In this paper, we use scalable iterative smoothing in our experiments.

4. EXPERIMENTAL RESULTS

4.1 Overall Process

Our experiments test the stability improvements achieved by the proposed meta-algorithmic approach in conjunction with several popular collaborative filtering techniques.

The experiments follow the two-phase stability computation from prior literature [2], discussed in Section 2. We used the standard train-test data splitting approach and divided known ratings data D into two sets: training data D_T (80%) and validation data D_V (20%), where $D = D_T \cup D_V$ and $D_T \cap D_V = \varnothing$. Training set D_T was used for building rating prediction models, while validation set D_V was reserved exclusively for evaluating the predictive accuracy of the final predictions. Similarly, a randomly chosen half of the unknown rating space E_T was dedicated for the stability evaluation of rating prediction models during the training phase, and the other half of the unknown rating space E_V was reserved exclusively for proper evaluation of the stability of the final predictions. Here, $S\backslash D = E_T \cup E_V$ and $E_T \cap E_V = \varnothing$.

Step 0: Create training and test datasets, i.e., D_T, D_V, E_T, E_V.
Step 1: Find the best model parameters:
- i. Use a portion (e.g., 75%) of training set D_T to build rating prediction models using iterative smoothing.
- ii. Compute model accuracy on other portion (25%) of D_T.
- iii. Compute model stability on predictions made on E_T.
- iv. Repeat steps i-iii with various parameter settings for iterative smoothing (i.e., number of iterations).
- v. Find the best parameters for iterative smoothing, i.e., the specifications that result in best stability and accuracy.

Step 2: Evaluate accuracy and stability of the final predictions
- i. Use the best parameter settings for iterative smoothing.
- ii. Train the system on the entire training set D_T using iterative smoothing.
- iii. Evaluate predictive accuracy on the reserved validation rating set D_V.
- iv. Evaluate recommendation stability on the reserved unknown rating space E_V.

Step 3: Report parameter setting(s) as well as the accuracy and stability of the final predictions.

Figure 6. Overall experimental process.

Additionally, the process of iterative smoothing involves multiple iterations to adjust the predictions of unknown ratings. One of the goals in this study is to find whether predictions converge during the process of iterative smoothing and, if so, when. In addition, there is possibility that iterative smoothing models can "over-adjust" rating predictions after a number of iterations, in their attempt to maximize the performance on training data. Over-fitting is a well-known phenomenon which occurs when a predictive model is fine-tuned to fit the training data (including the random errors, outliers, and noise in the data) too well, which typically leads to diminished predictive performance on test data.

Therefore, for a given algorithm, it is necessary to find the best number of iterations to use on a given dataset in the final iterative smoothing procedure. In order to find the optimal parameter settings, we further used the standard train-test data splitting approach internally *within* the training data to identify the best parameter values for the proposed approach, i.e., that result in best accuracy and stability. The overall experiment process is summarized in Figure 6.

4.2 Recommendation Algorithms

In our experiments, we test the proposed approach in conjunction with four popular recommendation algorithms: the simple baseline method, classic user- and item-based variations of neighborhood-based CF approaches, and the matrix factorization technique. A brief overview of each technique is provided below.

Baseline. In real-world settings, some users may systematically tend to give higher ratings than others, and some universally liked items might receive higher ratings than others. Without normalization, such user and item effects could bias system's predictions. Hence, recommender systems often involve a pre-processing step to remove these "global effects". One common practice is to estimate and remove three effects: the overall mean, the main effect of an item, and the main effect of a user [4]. Such "global effects" can serve as a *baseline estimate* for unknown rating of corresponding user and item, i.e.,

$$b_{ui} = \mu + b_u + b_i ,$$

where μ is the overall average rating, b_u is the average observed deviation from μ on ratings provided by user u, and b_i is the average observed deviation from μ on ratings given to item i. Note that, in all of our experiments (i.e., with all other recommendation algorithms), the ratings data were normalized by removing these global effects. Moreover, this estimate is often used as a baseline recommendation technique for comparison with other recommendation algorithms, i.e., $R^*(u,i) = b_{ui}$, and we investigate its performance in our experiments as well.

User-Based Collaborative Filtering (CF_User). The user-based nearest-neighbor collaborative filtering approach is a heuristic that makes predictions of unknown ratings for a user based on the ratings previously rated by this user's "nearest neighbors", i.e., other users who have similar rating patterns [6,20]. That is, the value of the unknown rating for user u and item i is usually computed as an aggregate of the neighbors' ratings for the same item i. The most common aggregation approach is the weighted sum of the neighbors' ratings, where the similarity of two users is used as a weight. I.e., the more similar user u' and target user u are, the more weight will be carried by the rating provided by user u' on item i in the weighted sum when computing the prediction. Predicted rating for user u on item i is computed as:

$$R^*(u,i) = b_{ui} + \frac{\sum_{v \in N(u,i)} sim_{uv} * (R(v,i) - b_{vi})}{\sum_{v \in N(u,i)} |sim_{uv}|}$$

where $N(u,i)$ is a set of "neighbors" with similar rating patterns to user u and that have provided ratings for item i, sim_{uv} is the similarity between users u and v, and b_{ui} is the baseline estimate for user u on item i. In our implementation, two users must have rated at least 3 items in common to allow computation of similarity between them. The similarity between two users is calculated as Pearson correlation between rating vectors (based on the commonly rated items) of the two users. Prediction of each unknown rating is formulated by combining the preferences of 20 most similar users who have rated the same item.

Item-Based Collaborative Filtering (CF_Item). The user-based collaborative filtering technique also has an analogous item-based version, where the ratings of the nearest-neighbor items are used to predict unknown ratings for a given item. Several studies have presented empirical evidence that item-based algorithms often provide better predictive accuracy than user-based methods (e.g., [22]). Thus, our experiments also test the standard item-based collaborative filtering in conjunction with the proposed approach. Similarly to the settings employed in user-based CF, in our experiments, two items are required to have been rated by 3 common users to allow similarity evaluation between them, and 20 nearest-neighbor items are used to formulate a prediction.

Matrix factorization (SVD). Matrix factorization technique is a model-based (as opposed to heuristic-based) collaborative filtering approach that characterizes items and users via a number of latent factors inferred from known ratings [8,15]. This technique models the $U \times I$ rating space as a product of two sub-matrices: user preference matrix ($U \times L$) and item feature matrix ($L \times I$). Each user and item is described by a vector of L latent variables. In our experiments L is set to be 20. The user vector indicates the preference of the user for several latent features, and the item vector represents an item's importance weights for the same latent features. *Singular value decomposition* (SVD) techniques are used to decompose original rating matrix into the two sub-matrices in an optimal way that minimizes the resulting approximation error. After the two sub-matrices are learned using known ratings, each unknown rating is estimated as a dot-product of the corresponding user- and item-factors vectors. Many variations of matrix factorization techniques have been developed during the recent Netflix Prize competition (e.g., [14,15,18,21]). Our experiments focus on the basic underlying version of the matrix factorization [8]; however, the proposed meta-algorithmic approach can be applied with any variation of this technique.

4.3 Results: Comparing Iterative Smoothing with Standard Recommendation Techniques

The objective of the experiment is to compare the performance of the proposed iterative smoothing approach with standard single-model recommendation techniques on several real world datasets.

The first dataset we used is the Movielens 100K dataset [10], which contains 100,000 known ratings on 1682 movies from 943 users (6.3% data density). Our second dataset is a sample extracted from the Movielens 1M dataset. The original Movielens 1M dataset consists of 1,000,000 ratings for 6040 movies by 3952 users (4.2% data density) [10]. From this dataset we extracted a random sample of 3000 users and 3000 movies. Resulted dataset contains 400,627 known ratings (i.e., 4.45% data density). Our third dataset is sampled from the Netflix 100M dataset used in the recent Netflix Prize competition [5]. Similarly to the second dataset, we sub-sampled 3000 random users and 3000 random movies from the original data file. The result data sample consists of 105,256 known ratings (i.e., 1.17% data density). The three datasets used in our experiments come from different sources and have different data characteristics (i.e., size and sparsity). All movie ratings in the Movielens and Netflix datasets are integer values between 1 and 5, where 1 represents the least liked movies, and 5 represents the most liked movies. The datasets used in this experiment are summarized in Table 1.

Table 1. Summary of Experimental Datasets.

DataSet	Description	Users	Items	Density
Movielens 100K	Movie ratings from Movielens movie recommender system.	943	1682	6.30%
Movielens 1M		3000	3000	4.45%
Netflix	Movie ratings distributed by Netflix company.	3000	3000	1.17%

The procedure of this experiment followed the general experimental process described in Figure 6. We examined the prediction accuracy and stability of the final predictions on the reserved validation datasets (as described in Figure 6, Step 2). For each recommendation algorithm used in our study, we compare two approaches: the standard (original) single-model approach and the scalable version of iterative smoothing approach. Accuracy and stability numbers (measured by RMSE and RMSS) of the two approaches on real-world movie rating datasets are provided in Table 2.

Experimental results are consistent across different datasets. On all three datasets, our proposed meta-algorithmic iterative smoothing approach outperformed the original recommendation techniques in both stability and accuracy in the vast majority of cases. In particular, on average (i.e., across all datasets), iterative smoothing provided a dramatic 55% improvement over the original recommendation algorithms in stability (as measured by RMSS) for CF_User and CF_Item algorithms. Even for the fairly stable SVD and baseline techniques, on average, iterative smoothing was able to further improve RMSS by 14%. In terms of predictive accuracy, on average, iterative smoothing provided

1.4% improvements in RMSE across different algorithms.

Table 2. Iterative Smoothing vs. Standard Techniques.

	Method	**Approach**	**Accuracy (RMSE)**	**Stability (RMSS)**
Movielens 100K	SVD	Standard	0.9437	0.0866
		Smoothing	0.9378	0.0779
	CF_User	Standard	0.9684	0.4023
		Smoothing	0.9586	0.2020
	CF_Item	Standard	0.9560	0.3234
		Smoothing	0.9320	0.1347
	Baseline	Standard	0.9758	0.1148
		Smoothing	0.9609	0.0569
Movielens 1M	SVD	Standard	0.8846	0.0804
		Smoothing	0.8798	0.0719
	CF_User	Standard	0.9393	0.3294
		Smoothing	0.9182	0.1145
	CF_Item	Standard	0.9135	0.2755
		Smoothing	0.8929	0.1089
	Baseline	Standard	0.9425	0.0910
		Smoothing	0.9346	0.0923
Netflix	SVD	Standard	0.9372	0.1208
		Smoothing	0.9363	0.1137
	CF_User	Standard	0.9608	0.4610
		Smoothing	0.9407	0.2462
	CF_Item	Standard	0.9579	0.4394
		Smoothing	0.9381	0.2291
	Baseline	Standard	0.9622	0.1465
		Smoothing	0.9556	0.1351

5. CONCLUSIONS AND FUTURE WORK

This paper introduces a general-purpose, practical meta-algorithmic approach – based on iterative smoothing – for improving stability of a variety of traditional recommendation algorithms. The iterative smoothing approach uses multiple iterations to repeatedly and explicitly adjust predictions of a recommendation algorithm based on its other predictions in order to make them more consistent with each other. We examined the performance of iterative smoothing approach on several real world datasets. Our experiments show that the proposed approach demonstrates effectiveness in their ability to improve stability for several widely used recommendation algorithms. Perhaps as importantly, the proposed approach not only does not sacrifice the predictive accuracy to obtain these stability improvements, but actually is able to provide some additional accuracy improvements at the same time.

This work provides several interesting directions for future research. This study shows that iterative smoothing can improve stability for different recommendation algorithms, providing larger improvements for some algorithms and smaller improvements for some others. Providing some additional theoretical understanding of what algorithmic and data characteristics can lead to larger vs. smaller improvements in recommendation stability for the proposed approach is an important direction for future work. Another interesting direction would be to perform user behavior studies to investigate the value of stable (i.e., as opposed to unstable) recommendations on users' usage patterns and acceptance of recommender systems.

ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation grant IIS-0546443.

REFERENCES

[1] Adomavicius, G., and Tuzhilin, A. 2005. "Toward the Next Generation of Recommendation Systems: A Survey of the State-of-the-Art and Possible Extensions," *IEEE Transactions on Knowledge and Data Engineering*, 17, (6), 734-749.

[2] Adomavicius, G., and Zhang, J. 2010. "On the Stability of Recommendation Algorithms," *ACM Conference on Recommender systems*, Barcelona, Spain: ACM New York, NY, USA 47-54.

[3] Balabanovic, M., and Shoham, Y. 1997. "Fab: Content-Based, Collaborative Recommendation," *Comm. of ACM*, 40, (3), 66-72.

[4] Bell, R.M., and Koren, Y. 2007. "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights " *Seventh IEEE International Conference on Data Mining*, Omaha, NE, USA.

[5] Bennett, J., and Lanning, S. 2007. "The Netflix Prize," *KDD-Cup and Workshop*, San Jose, CA.

[6] Breese, J.S., Heckerman, D., and Kadie, C. 1998. "Empirical Analysis of Predictive Algorithms for Collaborative Filtering," *14th Conf. on Uncertainty in Artificial Intelligence*, Madison, WI.

[7] D'astous, A., and Touil, N. 1999. "Consumer Evaluations of Movies on the Basis of Critics' Judgments," *Psychology & Marketing*, 16, 677–694.

[8] Funk, S. 2006. "Netflix Update: Try This at Home."

[9] Gershoff, A., Mukherjee, A., and Mukhopadhyay, A. 2003. "Consumer Acceptance of Online Agent Advice: Extremity and Positivity Effects," *J. of Consumer Psychology*, 13, (1&2), 161-170.

[10] Grouplens 2011. "Movielens Data Sets."

[11] Herlocker, J.L., Konstan, J.A., Terveen, K., and Riedl, J.T. 2004. "Evaluating Collaborative Filtering Recommender Systems," *ACM Transactions on Information Systems*, 22, (1), Jan, 5-53.

[12] Komiak, S., and Benbasat, I. 2006. "The Effects of Personalization and Familiarity on Trust and Adoption of Recommendation Agents," *MIS Quarterly*, 30, (4), 941-960.

[13] Koren, Y. 2009. "The Bellkor Solution to the Netflix Grand Prize." from http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf.

[14] Koren, Y. 2008. "Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model," *ACM Intl. Conf. on Knowledge Discovery and Data Mining (KDD'08)*, 426-434.

[15] Koren, Y., Bell, R., and Volinsky, C. 2009. "Matrix Factorization Techniques for Recommender Systems," *IEEE Computer*, 42, 30-37.

[16] Macskassy, S.A., and Provost, F. 2007. "Classification in Networked Data: A Toolkit and a Univariate Case Study," *Journal of Machine Learning R*, 8, 935-983.

[17] O'Donovan, J., and Smyth, B. 2005. "Trust in Recommender Systems," *10th Intl. Conf. on Intelligent User Interfaces*.

[18] Paterek, A. 2007. "Improving Regularized Singular Value Decomposition for Collaborative Filtering," *KDDCup*, 39-42.

[19] Pilaszy, I., and Tikk, D. 2009. "Recommending New Movies: Even a Few Ratings Are More Valuable Than Metadata,"*Third ACM Conf. on Recommender systems (RecSys '09)*, 93-100.

[20] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J. 1994. "Grouplens: An Open Architecture for Collaborative Filtering of Netnews.," *Conf. on Comp. Supported Cooperative Work*, 175–186.

[21] Salakhutdinov, R., and Mnih, A. 2008. "Bayesian Probabilistic Matrix Factorization Using Markov Chain Monte Carlo," *25th Intl. Conf. on Machine Learning*, Helsinki, Finland: ACM, 880-887.

[22] Sarwar, B., Karypis, G., Konstan, J.A., and Riedl, J. 2001. "Item-Based Collaborative Filtering Recommendation Algorithms," *10th International WWW Conference*, Hong Kong, 285 - 295.

[23] Shani, G., and Gunawardana, A. 2011. "Evaluating Recommendation Systems," in *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira and P.B. Kantor (eds.). 257-294.

[24] Van Swol, L.M., and Sniezek, J.A. 2005. "Factors Affecting the Acceptance of Expert Advice," *British Journal of Social Psychology*, 44, (3), 443-461.

Toward a New Protocol to Evaluate Recommender Systems

Frank Meyer, Françoise Fessant, Fabrice Clérot
Orange Labs
av. Pierre Marzin
22307 Lannion cedex
France
{franck.meyer,francoise.fessant,fabrice.clerot}@orange.com

Eric Gaussier
University of Grenoble - LIG
UFR IM2AG - LIG/AMA
Grenoble Cedex 9
France
eric.gaussier@imag.fr

ABSTRACT

In this paper, we propose an approach to analyze the performance and the added value of automatic recommender systems in an industrial context. We show that recommender systems are multifaceted and can be organized around 4 structuring functions: help users to decide, help users to compare, help users to discover, help users to explore. A global off line protocol is then proposed to evaluate recommender systems. This protocol is based on the definition of appropriate evaluation measures for each aforementioned function. The evaluation protocol is discussed from the perspective of the usefulness and trust of the recommendation. A new measure called Average Measure of Impact is introduced. This measure evaluates the impact of the personalized recommendation. We experiment with two classical methods, K-Nearest Neighbors (KNN) and Matrix Factorization (MF), using the well known dataset: Netflix. A segmentation of both users and items is proposed to finely analyze where the algorithms perform well or badly. We show that the performance is strongly dependent on the segments and that there is no clear correlation between the RMSE and the quality of the recommendation.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering – *collaborative filtering, recommender system;* H.3.4 [**Systems and Software**]: Performance evaluation (efficiency and effectiveness) – *performance measures, usefulness of recommendation.*

General Terms

Algorithms, Measurement, Performance, Experimentation.

Keywords

Recommender systems, Industrial context, evaluation, Compare, Explore, Decide, Discover, RMSE, utility of recommendation

1. INTRODUCTION

The aim of recommender systems is to help users to find items that should interest them, from large catalogs. One frequently adopted measure of the quality of a recommender system is accuracy (for the prediction of ratings of users on items) [1,14]. Yet in many implementations of recommender system services, the rating prediction function is either not provided, or not highlighted when it is provided (in industrial contexts, the generated recommendations themselves and their utility are more important than the rating predictions). There is increasing consensus in the community that accuracy alone is not enough to assess the practical effectiveness and added-value of recommendations [8,13]. Recommender systems in industrial context are multifaceted and we propose to consider them around the definition of 4 key recommendation functions which meet the needs of users facing a huge catalog of items: how to decide, how to compare, how to explore and how to discover. Once the main functions are defined, the next question is how to evaluate a recommender system on its various facets? We will review for each function the key points for their evaluation and the available measures if they exist. In particular, we will introduce a dedicated measure for the function "help to discover". This function raises the question of the evaluation from the point of view of the usefulness of the recommendation. We will also present a global evaluation protocol able to deal with the multifaceted aspect of recommender systems, which requires at least a simple segmentation of users and items. The remainder of the paper is organized as follow: the next section introduces the four core functions of an industrial recommender system. Then the appropriate measures for each core function are presented as well as the global evaluation protocol. The last part of the paper is dedicated to experimental results and conclusion.

Workshop on Recommendation Utility Evaluation: Beyond RMSE (RUE 2012), held in conjunction with ACM *RecSys 2012*. September 9, 2012, Dublin, Ireland.

2. MAIN FEATURES OF RECOMMENDER SYSTEMS

Automatic recommender systems are often used on e-commerce websites. These systems work in conjunction with a search engine for assistance in catalog browsing to help users find relevant content. As many users of e-commerce websites are anonymous, a very important feature is the contextual recommendation of item, for anonymous users. The purpose of these systems being also to increase usage (the audience of a site) or sales, the recommendation itself is more important than the rating predicted. Moreover, prioritizing a list of items on a display page is a more important functionality than the prediction of a rating. These observations, completed with interviews with marketers and project managers of Orange about their requirements relatively to recommender systems and an overview of recommender systems both in the academic and in the industrial fields [10] has led us to organize the recommender systems' functionalities into 4 main features:

Help to Decide. Given an item, a user wants to know if he will appreciate the item. This feature consists of the prediction of a rating for a user and an item and is today mainstream in academic literature [14].

Help to Compare. Given several items, a user wants to know what item to chose. This feature corresponds to a ranking function. It can be used to provide recommendation lists [5] or to provide personalized sorting results of requests on a catalog.

Help to Discover. Given a huge catalog of items, a user wants to find a short list of new interesting items. This feature is usually called item-based top-N recommendation in the academic literature [6]. It corresponds to personalized recommendation. Note that the prediction of the highest rated item is not necessarily the most useful recommendation [5]. For instance the item with the highest predicted rating will most likely be already known by the user.

Help to Explore (or Navigate). Given one item, an (anonymous) user wants to know what the related items are. This feature corresponds to the classical item-to-item recommendation to anonymous users popularized by the e-commerce website Amazon [9] during catalog browsing. This function is widely used in the industry because it can make recommendations for anonymous users, based on the items she consults. It requires a similarity function between items.

3. EVALUATION OF INDUSTRIAL RECOMMENDER SYSTEMS

In this section we discuss the appropriate measures for each core function and a global protocol for the evaluation of the recommender system. The evaluation is viewed from the standpoint of the utility of the recommendation for each user and each item.

3.1 Utility of the recommendation

A good recommender system should avoid bad and trivial recommendations. The fact that a user likes an item and the fact that an item is already known by the user have to be distinguished [7]. A good recommendation corresponds to an item that would probably be well rated by the user but also an item that the user does not know. For instance it is worthless recommending to all users the blockbuster of the year: it should be a good rated movie on the average, but it is not a useful recommendation as most of people may have already seen it.

3.2 Item segmentation and user segmentation

Another important issue for an industrial application is to fully exploit the available catalog, including its long tail, consisting of items rarely purchased [2]. A system's ability to make a recommendation, in a relevant way, for all items in the catalog is therefore important. However Tan and Netessine [16] have observed on the Netflix dataset for instance, that the long tail effect is not so obvious. There's more of a Pareto distribution (20% of the most rated items represents 80% of the global ratings) in the Netflix data than a long tail distribution as proposed by Anderson [2] (where infrequent items globally represent more ratings). They also noticed that the behavior of the users and the type of items they purchase are linked. In particular, customers who watched items in the long tail are in fact heavy users, light users tend to focus only on popular items. These observations lead us to the introduction of the notion of segments of items and users. The definition of the segment thresholds must be relative and catalog dependant. We will use the terms of light/heavy users segment and of unpopular/popular item segment instead of using long tail and short head concepts. In a first step we will use this simple segmentation to analyze how an industrial recommender system can help all users both heavy and light and how it can recommend all items, both popular and unpopular.

3.3 Measures of performance

For our protocol we use a classic train/test split of the data. The train set will be used to compute statistics and thresholds and to build a predictive model. The test set will be used to compute the performance measures. The predictive model should at least be able to provide a rating prediction function for any couple of user and item. We will see that to provide the "Help to Explore" functionality the predictive model also must be able, in some way, to produce an item-item similarity matrix allowing it to select, for each item i, its most similar items (the related items). We first detail the performance measures we use for our protocol, according to the 4 core functions.

Help to Decide. The main use case is a user watching an item description on a screen and wondering if he would enjoy it. Giving a good personalized rating prediction will help the user to choose. The "help to decide" function can be given by the rating prediction function and must be measured by an accuracy measure which penalizes extreme errors. The Root Mean Squared Error (RMSE) is the natural candidate [14].

Help to Compare. The main use case here is a user getting an intermediate short list of items after having given her preferences. This user then wants to compare the items of this short list, in order to choose the one she will enjoy most. The function needs a ranking mechanism with a homogeneous quality of ranking over the catalog. A simple measure is the percentage of compatible rank indexes. After modeling, for each user u and for each couple of item (i, j) in the test set rated by u with $r_{u,i} \neq r_{u,j}$, the preference given by u is compared with the predicted preference given by the recommender method, using the predicted ratings $\hat{r}_{u,i}$ and $\hat{r}_{u,j}$. The percentage of compatible preferences is given by:

$$comp = \frac{\sum_{\{(r_{u,i}) \in \mathrm{T}, (r_{u,j}) \in \mathrm{T}, \mathrm{r_{u,j}} \neq \mathrm{r_{u,i}}\}} c(r_{u,i}, r_{u,j})}{|\{(r_{u,i}) \in \mathrm{T}, (r_{u,j}) \in \mathrm{T}, \mathrm{r_{u,j}} \neq \mathrm{r_{u,i}}\}|} \qquad (3\text{-}1)$$

with $c(r_{u,i}, r_{u,j}) = \delta\left(sign(r_{u,i} - r_{u,j}), sign(\hat{r}_{u,i} - \hat{r}_{u,j})\right)$, where $\delta(sign(r_{u,i} - r_{u,j}), sign(\hat{r}_{u,i} - \hat{r}_{u,j}))$ is 1 if $r_{u,i} - r_{u,j}$ has the same sign as $\hat{r}_{u,i} - \hat{r}_{u,j}$ and 0 otherwise, and $|\{(r_{u,i}) \in \mathrm{T}, (r_{u,j}) \in \mathrm{T}, \mathrm{r_{u,j}} \neq \mathrm{r_{u,i}}\}|$ is the number of elements of $\{(r_{u,i}) \in \mathrm{T}, (r_{u,j}) \in \mathrm{T}, \mathrm{r_{u,j}} \neq \mathrm{r_{u,i}}\}$.

Help to Discover. The main use case here is a user getting recommended items: these recommendations must be relevant and useful. For relevancy our approach is the following: an item i recommended for the user u

- is considered **relevant** if u has rated i in the test set with a rating greater than or equal to u's mean of ratings,

- is considered **irrelevant** if u has rated i in the test set with a rating lower than u's mean of ratings

- is not evaluated if not present for u (not rated by u) in the test set.

The classical measure to evaluate recommendation list is the precision measure (recall being difficult to apply in the context of recommendation, as in huge catalogs one does not know all the items relevant for each user). For each user u:

$$\mathrm{precision_u} = \frac{\text{number of relevant recommended items}_\mathrm{u}}{|\mathrm{H_u}|} \quad (3\text{-}2)$$

H_u stands for the subset of evaluable recommendations in the test set for u, that is to say the set of couples (u,i), i being the recommended item to the user u. $|H_u|$ is the size of H_u, in number of couples (u, i).

However the precision is not able to measure the usefulness of the recommendations: recommending well-known blockbusters, already known by the user will lead to a very high precision although this is of very low utility. To account for this, we introduce here the concept of recommendation impact. The basic idea is that, the more frequent a recommended item is, the less impact the recommendation has. This is summarized in Table 1:

Table 1. The notion of recommendation Impact

	Impact of the recommendation	
	Impact if the user **likes** the item	**Impact** if the user **dislikes** the item
Recommending a popular item	**Low**: the item is likely to be already known at least by name by the user.	**Low**: even if the user dislikes this item he can understand that as a popular item this recommendation is likely to appear... at least at the beginning
Recommending an unpopular (infrequent) item	**High:** the service provided by the recommender system is efficient. The rarest the item was, the less likely the user would have found it alone.	**High:** not only the item was unknown and did not inspire confidence, but it also was not good.

We then define the Average Measure of Impact (AMI) for the performance evaluation of the function "Help user to Discover". The AMI of a recommendation list Z for a user u with an average of rating $\bar{r}_u$ is given by:

$$AMI_u(Z) = \frac{1}{|H_u|} \sum_{(u,i) \in Z, (u,i) \in H} \frac{1}{count(i)} \times sign(r_{u,i} - \bar{r}_u) \times |I| \quad (3\text{-}3)$$

Where H_u denotes the subset of the evaluable recommendations in the test set, Z denotes the set of couples (user, item), representing a set of recommendations, count(i) the number of logs in the train set related to the item i, and $|I|$ the size of the catalog of items.

The rarer an item i (rarity being estimated in the train set), the greater the AMI if i is both recommended and relevant for a user u. The greater the AMI, the better the positive impact of the recommendations on u. The AMI will have to be calibrated as we do not know yet what is a "good AMI". But we can already compare different algorithms, or different recommendation strategies (such as post filtering methods to add serendipity) with this measure.

Help to Explore. The main case here is the item-to-item recommendation for an anonymous user who is watching an item description on a screen: the recommender system should propose items similar to that being watched. We can try to evaluate the performance of this functionality by associating, with each context item i, the KNN of i, using an overall precision measure for the recommended items. But, we will have an issue: it can be more effective to associate each context item i with N items optimized only for precision, rather than N items similar to the context item i. It may be more efficient, to optimize precision, to associate blockbusters for each source item. In fact we want to assess the quality of the Help to Explore (navigate) function: we want a good semantic, meaningful similarity for each associated item. But only an experiment with real users can assess this semantic similarity.

Our solution is to use the underlying item-item similarity matrix for this evaluation. We can assess the overall quality of the pairs of similar items by an indirect method: 1. given a predictive model, find a way to compute similarities between any pair of items, building an item-item similarity matrix. 2. use an item-item K-Nearest Neighbors (KNN) model [12] using this matrix. The assumption is that a good similarity matrix must lead to good performances for other aspects of the recommendation when used into an item-item KNN model. This is the approach we take, using RMSE, precision, and ranking performance measures. For a KNN type algorithm, this analysis is straightforward and simple: the similarity matrix is already the kernel of the model. The algorithms that are not directly based on a similarity measure need a method for extracting the similarities between the items. For matrix-factorization-based algorithm, this can correspond to a method to compute similarities between the factors of the items.

3.4 Evaluation Protocol

The evaluation protocol is then designed thanks to the mapping between the 4 core functions and the associated performance measures as summarized in Table 2.

Table 2. Adapted measures for each core function

Functions	**Quality criterion**	**Measure**
Decide	Accuracy of the rating prediction Penalization of extreme errors to minimize the risk of wrong decision	RMSE
Compare	Good predicted ranking for every couple of items of the catalog	COMP % of compatible rank indexes
Discover	Selection for a user the most preferred items in a list of items Identification of good/bad recommendations Precise, useful, trusted recommendation	(Precision, not recommended!) Average Measure of Impact (AMI)
Explore	Precise recommendations Identification of good/bad recommendations	Similarity matrix leading to good performances, in accuracy, relevancy, usefulness and trust

The following notations are adopted: a log *(u, i, r)* corresponds to a user *u* who rated an item *i* with the rating *r*. *U* is the set of all the users, *I* is the set of all the items. Given a dataset *D* of logs and an algorithm *A*, the evaluation protocol we propose is as follow:

```
Initialization
Randomly split the dataset into 2 datasets train and test
Use the train dataset to generate a model with the algorithm A.
Evaluation
1. For each log (u, i, r) of the test set:
        1.1 compute the predicted rating of the model
        1.2 compute the predicted rating error
2. Use the RMSE which gives an indicator of the performance of the Help
to Decide function.
3. For each user u of U:
        3.1 sort all u's logs of the test set by ratings
        3.2 sort all u's logs of the test set by rating prediction
        3.3 compute COMP comparing the indexes of u's logs and the
        indexes of the predicted ratings of he logs.
```

4. Use the averaged COMP as an indicator of the Help to Compare function.

5. For each item i of I, compute count(i) which is the number of logs in the train set referencing i.

6. For each user u of U:

    6.1 compute the predicted rating of each item i of I.

    6.2 select the top-N highest predicted rating items noted $i_{u,1}$ to $i_{u,N}$ which are the Top-N recommended items.

    6.3 compute the rating average of u, noted $\bar{u}$.

    6.4 for each recommended item $i_{u,j}$ of u:

        6.4.1 check if a corresponding log (u, $i_{u,j}$,r) exists, If so the recommendation of $i_{u,j}$ is evaluable else skip the step 6.4.2.

        6.4.2. If $r \geq \bar{u}$ then the recommendation is considered relevant (and irrelevant in the other case).

    6.5 compute the Precision and the AMI for the evaluable recommendations

7. Use the Precision and the AMI, averaged by users, as the indicators for the Help to Discover Function

8. Specify a way to compute efficiently, using the model of the algorithm A, the similarity between every couple of items (i,j).

9. Compute the similarity matrix of all the couple (i, j) for I×I.

10. Use this similarity matrix as the kernel of an item-item K-Nearest Neighbor model, then run the protocol for the steps 1 to 7 for RMSE, COMP, AMI and Precision to obtain a 4-dimensional indicator of the quality of the Help to Explore function.

4. EXPERIMENTS

4.1 Datasets and configuration

Experiments are conducted on the widely used dataset Netflix [3]. This dataset has the advantage of being public and allows performance comparisons with many others techniques. Agnostic thresholds are used for segments of users and items, depending of datasets. We used simple thresholds based on the mean of the number of ratings to split items into popular items and unpopular (infrequent) items, and similarly to split users into heavy users and light users. For instance, on Netflix, using a Train Set of 90% of the total of logs, the mean of the number of rating for the users is 190 (heavy users are users who gave more than 190 ratings otherwise they are light users) and the mean of number of ratings for the items is 5089 (popular items are items with more than 5089 ratings otherwise they are unpopular items). The number of generated items for the Top-N recommendation is always N=10. All our tests are carried out on this configuration: Personal Computer with 12 GB Ram, processor Intel^{TM} Xeon^{TM} W3530 64-bit-4-core processor running at 2.8 GHz, hard disk of 350 GB. All algorithms and the benchmark process are written in Java^{TM}.

4.2 Algorithms

We chose to use 2 models: fast matrix factorization using the MF algorithm presented in [15] and an item-item KNN algorithm [12]. These algorithms are mainstream techniques for recommender systems. For MF we analyze the effect of the number of factors, for the KNN algorithm we analyze the effect of K, the number of Nearest Neighbor kept in the model. In addition, to compare the performances of these 2 algorithms, 2 baseline algorithms are also used:

- a simple default predictor using the mean of items and the mean of the users (the sum of the two means if available, divided by 2). This algorithm is also used by the KNN algorithm when no KNN items are available for a given item to score.

- a random predictor, generating uniform ratings between [1..5] for each rating prediction.

One industrial requirement of our system was that it could take into account new items and new users every 2 hours. Considering other process and I/O constraints, for all the algorithms the modeling time was then restricted to 1.5 hours. This has implications for the MF algorithm as on Netflix it always reaches an optimum between 16 and 32 factors: this is a constant for all our tests, for all the performances. Beyond 32 factors, MF does not have enough time to converge. Note that this convergence may be slow, longer than 24 hours for more than 100 factors on the Netflix dataset.

Implementations details

Our implementation of MF is similar to those of the BRISMF implementation [15] with a learning rate of 0.030 and a regularization factor of 0.008, with early stopping. Learning process is stopped after 1.5 hours, or when the RMSE increases three consecutive times (the increase or decrease of the RMSE is controlled on a validation set consisting of 1.5% of the train set). We used an implementation of item-item KNN model as described in [11]. The similarity function is the Weighted Pearson similarity [4]. All details about implementations can be found in [10].

5. NUMERICAL RESULTS

The following abbreviations are used for the segmentation of the performance: Huser: Heavy users, Luser: Light users, Pitem: Popular items and Uitem: Unpopular items (the meaning of unpopular is rather "rare", "infrequent"). For MF we analyzed the number of factors used and for KNN the number of NN kept. The full results of our experiments are available in [10].

5.1 “Help to Decide” performances

The global default predictor has a RMSE of 0.964 and the global random predictor has a RMSE of 1.707.

KNN's RMSE performances: Different sizes of neighborhoods (K) have been tested, compliant with our tasks in an industrial context. Increasing K generally increases the performances. However the associated similarity matrix weights must be kept in RAM for efficiency purposes, which is difficult, if not possible, with high values of K. For very large catalog applications, the size of the KNN matrix must be reasonable (up to 200 neighbors in our tests). The KNN method performs well except when K is small and except for the light-user-unpopular item segment (Luser Uitem). There is a significant gap between the RMSE for the LuserUitem segment (RMSE=1.05) and the RMSE of the heavy-user-popular-item segment (RMSE=0.8). Clearly, the KNN model is not adapted to the former, whereas it performs well on the later. Optimal number of neighbors is around K= 100.

MF's RMSE performances: Different numbers of factors have been tested. MF has difficulties modeling the Luser-Uitem segment: on this segment the RMSE never decreases under 0.96. On the contrary the RMSE for heavy-user-popular-item is close to 0.81, and the two symmetrical segments light-user-popular item and heavy-user-unpopular-item both have a good (low) RMSE (0.84 and 0.85). The RMSE decreases when number of the factor increases up to around 20 factors. After that number, the RMSE increases. It is a consequence of our time-constrained early stopping condition. This corresponds to about 140 passes on the train set. The optimal number of factors seems to be between 16 and 32.

5.2 “Help to Compare” performance

The default global predictor has a percentage of compatible rank indexes (COMP) of 69% and the random global predictor has a performance of 49.99%.

MF's and KNN’s ranking performances: The results are given for the time limited version of run for MF. MF outperforms the KNN model for the light user segments (with a COMP of 73.5% for MF and 66% for KNN). For the rest, the performances are similar to those of KNN. The maximum of ranking compatibility is around 77% for heavy users' segments.

5.3 “Help to Discover” performance

5.3.1 Analysis using the Precision

The global default predictor has a precision of 92.86 % which is questionable: one can see that a simple Top-10 based on high rating average is sufficient to obtain good precision performance. The global random predictor has a precision of 53.04%.

KNNs' precision performances: The precision increases as the number of K increases. But the results are not significantly better than that of the default predictor. The precision is better than the default predictor for only the Huser-Pitem segment and only for at least K=200. Under K=100, it seems better to use a default predictor than a KNN predictor for ranking tasks. Nevertheless the Huser-Pitem segment is well modeled: the precision for 10 generated items for the KNN model is greater than 97% for the model with 200 neighborhoods.

MF's precision performances: MF has a better behavior than the KNN model, especially for the light-user-unpopular-item segment (precision of 96% for F=32 factors, precision of 83% for the KNN with K>=100).

5.3.2 Analysis using the AMI

The Average Measure of Impact gives slight negative performances for the random predictor and a small performance to the default predictor: the default predictor "wins" its impact values on Unpopular items. Note that the supports for the different evaluated segments are very different and the weights of the two popular item segments are significantly higher The KNN model behaves significantly better that the default predictor for the AMI. For MF, the behavior is much worse than that a KNN model. In general, the impact of MF is similar to, or lower than that of the default predictor. An analysis according to the segmentation gives a more detailed view of where are the impacts. Numerical results are summarized in Table 3.

Table 3. AMI according to the segmentation

Best model	Huser Pitem	Luser Pitem	Huser Uitem	Luser Uitem	**Global**
MF F=32	0.38	0.26	8.93	10.61	**0.5**
KNN K=100	**0.71**	**0.43**	9.59	8.84	**2.0**
Default Pred	0.29	0.25	**21.22**	**12.31**	**0.5**
Random Pred	0.00	0.03	-5.13	-0.53	**-0.6**
Best algorithm	KNN	KNN	Default Predictor	Default Predictor	**KNN**

5.4 Summary for Decide, Compare, Discover

Four models have been analyzed: a KNN model, a MF model, a random model and a default predictor model, on 3 tasks adapted to a rating-predictor-based recommender system: Decide, Compare, Discover and on 4 user-item segments: heavy-user-popular-item, heavy-user-unpopular-item, light-user-popular-item and light-user-unpopular item. A summary of the results is given in Table 4. An analysis of the results by segments shows that globally, KNN is well adapted for the heavy-user segments and that MF, and the default predictor are well adapted to light-user segments. Globally, for the tasks "Help to Decide" and "Help to Compare", MF is the best-suited algorithm in our tests. For the task "Help to Discover" KNN is more appropriate. Note that a switch-based hybrid recommender [14], based on item and user segmentation could exploit this information to improve the global performances of the system. Finally 3 main facts will have to be considered:

1. Performances strongly vary according to the different segments of users and items.

2. MF, KNN and default methods are complementary as they perform differently across the different segments.

3. RMSE is not strictly linked to other performance measure, as mentioned for instance in [5].

Table 4. Global results, summary

	Heavy Users **Popular items**	**Heavy Users** **Unpopular items**	**Light Users** **Popular Items**	**Light Users** **Unpopular Items**
Decide RMSE	KNN	MF	MF	MF
Compare %Compatible preferences	KNN	KNN	MF	MF
Discover Precision	KNN	MF	Default Predictor	MF
Discover Average Measure of Impact	KNN	Default Predictor	KNN	Default Predictor

When designing a recommender engine, we have to think about the impact of the recommender: recommending popular items to heavy users might be not so useful. On the other hand, it can be illusory to make personalized recommendations of unpopular (and unknown) items to light (and unknown) users. A possible simple strategy could be:

- rely on robust default predictors, for instance based on robust means of items to try to push unknown golden nuggets to unknown users,
- use personalized algorithms to recommend popular items to light users,
- finally, use personalized algorithms to recommend unpopular items of the long tail for heavy "connoisseurs".

5.5 “Help to Explore” performance

To analyze the performance of the "Help to Explore" functionality we have to compare the quality of the similarities extracted from the models. We use the protocol defined before: a good similarity matrix for the task "Help to Explore" is a similarity matrix leading to global good performances, when used in a KNN model. We choose a similarity matrix with 100 neighbors for each item: this is largely enough for item-to-item tasks where generally a page displays 10 to 20 similar items. Results are presented in Table 5 for the KNN models with K=100, comparing KNN computed on

MF's items factors, native KNN and a Random KNN model used as baseline. As item-item similarity matrix is the kernel of a item-item KNN model, compute similarities in this case is straightforward. To compute similarities between items for MF, we use the MF-based representation of items (the vectors of the factor of the items), with a Pearson similarity. The KNN model computed on the MF's factors of the items can be viewed as a MF-emulated KNN model. Note that as the default predictor model based on items' means and users' means cannot itself produce a similarity matrix, it is disqualified for this task. For the RMSE, the MF-Emulated KNN model looses 0.025 point going from 0.844 to 0.870. Compared with other models, it still performs correctly.

Table 5. Quality of an item-item similarity matrix according to 4 measures: results on Netflix

	Native KNN K=100	KNN computed on MF's items factors K=100, number of factors=16
RMSE	0.8440	0.8691
Ranking: % compatible	77.03%	75.67%
Precision	91.90%	86.39%
AMI	2.043	2.025
(Global time of the modeling task)	(5290 seconds)	(3758 seconds)

For the global ranking, the difference between the MF-Emulated model and the native KNN model is still low, whereas a random KNN model performs very badly. For the precision, for a Top-10 ranking, the MF-Emulated KNN model performs significantly worse than a native KNN model. For the Average Measure of Impact, the MF-emulated KNN model and the native KNN model perform almost identically. These results show that MF could be used to implement a similarity function between items to support the "Help to Explore" function, and that MF could be used as a component for faster KNN search.

6. CONCLUSION

We have proposed a new approach to analyze the performance and the added value of automatic Recommender Systems in an industrial context. First, we have defined 4 core functions for these systems, which are: Help users to Decide, Help users to Compare, Help users to Discover, Help users to Explore. Then we proposed a general off-line protocol crossing our 4 core functions with a simple 4 users×items segments to evaluate a recommender system according to the industrial and marketing requirements. We compared two major state of the art methods, item-item KNN and MF, with 2 baselines methods used as reference. We showed that the two major methods are complementary as they perform differently across the different segments. We proposed a new measure, the Average Measure of Impact, to deal with the usefulness and the trust of the recommendations. Using the precision measure, and the AMI, we showed that there is no clear evidence of correlation between the RMSE and the quality of the recommendation. We have demonstrated the utility of our protocol as it may change

- the classical vision of the recommendation evaluation, often focused on the RMSE/MAE measures as they are assumed correlated with the system overall performances,

- and the way to improve the recommender systems to achieve their tasks.

7. REFERENCES

[1] Adomavicius, G. and Tuzhilin, A. 2005. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions, *IEEE Trans. Knowl. Data Eng,* 17 (6), 2005, pp. 734-749.

[2] Anderson, C. 2006. The Long Tail. Why the future of business is selling less of more. Hyperion Verlag.

[3] Bennet, J and Lanning, S. 2007. The Netflix Prize, KDD Cup and Workshop. 2007. www.netflixprize.com.

[4] Candillier, L., Meyer, F., Fessant, F. 2008. Designing Specific Weighted Similarity Measures to Improve Collaborative Filtering Systems. ICDM 2008: 242-255.

[5] Cremonesi, P., Koren, Y., and Turrin, R. 2010. Performance of recommender algorithms on Top-N recommender tasks. RecSys 2010.

[6] Deshpande, M., and Karypis, G. 2004. Item-based top-N recommendation algorithms. In ACM Transactions on Information Systems, 22(1), 143–177.

[7] Herlocker, J. L., Konstan, J. A., Terveen, L.G. and Riedl, J. 2004. Evaluating collaborative filtering recommender systems. In ACM Transactions on Information Systems 22 (1), 5–53.

[8] Knijnenburg, B. P., Willemsen, M.C., Kobsa, A. 2011: A pragmatic procedure to support the user-centric evaluation of recommender systems.RECSYS 2011, 321-324

[9] Linden, G. Smith,.B, and York, J. 2003. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing*, 7 (1), 2003, pp. 76-80.

[10] Meyer, F. 2012. Recommender systems in industrial contexts. ArXiv e-prints. http://arxiv.org/abs/1203.4487.

[11] Meyer; F., Fessant, F. 2011. Reperio: a generic and flexible recommender system. IEEE/WIC/ACM Conference on Web Intelligence, 2011.

[12] Sarwar, B., Karypis, G., Konstan, J., and Reidl, J. 2001. Item-based collaborative filtering recommendation algorithms. In WWW'01: Proceedings of 10th International Conference on World Wide Web, pages 285–295.

[13] Schroder, G., Thiele, M. and Lehner, W. 2011. Setting Goals and Choosing Metrics for Recommender System Evaluation. UCERSTI 2 -RECSYS 2011

[14] Su, X., and Khoshgoftaar, T.M. 2009. A survey of collaborative filtering techniques. In Advances in Artificial Intelligence, 2009.

[15] Takács, G., Pilászy, I., Németh, B., Tikk, D. 2009. Scalable Collaborative Filtering Approaches for Large Recommender Systems. Journal of Machine Learning Research 10: 623-656 2009.

[16] Tan, T.F. and Netessine, S. 2011, Is Tom Cruise Threatened? An Empirical Study of the Impact of Product Variety on Demand Concentration. ICIS 2011.

Using Crowdsourcing to Compare Document Recommendation Strategies for Conversations

Maryam Habibi
Idiap Research Institute and EPFL
Rue Marconi 19, CP 592
1920 Martigny, Switzerland
maryam.habibi@idiap.ch

Andrei Popescu-Belis
Idiap Research Institute
Rue Marconi 19, CP 592
1920 Martigny, Switzerland
andrei.popescu-belis@idiap.ch

ABSTRACT

This paper explores a crowdsourcing approach to the evaluation of a document recommender system intended for use in meetings. The system uses words from the conversation to perform just-in-time document retrieval. We compare several versions of the system, including the use of keywords, retrieval using semantic similarity, and the possibility for user initiative. The system's results are submitted for comparative evaluations to workers recruited via a crowdsourcing platform, Amazon's Mechanical Turk. We introduce a new method, Pearson Correlation Coefficient-Information Entropy (PCC-H), to abstract over the quality of the workers' judgments and produce system-level scores. We measure the workers' reliability by the inter-rater agreement of each of them against the others, and use entropy to weight the difficulty of each comparison task. The proposed evaluation method is shown to be reliable, and the results show that adding user initiative improves the relevance of recommendations.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Query formulation, Retrieval models*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Performance evaluation*

General Terms

Evaluation, Uncertainty, Reliability, Metric

Keywords

Document recommender system, user initiative, crowdsourcing, Amazon Mechanical Turk, comparative evaluation

1. INTRODUCTION

A document recommender system for conversations provides suggestions for potentially relevant documents within a conversation, such as a business meeting. Used as a virtual secretary, the system constantly retrieves documents that are related to the words of the conversation, using automatic speech recognition, but users could also be allowed to make explicit queries. Such a system builds upon previous approaches known as implicit queries, just-in-time retrieval, or zero query terms, which were recently confirmed as a promising research avenue [1].

Evaluating the relevance of recommendations produced by such a system is a challenging task. Evaluation in use requires the full deployment of the system and the setup of numerous evaluation sessions with realistic meetings. That is why alternative solutions based on simulations are important to find. In this paper, we propose to run the document recommender system over a corpus of conversations and to use crowdsourcing to compare the relevance of results in various configurations of the system.

A crowdsourcing platform, here Amazon's Mechanical Turk, is helpful for several reasons. First, we can evaluate a large amount of data in a fast and inexpensive manner. Second, workers are sampled from the general public, which might represent a more realistic user model than the system developers, and have no contact with each other. However, in order to use workers' judgments for relevance evaluation, we have to circumvent the difficulties of measuring the quality of their evaluations, and factor out the biases of individual contributions.

We will define an evaluation protocol using crowdsourcing, which estimates the quality of the workers' judgments by predicting task difficulty and workers' reliability, even if no ground truth to validate the judgments is available. This approach, named Pearson Correlation Coefficient-Information Entropy (PCC-H), is inspired by previous studies of inter-rater agreement as well as by information theory.

This paper is organized as follows. Section 2 describes the document recommender system and the different versions which will be compared. Section 3 reviews previous research on measuring the quality of workers' judgments for relevance evaluation and labeling tasks using crowdsourcing. Section 4 presents our design of the evaluation micro-tasks – "Human Intelligence Tasks" for the Amazon's Mechanical Turk. In Section 5, the proposed PCC-H method for measuring the quality of judgments is explained. Section 6 presents the results of our evaluation experiments, which on the one hand validate the proposed method, and on the other hand indicate the comparative relevance of the different versions of the recommender system.

2. OUTLINE OF THE DOCUMENT RECOMMENDER SYSTEM

The document recommender system under study is the Automatic Content Linking Device (ACLD [15, 16]), which uses real-time automatic speech recognition [8] to extract words from a conversation in a group meeting. The ACLD filters and aggregates the words to prepare queries at regular time intervals. The queries can be addressed to a local database of meeting-related documents, including also transcripts of past meetings if available, but also to a web search engine. The results are then displayed in an unobtrusive manner to the meeting participants, which can consult them if they find them relevant and purposeful.

Since it is difficult to assess the utility of recommended documents from an absolute perspective, we aim instead at comparing variants of the ACLD, in order to assess the improvement (or lack thereof) due to various designs. Here, we will compare four different approaches to the recommendation problem – which is in all cases a cold-start problem, as we don't assume knowledge about participants. Rather, in a pure content-based manner, the ACLD simply aims to find the closest documents to a given stretch of conversation.

The four compared versions are the following ones. Two "standard" versions as in [15] differ by the filtering procedure for the conversation words. One of them (noted AW) uses all the words (except stop words) spoken by users during a specific period (typically, 15 s) to retrieve related documents. The other one (noted KW) filters the words, keeping only keywords from a pre-defined list related to the topic of the meeting.

Two other methods depart from the initial system. One of them implements semantic search (noted SS [16]), which uses a graph-based semantic relatedness measure to perform retrieval. The most recent version allows user initiative (noted UI), that is, it can answer explicit queries addressed by users to the system, with results replacing spontaneous recommendations for one time period. These are processed by the same ASR component, with participants using a specific name for the system ("John") to solve the addressing problem.

In the evaluation experiments presented here, we only use human transcriptions of meetings, to focus on the evaluation of the retrieval strategy itself. We use one meeting (ES2008b) from the AMI Meeting Corpus [6] in which the design of a new remote control for a TV set is discussed. The explicit users' requests for the UI version are simulated by modifying the transcript at 24 different locations where we believe that users are likely to ask explicit queries – a more principled approach for this simulation is currently under study. We restrict the search to the Wikipedia website, mainly because the semantic search system is adapted to this data, using a local copy of it (WEX) that is semantically indexed. Wikipedia is one of the most popular general reference works on the Internet, and recommendations over it are clearly of high potential interest. But alternatively, all our systems (except the semantic one) could also be run with non-restricted web searches via Google, or limited to other web domains or websites.

The 24 fragments of the meeting containing the explicit queries are submitted for comparison. That is, we want to know which of the results displayed by the various versions at the moment following the explicit query are considered most relevant by external judges. As the method allows only binary comparisons, as we will now describe, we will compare UI with the AW and KW versions, and then SS with KW.

3. RELATED WORK

Relevance evaluation is a difficult task because it is subjective and expensive to be performed. Two well-known methods for relevance evaluation are the use of a click-data corpus, or the use of human experts [18]. However, in our case, producing click data or hiring professional workers for relevance evaluation would both be overly expensive. Moreover, it is not clear that evaluation results provided by a narrow range of experts would be generalizable to a broader range of end users. In contrast, crowdsourcing, or peer collaborative annotation, is relatively easy to prototype and to test experimentally, and provides a cheap and fast approach to explicit evaluation. However, it is necessary to consider some problems which are associated to this approach, mainly the reliability of the workers' judgments (including spammers) and the intrinsic knowledge of the workers [3].

Recently, many studies have considered the effect of the task design on relevance evaluation, and proposed design solutions to decrease time and cost of evaluation and to increase the accuracy of results. In [9], several human factors are considered: query design, terminology and pay, with their impact on cost, time and accuracy of annotations. To collect proper results, the effect of user interface guidelines, inter-rater agreement metrics and justification analysis were examined [2], showing e.g. that asking workers to write a short explanation in exchange of a bonus is an efficient method for detecting spammers. In addition, in [11], different batches of tasks were designed to measure the effect of pay, required effort and worker qualifications on the accuracy of resulting labels. Another paper [13] has studied how the distribution of correct answers in the training data affects worker responses, and suggested to use a uniform distribution to avoid biases from unethical workers.

The Technique for Evaluating Relevance by Crowdsourcing (TERC, see [4]) emphasizes the importance of qualification control, e.g. by creating qualification tests that must be passed before performing the actual task. However, another study [2] showed that workers may still perform tasks randomly even after passing qualification tests. Therefore, it is important to perform partial validation of each worker's tasks, and weight the judgments of several workers to produce aggregate scores [4].

Several other studies have focused on Amazon's Mechanical Turk crowdsourcing platform and have proposed techniques to measure the quality of workers' judgments when there is no ground truth to verify them directly [17, 19, 7, 10, 12]. For instance, in [5], the quality of judgments for a labeling task is measured using the inter-rater agreement and majority voting. Expectation maximization (EM) has sometimes been used to estimate true labels in the absence of ground truth, e.g. in [17] for an image labeling task. In order to improve EM-based estimation of the reliability of workers, the confidence of workers in each of their judgments has been used in [7] as an additional feature – the task being dominance level estimation for participants in a conversation. As the performance of the EM algorithm is not guaranteed, a new method [10] was introduced to estimate reliability based on low-rank matrix approximation.

All of the above-mentioned studies assume that tasks share the same level of difficulty. To model both task difficulty and user reliability, an EM-based method named GLAD was proposed by [19] for an image labeling task. However, this method is sensitive to the initialization value, hence a good estimation of labels requires a small amount of data with ground truth annotation [12].

4. SETUP OF THE EXPERIMENT

Amazon's Mechanical Turk (AMT) is a crowdsourcing platform which gives access to a vast pool of online workers paid by requesters to complete human intelligence tasks (HITs). Once designed and published, registered workers that fulfill the requesters' selection criteria are invited by AMT service to work on HITs in exchange for a small amount of money per HIT [3].

As it is difficult to find an absolute relevance score for each version of the ACLD recommender system, we only aim for comparative relevance evaluation between versions. For each pair of versions, a batch of HITs was designed with their results. Each HIT (see example in Fig. 1) contains a fragment of conversation transcript with the two lists of document recommendations to be compared. Only the first six recommendations are kept for each version. The lists from the two compared versions are placed in random positions (first or second) across HITs, to avoid biases from a constant position.

We experimented with two different HIT designs. The first one offers evaluators a binary choice: either the first list is considered more relevant than the second, or vice-versa. In other words, workers are obliged to express a preference for one of the two recommendation sets. This encourages decisions, but of course may be inappropriate when the two answers are of comparable quality, though this may be evened out when averaging over workers. The second design gives workers four choices (as in Figure 1): in addition to the previous two options, they can indicate either that both lists seem equally relevant, or equally irrelevant. In both designs, workers must select exactly one option.

To assign a value to each worker's judgment, a binary coding scheme will be used in the computations below, assigning a value of 1 to the selected option and 0 to all others. The relevance value RV of each recommendation list for a meeting fragment is computed by giving a weight to each worker judgment and averaging them. The *Percentage of Relevance Value*, noted PRV, shows the relevance value of each compared system, and is computed by assigning a weight to each part of the meeting and averaging the relevance values RV for all meeting fragments.

There are 24 meeting fragments, hence 24 HITs in each batch for comparing pairs of systems, for UI vs. AW and UI vs. KW. As user queries are not needed for comparing SS vs. KW, we designed 36 HITs, with 30-second fragments for each. There are 10 workers per HIT, so there are 240 total assignments for UI-vs-KW and for UI-vs-AW (with a 2-choice and 4-choice design for each), and 360 for SS-KW. As workers are paid 0.02 USD per HIT, the cost for the five separate experiments was 33 USD, with an apparent average hourly rate of 1.60 USD. The average time per assignment is almost 50 seconds. All five tasks took only 17 hours to be performed by workers via AMT. For qualification control we allow workers with greater than 95% approval rate or with more than 1000 approved HITs.

5. THE PCC-H METHOD

Majority voting is frequently used to aggregate multiple sources of comparative relevance evaluation. However, this assumes that all HITs share the same difficulty and all the workers are equally reliable. We will take here into account the task difficulty W_q and the workers' reliability r_w, as it was shown that they have a significant impact on the quality of the aggregated judgments. We thus introduce a new computation method called PCC-H, for *Pearson Correlation Coefficient-Information Entropy.*

5.1 Estimating Worker Reliability

The PCC-H method computes the W_q and r_w values in two steps. In a first step, PCC-H estimates the reliability of each worker r_w based on the Pearson correlation of each worker's judgment with the average of all the other workers judgments (see Eq. 1).

$$r_w = \frac{\sum_{a=1}^{A}\sum_{q=1}^{Q}(X_{qwa} - \bar{X_{wa}})(Y_{qa} - \bar{Y_a})}{(Q-1)S_{X_{wa}}S_{Y_a}} \quad (1)$$

In Equation 1, Q is number of meeting fragments, X_{wqa} is the value that worker w assigned to option a of fragment q, X_{wqa} has value 1 if that option a is selected by worker w, otherwise it is 0. $\bar{X}_{wa}$ and $S_{X_{wa}}$ are the expected value and standard deviation of variable X_{wqa} respectively. Y_{qa} is the average value which all other workers assign to the option a of fragment q. $\bar{Y_a}$ and S_{Y_a} are the expected value and standard deviation of variable Y_{qa}.

The value of r_w computed above is used as a weight for computing RV_{qa}, the relevance value of option a of each fragment q, according to Eq. 2 below:

$$RV_{qa} = \frac{\sum_{w=1}^{W} r_w X_{wqa}}{\sum_{w=1}^{W} r_w} \quad (2)$$

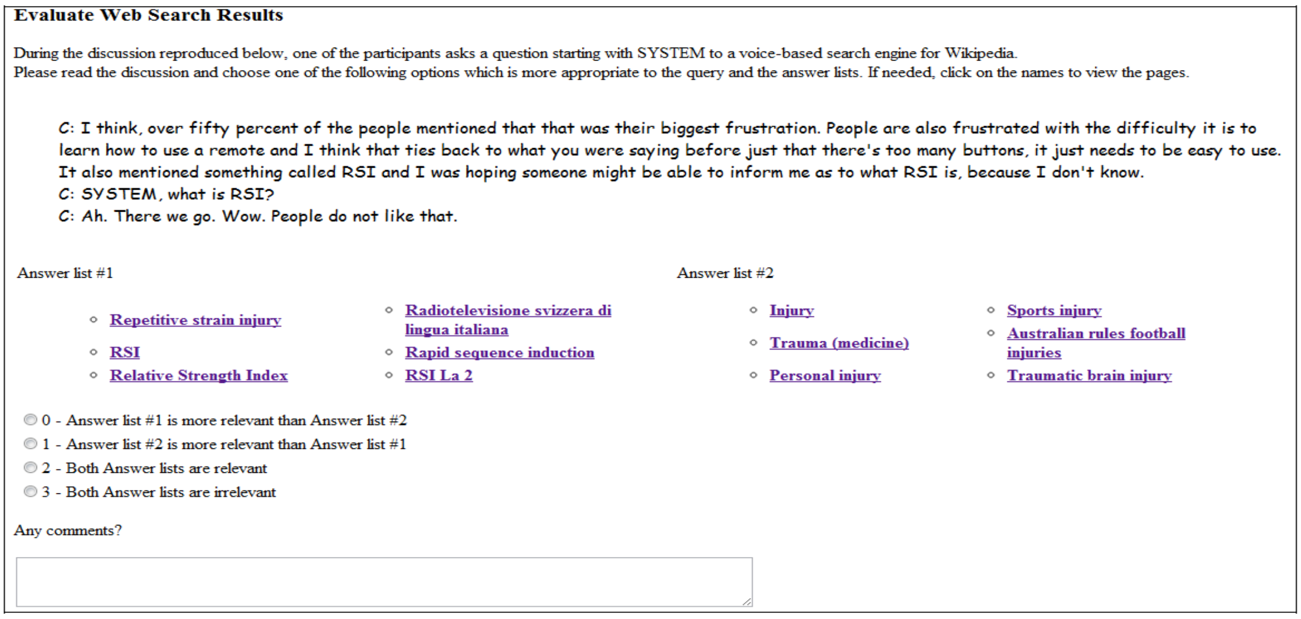
For HIT designs with two options, RV_{qa} shows the relevance value of each answer list a. However, for the four option HIT designs, RV_{ql} for each answer list l is formulated as Eq. 3 below:

$$RV_{ql} = RV_{ql} + \frac{RV_{qb}}{2} - \frac{RV_{qn}}{2} \quad (3)$$

In this equation, half of the relevance value of the case in which both lists are relevant RV_{qb} is added as a reward, and half of the relevance value of the case in which both lists are irrelevant RV_{qn} is subtracted as a penalty from the relevance value of each answer list RV_{ql}.

5.2 Estimating Task Difficulty

In a second step, PCC-H considers the task difficulty for each fragment of the meeting. The goal is to reduce the effect of some fragments of the meeting, in which there is an uncertainty in the workers judgments, e.g. because there are no relevant search results in Wikipedia for the current fragment. To lessen the effect of uncertainty in our judgments, the entropy of answers for each fragment of the meeting is computed and a function of it is used as a weight for each fragment. This weight is used for computing the percentage of relevance value PRV. Entropy, weight and PRV are defined in Eqs. 4–6, where A is the number of options, and H_q and W_q are the entropy and weight of fragment q.


Evaluate Web Search Results

During the discussion reproduced below, one of the participants asks a question starting with SYSTEM to a voice-based search engine for Wikipedia.
Please read the discussion and choose one of the following options which is more appropriate to the query and the answer lists. If needed, click on the names to view the pages.

C: I think, over fifty percent of the people mentioned that that was their biggest frustration. People are also frustrated with the difficulty it is to learn how to use a remote and I think that ties back to what you were saying before just that there's too many buttons, it just needs to be easy to use. It also mentioned something called RSI and I was hoping someone might be able to inform me as to what RSI is, because I don't know.
C: SYSTEM, what is RSI?
C: Ah. There we go. Wow. People do not like that.

Answer list #1

- Repetitive strain injury
- RSI
- Relative Strength Index
- Radiotelevisione svizzera di lingua italiana
- Rapid sequence induction
- RSI La 2

Answer list #2

- Injury
- Trauma (medicine)
- Personal injury
- Sports injury
- Australian rules football injuries
- Traumatic brain injury

0 - Answer list #1 is more relevant than Answer list #2
1 - Answer list #2 is more relevant than Answer list #1
2 - Both Answer lists are relevant
3 - Both Answer lists are irrelevant

Any comments?

Figure 1: Snapshot of a 4-choice HIT: workers read the conversation transcript, examine the two answer lists (with recommended documents for the respective conversation fragment) and select one of the four comparative choices (#1 better than #2, #2 better than #1, both equally good, both equally poor). A short comment can be added.

$$H_q = -\sum_{a=1}^{A} RV_{qa} log(RV_{qa}) \tag{4}$$

$$W_q = 1 - H_q \tag{5}$$

$$PRV_a = \frac{\sum_{q=1}^{Q} W_q RV_{qa}}{\sum_{q=1}^{Q} W_q} \tag{6}$$

6. RESULTS OF THE EXPERIMENTS

Two sets of experiments were performed. First, we attempt to validate the PCC-H method. Then, we apply the PCC-H method to compute PRV for each answer list to conclude which version of the system outperforms the others.

In order to make an initial validation of the workers judgments, we compare the judgments of individual workers with those of an expert. For each worker, the number of fragments for which the answer is the same as the expert's answer is counted, and the total is divided by the number of fragments to compute accuracy. Then we compare this value with r_w, which is estimated as the reliability measurement for each worker's judgment. The percentage of agreement between each worker vs. the expert e_w and the r_w for each worker for one of the batches is shown in Table 1, with an overall agreement between these two values for each worker. In other words, workers who have more similarity with our expert also have more inter-rater agreement with other workers. Since in the general case there is no ground truth (expert) to verify workers judgments, we rely on the inter-rater agreement for the other experiments.

Firstly, equal weights for all the user evaluations and fragments are assigned to compute PRVs for two answer lists of our experiments, which are shown in Table 2.

Table 1: Percentage of agreement between a single worker and the expert, and a single worker and the other workers, for the KW system and 4-choice HITs

Worker #	e_w	r_w
1	0.66	0.81
2	0.54	0.65
3	0.54	0.64
4	0.50	0.71
5	0.50	0.60
6	0.50	0.35
7	0.41	0.24
8	0.39	0.33
9	0.36	0.34
10	0.31	0.12

In this approach, it is assumed that all the workers are reliable and all the fragments share the same difficulty. To handle workers' reliability, we consider workers with lower r_w as outliers. One approach is to remove all the outliers. For instance, the four workers with lowest r_w are considered outliers and are deleted, and the same weight is given to the remaining six workers. The result of comparative evaluation based on removing outliers is shown in Table 3.

In the computation above, an arbitrary border was defined between outliers and other workers as a decision boundary for removing outliers. However, instead of deleting workers with lower r_w, which might still have potentially useful insights on relevance, it is rational to give a weight to all workers' judgments based on a confidence value. The PRV for each answer list of four experiments based on assigning weight r_w to each worker's evaluation, and equal weights to all meeting fragments are shown in Table 4.

Table 2: *PRV*s for AW-vs-UI and KW-vs-UI pairs

All workers and fragments with equal weights		2-choice HITs	4-choice HITs
AW-vs-UI	PRV_{AW}	30%	26%
	PRV_{UI}	70%	74%
KW-vs-UI	PRV_{KW}	45%	35%
	PRV_{UI}	55%	65%

Table 3: *PRV*s for AW-vs-UI and KW-vs-UI pairs

Six workers and fragments with equal weights		2-choice HITs	4-choice HITs
AW-vs-UI	PRV_{AW}	24%	13%
	PRV_{UI}	76%	86%
KW-vs-UI	PRV_{KW}	46%	33%
	PRV_{UI}	54%	67%

In order to show that our method is stable on different HIT designs, we used two different HIT designs for each pair as mentioned in Section 4. We show that *PRV* converges to the same value for each pair with different HIT designs. As observed in Table 4, *PRV*s of AW-vs-UI pair are not quite similar for two different HIT designs, although the answer lists are the same. In fact, we observed that, in several cases, there was no strong agreement among workers to decide which answer list is more relevant to that meeting fragment, and we consider that these are "difficult" fragments. Since the source of uncertainty is undefined, we can reduce the effect of that fragment on the comparison by giving a weight to each fragment in proportion of the difficulty of assigning RV_{ql}. The *PRV* values thus obtained for all experiments are represented in Table 5. As shown there, the *PRV*s of AW-vs-UI pair are now very similar for 2-HIT and 4-HIT tasks. Moreover, the difference between the system versions is emphasized, which indicates that the sensitivity of the comparison method has increased.

Moreover, we compare the PCC-H method with the majority voting method and the GLAD method (Generative model of Labels, Abilities, and Difficulties [19]) for estimating comparative relevance value through considering task difficulty and worker reliability parameters. We run the GLAD algorithm with the same initial values for all four experiments. The *PRV*s which are computed by majority voting, GLAD and PCC-H are shown in Table 6.

As shown in Table 6, *PRV*s which are computed by the PCC-H method for both HIT designs are very close to those of GLAD for the 4-choice HIT design. Moreover, the *PRV* values obtained by the PCC-H method for the two different HIT designs are very similar, which is less the case for majority voting and GLAD. This means that PCC-H method is able to calculate the *PRV*s independent of the exact HIT design. Moreover, the *PRV* values calculated using PCC-H are more robust since the proposed method is not dependent on initialization values, as GLAD is. Therefore, using PCC-H for measuring the reliability of workers judgments is also an appropriate method for qualification control of workers from crowdsourcing platforms.

The proposed method is also applied for comparative evaluation of SS-vs-KW search results (semantic search vs. keyword-based search). The *PRVs* are calculated by three different methods as shown in Table 7. The first method is the majority voting method which considers all the workers and fragments with the same weight. The second method assigns weights computed by PCC-H method to measure *PRVs*, the third one is the GLAD method. Therefore the SS version outperforms the KW version according to all three scores.

Table 4: *PRV*s for AW-vs-UI and KW-vs-UI pairs

All workers with different weights and parts with equal weights		2 choices HIT design	4 choices HIT design
AW-vs-UI	PRV_{AW}	24%	18%
	PRV_{UI}	76%	82%
KW-vs-UI	PRV_{KW}	33%	34%
	PRV_{UI}	67%	66%

Table 5: *PRVs* for AW-vs-UI and KW-vs-UI pairs

All workers with different weights and fragments with different weights (PCC-H method)		2-choice HITs	4-choice HITs
AW-vs-UI	PRV_{AW}	19%	15%
	PRV_{UI}	81%	85%
KW-vs-UI	PRV_{KW}	23%	26%
	PRV_{UI}	77%	74%

7. CONCLUSION AND PERSPECTIVES

In all the evaluation steps, the UI system appeared to produce more relevant recommendations than AW or KW. Using KW instead of AW improved *PRV* by 10 percent. This means that using UI, i.e. when users ask explicit queries in conversation, improves over AW or KW versions, i.e. with spontaneous recommendations. Nevertheless, KW can be used as an assistant which suggests documents based on the context of the meeting along with the UI version, that is, spontaneous recommendations can be made when no user initiates a search. Moreover, the SS version works better than the KW version, which shows the advantage of semantic search.

As for the evaluation method, PCC-H outperformed the GLAD method proposed earlier for estimating task difficulty and reliability of workers in the absence of ground truth. Based on the evaluation results, the PCC-H method is acceptable for qualification control of AMT workers or judgments, because it provides a more stable *PRV* score across different HIT designs. Moreover, PCC-H does not require any initialization.

The comparative nature of PCC-H imposes some restrictions on the evaluations that can be carried out. For instance, if N versions must be compared, this calls in theory for $N * (N - 1)/2$ comparisons, which is clearly impractical when N grows. This can be solved if *a priori* knowledge about the quality of the systems is available, to avoid redundant comparisons. Moreover, an approach to reduce the number of pairwise comparisons required from human raters proposed in [14] could be ported to our context. For

Table 6: PRVs computed by the majority voting, the GLAD, and the PCC-H methods

Methods pairs		Majority voting, GLAD, PCC-H	
		2-choice HITs	4-choice HITs
AW-vs-UI	PRV_{AW}	30%, 23%, 19%	26%, 13%, 15%
	PRV_{UI}	70%, 77%, 81%	74%, 87%, 85%
KW-vs-UI	PRV_{KW}	45%, 47%, 23%	35%, 23%, 26%
	PRV_{UI}	55%, 53%, 77%	65%, 77%, 74%

Table 7: PRVs for SS-vs-KW

Method pair		Majority voting, GLAD, PCC-H
		4-choice HITs
SS-vs-KW	PRV_{SS}	88%, 88%, 93%
	PRV_{KW}	12%, 12%, 7%

progress evaluation, a new version must be compared with the best performing previous version, looking for measurable improvement, in which case PCC-H fully answers the evaluation needs.

There are instances in which the search results of both versions are irrelevant. The goal of future work will be to reduce the number of such uncertain instances, to deal with ambiguous questions, and to improve the processing of user-directed queries by recognizing the context of the conversation. Another experiment should improve the design of simulated user queries, in order to make them more realistic.

8. ACKNOWLEDGMENTS

The authors are grateful to the Swiss National Science Foundation for its financial support under the IM2 NCCR on Interactive Multimodal Information Management (see www.im2.ch).

9. REFERENCES

[1] J. Allan, B. Croft, A. Moffat, and M. Sanderson. Frontiers, challenges and opportunities for information retrieval: Report from SWIRL 2012. *SIGIR Forum*, 46(1):2–32, 2012.

[2] O. Alonso and R. A. Baeza-Yates. Design and implementation of relevance assessments using crowdsourcing. In *Proceedings of the European Conference on Information Retrieval (ECIR)*, pages 153–164, 2011.

[3] O. Alonso and M. Lease. Crowdsourcing 101: Putting the "wisdom of the crowd" to work for you. WSDM Tutorial, 2011.

[4] O. Alonso, D. Rose, and B. Stewart. Crowdsourcing for relevance evaluation. *SIGIR Forum*, 42:9–15, 2008.

[5] J. Carletta. Assessing agreement on classification tasks: The kappa statistic. *Computational Linguistics*, 22:249–254, 1996.

[6] J. Carletta. Unleashing the killer corpus: experiences in creating the multi-everything AMI Meeting Corpus. *Language Resources and Evaluation Journal*, 41(2):181–190, 2007.

[7] G. Chittaranjan, O. Aran, and D. Gatica-Perez. Exploiting observers' judgments for nonverbal group interaction analysis. In *Proceedings of the IEEE Conference on Automatic Face and Gesture Recognition (FG)*, 2011.

[8] P. N. Garner, J. Dines, T. Hain, A. El Hannani, M. Karafiat, D. Korchagin, M. Lincoln, V. Wan, and L. Zhang. Real-time ASR from meetings. In *Proceedings of Interspeech*, pages 2119–2122, 2009.

[9] C. Grady and M. Lease. Crowdsourcing document relevance assessment with mechanical turk. In *Proceedings of the NAACL-HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, pages 172–179, 2010.

[10] D. R. Karger, S. Oh, and D. Shah. Budget-optimal crowdsourcing using lowrank matrix approximations. In *Proceedings of the Allerton Conference on Communication, Control and Computing*, 2011.

[11] G. Kazai. In search of quality in crowdsourcing for search engine evaluation. In *Proceedings of the European Conference on Information Retrieval (ECIR)*, pages 165–176, 2011.

[12] F. K. Khattak and A. Salleb-Aouissi. Quality control of crowd labeling through expert evaluation. In *Proceedings of the NIPS 2nd Workshop on Computational Social Science and the Wisdom of Crowds*, 2011.

[13] J. Le, A. Edmonds, V. Hester, and L. Biewald. Ensuring quality in crowdsourced search relevance evaluation : The effects of training question distribution. In *Proceedings of the SIGIR 2010 Workshop on Crowdsourcing for Search Evaluation*, pages 17–20, 2010.

[14] X. Llorà, K. Sastry, D.E. Goldberg, A. Gupta, and L. Lakshmi. Combating user fatigue in iGAs: Partial ordering, support vector machines, and synthetic fitness. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO '05)*, pages 1363–1370, 2005.

[15] A. Popescu-Belis, E. Boertjes, J. Kilgour, P. Poller, S. Castronovo, T. Wilson, A. Jaimes, and J. Carletta. The AMIDA automatic content linking device: Just-in-time document retrieval in meetings. In *Proceedings of Machine Learning for Multimodal Interaction (MLMI)*, pages 272–283, 2008.

[16] A. Popescu-Belis, M. Yazdani, A. Nanchen, and P. Garner. A speech-based just-in-time retrieval system using semantic search. In *Proceedings of the 49th Annual Meeting of the ACL*, pages 80–85, 2011.

[17] P. Smyth, U. M. Fayyad, M. C. Burl, P. Perona, and P. Baldi. Inferring ground truth from subjective labeling of venus images. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1085–1092, 1994.

[18] P. Thomas and D. Hawking. Evaluation by comparing result sets in context. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 94–101, 2006.

[19] J. Whitehill, P. Ruvolo, T.-F. Wu, J. Bergsma, and J. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2035–2043. 2009.

Recommender Systems Evaluation: A 3D Benchmark

Alan Said
TU Berlin
alan@dai-lab.de

Domonkos Tikk
Gravity R&D
domonkos.tikk@gravityrd.com

Yue Shi
TU-Delft
y.shi@tudeft.nl

Martha Larson
TU-Delft
m.a.larson@tudelft.nl

Klara Stumpf
Gravity R&D
klara@gravityrd.com

Paolo Cremonesi
Politecnico di Milano
paolo.cremonesi@polimi.it

ABSTRACT

Recommender systems add value to vast content resources by matching users with items of interest. In recent years, immense progress has been made in recommendation techniques. The evaluation of these has however not been matched and is threatening to impede the further development of recommender systems. In this paper we propose an approach that addresses this impasse by formulating a novel evaluation concept adopting aspects from recommender systems research and industry. Our model can express the quality of a recommender algorithm from three perspectives, the end consumer (user), the service provider and the vendor (business and technique for both). We review current benchmarking activities and point out their shortcomings, which are addressed by our model. We also explain how our 3D benchmarking framework would apply to a specific use case.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - Retrieval models

1. INTRODUCTION & MOTIVATION

Recommender systems identify items suitable for specific users in large content collections. Despite recent commercial and research efforts, a systematic evaluation model that addresses and considers all aspects and participants of the recommender system is still missing. In this paper we propose a *3D Recommender System Benchmarking Model* that covers all dimensions that impact the effectiveness of recommender systems in real-world settings. The concept builds on a study of benchmarking settings from research and industry and provides a common comparison of recommender systems, independent of setting, data and purpose. Our benchmarking concept captures three evaluation aspects which are shared in all recommender systems, independent of whether they are research systems or industrial products. As three main evaluation dimensions we identify *user requirements*, *business requirements* and *technological constraints*, each represented by a set of qualities which ensure the general applicability of these procedures. For each particular recommendation problem, the instantiation and relevance of these requirements should be specified.

The motivation behind this framework is the growing importance of recommender systems. Users cannot be assumed to have the necessary overview to specify their information needs in vast content collections. However, with a variety of data and the recommendation task, the comparison of algorithms, approaches and general concepts becomes infeasible due to the inherent differences in requirements, design choices, etc. This calls for a comprehensive benchmarking framework that sets data- and task-specific requirements driven by particular real-world applications.

The benefits of benchmarking. Benchmarks formulate standardized tasks making it possible to compare the performance of algorithms. They have been highly successful in the areas of information retrieval, e.g. Text Retrieval Conference (TREC) [12] and the multimedia retrieval ImageCLEF [7], TRECVid [9] and MediaEval [6]. Benchmarks yield two types of benefits; (1) they serve to support the development of new technologies in the research community [9, 11] and (2) they create economic impact by bringing research closer to the market [8].

Existing recommendation benchmarks. Today's benchmarks are limited by their simplified views of users and of data. The problem setting of the Netflix Prize[1], groundbreaking at its time, was focused on a single *functional requirement*: the qualitative assessment of recommendation was simplified to the root mean squared error of predicted ratings. Its simplified view treated users as needing no further output from the recommender system than a rating on individual items. The data set was equally restricted to user ratings, additional information available in a real-world recommender system environment were not considered. Furthermore, the Prize did not take *non-functional requirements* into account, which arise from business goals and technical parameters of the recommendation service, though aspects as *scalability*, *reactivity*, *robustness* and *adaptability* are key for the productive operation of recommender systems.

The series of context-aware movie recommendation (CAMRa) challenges explored the usefulness of contextual data in recommendations. The 2010 challenge[2] provided special features on the movie mood, movie location, and intended audience (Moviepilot track), as well as social relationship between users and user activities on a movie-related social site (Filmtipset track). The time of the recommendation was also considered as context (Week track). Although the challenges expanded the data sources used, the evaluation translated real-world user needs into the classification accuracy metrics to evaluate the system in the contest, and non-functional requirements of the solutions were not investigated.

The limitations of the Netflix Prize and CAMRa series are characteristics of currently existing benchmarks and data sets. The concept presented in this paper approaches this

Copyright is held by the author/owner(s).
Workshop on Recommendation Utility Evaluation: Beyond RMSE (RUE 2012), held in conjunction with ACM RecSys 2012 September 9, 2012, Dublin, Ireland

[1] http://www.netflixprize.com
[2] http://www.dai-labor.de/camra2010/challenge/

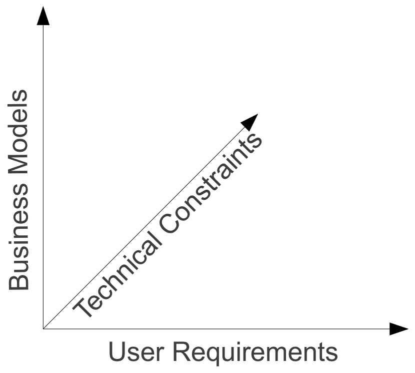


Figure 1: The three proposed evaluation axes.

challenge by placing central focus on real-world user needs; large, heterogeneous, multi-source data sets and evaluating both *functional* (quality-related) and *non-functional* (technical and business goals-related) requirements.

2. 3D RECOMMENDATION EVALUATION

In order to extend the state of the art of evaluation, we propose a concept for evaluation metrics that incorporates the needs from all perspectives in the recommendation spectrum. The concept defines a set of benchmarking techniques that select the correct combination of (*i*) data sets, (*ii*) evaluation methods and (*iii*) metrics according to a three dimensional requirement space: *business models*, *user requirements* and *technical constraints*, see Fig. 1.

Business models allow a company to generate revenue. Different models lead to different requirements in terms of the expected value from a recommender system. For instance, in a pay-per-view video-on-demand business model, the goal of the recommender system is to increase sales to allow the company to maximize revenues. However, in subscriber-based video-on-demand business models, the driving forces may be to get users to return to the service in the future (a typical showcase where recommender systems help [1]). Business models may be influenced by the choice of the objective function in the recommender algorithm; prediction-based or ranking-based functions reflect different business metrics.

User requirements reflect users' perspectives. Recommenders are assets for user satisfaction and persuasion, i.e., they try to influence a user's attitude or behavior [4], the usability of the systems affect the user's perception of the system. Recommendations may have different goals, e.g. reduce information overload, facilitate search, and find interesting items increasing the quality and decreasing the time of the decision-making process.

Technical constraints. Recommender systems in real-life must take into account a number of technical requirements and constraints. These can be classified as *data* and *system constraints*, *scalability* and *robustness* requirements. **Data constraints** relate to the service architecture, e.g. satellite TV lacks a return channel for feedback, hindering the use of collaborative filtering algorithms. **System constraints** derive from hardware and/or software limitations, e.g. in a mobile TV scenario, the processing power in the hand-held device is limited; excluding resource-heavy algorithms on the client side. **Scalability requirements** derive from the need of instant recommendations to all users on all items. These requirements are particularly strict in linear TV, where viewers are used to quick responsiveness. **Robustness requirements** are needed to create good services, able to work in case of data or component failure in distributed systems.

3. EVALUATION SETTING

3.1 Current evaluation methodologies

Existing evaluation methods for recommender systems can be classified into *system-oriented evaluation*, *user-oriented evaluation* or a combination of both [3].

In **system-oriented evaluation** (*off-line*) users are not involved in the evaluation, instead, a data set is partitioned into training and test sets. Using the training set, data points in the test set are predicted. In **user-oriented evaluation** (*on-line*) feedback from users interacting with the system is collected by explicit questions or implicit observing.

Competitions and challenges built around recommender systems are mostly organized to find the most accurate models. As described in Table 1, recommender systems are mostly evaluated off-line and often, the business value of the technologies is not examined. Even though the accuracy may influence user satisfaction and revenue increase indirectly, there exists no way to evaluate the dimensions of user requirements and business models. In most of the cases, the off-line evaluation scheme is chosen. Algorithms are often evaluated by error, ranking or classification accuracy measures. Many challenges (e.g. Netflix Prize) use explicit ratings to profile users, other recommender scenarios (e.g. item-2-item recommendation) are not addressed. Technical constraints are uncommon in contests, the exception being the RecLab Prize[3]. If a certain method performs well on a data set, the integrability in a real-world system is still not addressed. This deficiency is partially solved by online testing methods (as seen in CAMRa): recommender systems were tested in a real environment, but an objective metric to show the real applicability of the tested system is missing. In the RecLab Prize, the evaluated metric is revenue increase generated by the system. The organizers also specified non-functional requirements to be eligible for the semi-final (top 10 teams), but user requirements are not considered. These approaches all contain metrics and methods moving towards our 3D model, but none of them provide a comprehensive model.

3.2 Currently existing metrics

On-line evaluation is the only technique able to measure the true user satisfaction; conducting such evaluations is however time consuming, and cannot be generally applied, rather only to limited scenarios [2]. Contrary, off-line testing has the advantage to be immediate, and easy to perform on several data sets with multiple algorithms. The question is whether differences between the off-line performance of algorithms can be carried over to differentiate their online performance in various recommendation situations.

Classification metrics measure how well a system is able to classify items correctly, e.g. precision and recall. **Predictive metrics** measure to what extent a system can predict ratings of users. As rated items have an order, predictive accuracy metrics can be used to measure the item ranking ability. **Coverage metrics** measure the percentage of items for which the system can make recommendations [13]. **Confidence metrics** measure how certain the system is of the accuracy of the recommendations. Additionally, many recommender systems algorithms use **learning rate metrics** in order to gradually increase quality.

A recommender system can recommend accurate items, have good coverage and diversity and still not satisfy a user, if they are trivial [10]. The state-of-the-art of the evaluation metrics of recommendation reflects different recommendation tasks. **Diversity**, **novelty**, **serendipity** and **user**

[3]`http://overstockreclabprize.com/`

Table 1: An overview of some recommender system-related contests from the perspective of our 3D evaluation

Challenge	Task(s)	Metric	Mode	User	Business	Technical
Netflix Prize	minimize rating prediction error	RMSE	off-line	indirect: error measure	not addressed	not addressed
KDD-Cup'07	**1**: predict who rated what **2**: predict number of ratings	RMSE	off-line	not addressed	detect trends & popular items	not addressed
RecLab Prize	Increase revenue	revenue lift	online & off-line	not addressed	revenue lift	response/learning time, scalability
KDD-Cup'11	minimize rating prediction error split popular/unpopular items	RMSE ErrorRate	off-line	indirect: error measure find interesting or irrelevant items	not addressed	not addressed
KDD-Cup'12	prediction followed users click trough rate prediction	MAP@3 MAE, AUC	off-line	exploring interesting users & sources	not addressed ad targeting (CTR)	not addressed
CAMRa'10	context-aware; **1**: temporal, **2**: emotional, **3**: social	MAP, P@N, AUC	off-line & online	contextual information influences preference	not addressed	not addressed
CAMRa'11	group recommendation rater identification	ErrorRate	off-line	group & target recommendation	indirect: satisfaction	not addressed
CAMRa'12	find users for specific items	impact	on-line	split interesting and irrelevant content	increase audience	not addressed

satisfaction are especially difficult to measure off-line. Diversity is important for the usefulness of a recommendation and therefore there is a need to define an intra-list similarity metric [13]. Novelty and serendipity are two dimensions of non-obviousness [3].

3.3 Possible Extensions of Methods & Metrics

Real-world recommender systems should satisfy (*1*) functional requirements that relate to qualitative assessment of recommendations and (*2*) non-functional requirements specified by the technological parameters and business goals of the service. Functional and non-functional requirements should be evaluated together: without the ability to provide accurate recommendations, no recommender system can be valuable. As poor quality has adverse effects on customers, it will not serve the business goal. Similarly, if the recommender does not scale with a service, not being able to provide recommendation in real time, neither users nor service provider benefit from it. Thus, a trade-off between these requirements is needed for an impartial and comprehensive evaluation of real-world recommenders. Scalable recommenders provide good quality recommendations independently of the data size, growth and dynamic. They are able to (*1*) process huge volumes of data during initialization using computation resources linearly scalable with data size; and (*2*) serve large amounts of parallel recommendation requests in real time without significant degradation in service quality. In our model, scalability is found on the technical requirement axis.

Reactivity ensures good recommendations in real-time where the time threshold depends on the use case, typically in the range of 10–1000 ms. Adaptability is important to react for changes in user preferences, content availability and contextual parameters. In our 3D model, reactivity and adaptability belong to the user requirement axis.

Robustness is needed to handle partial, missing or corrupted data both in the system initialization and operational phases. Robustness belongs to the business axis of our model.

Generally speaking, none of the requirements are mutually exclusive, instead, optimization should be based on a combination of them – adapted for the setting in which the recommender system will be deployed [5].

This example of a Video-on-Demand (VoD) service from the IPTV industry serves as a potential scenario for our model. Business goals include increased VoD sales and customer retention, but may have additional aspects (promoting content). The technical constraints are partly specified by the middleware and the hardware/software configuration of the service provider, these all influence the response time of the service which is crucial. Via the service interface, the user gets recommendations based on the context, which might be translated into different recommendation tasks. From a user perspective, easy content exploration and context dependent recommendation may be the most important aspects.

4. CONCLUSION

We proposed a 3D Recommender System Benchmarking model that extends the state-of-the-art and addresses both functional and non-functional real-word application-driven aspects of recommender systems. Following the proposed concept, the benchmarking activities within the community will encompass the full range of other recommender system use cases and algorithmic approaches. The comprehensive evaluation methodology will boost the development of more effective recommender systems, and make it possible to focus research resources productively and for industry technology providers to increase the uptake of recommender technology.

5. REFERENCES

[1] M. B. Dias, D. Locher, M. Li, W. El-Deredy, and P. J. Lisboa. The value of personalised recommender systems to e-business: a case study. In *RecSys '08*. ACM, 2008.

[2] M. Gorgoglione, U. Panniello, and A. Tuzhilin. The effect of context-aware recommendations on customer purchasing behavior and trust. In *RecSys '11*, pages 85–92. ACM, 2011.

[3] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1), 2004.

[4] R. Hu. Design and user issues in personality-based recommender systems. In *RecSys '10*. ACM, 2010.

[5] T. Jambor and J. Wang. Optimizing multiple objectives in collaborative filtering. In *RecSys '10*. ACM, 2010.

[6] M. Larson, M. Soleymani, P. Serdyukov, S. Rudinac, C. Wartena, V. Murdock, G. Friedland, R. Ordelman, and G. J. F. Jones. Automatic tagging and geotagging in video collections and communities. In *ICMR '11*. ACM, 2011.

[7] H. Müller. *ImageCLEF experimental evaluation in visual information retrieval.* Springer, Heidelberg, 2010.

[8] B. Rowe, D. Wood, A. Link, and D. Simoni. Economic impact assessment of NIST's text retrieval conference (TREC) program. Technical report, July 2010.

[9] A. Smeaton, P. Over, and W. Kraaij. Evaluation campaigns and TRECVid. In *MIR '06*, 2006.

[10] L. Terveen and W. Hill. Beyond recommender systems: Helping people help each other. In *HCI in the New Millennium.* Addison-Wesley, 2001.

[11] T. Tsikrika, J. Kludas, and A. Popescu. Building reliable and reusable test collections for image retrieval: The Wikipedia Task at ImageCLEF. *IEEE Multimedia*, 99(PrePrints), 2012.

[12] E. M. Voorhees. Overview of TREC 2005. In *TREC*, 2005.

[13] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW '05*. ACM, 2005.

On the use of Weighted Mean Absolute Error in Recommender Systems

S. Cleger-Tamayo
Dpto. de Informática.
Universidad de Holguín, Cuba
sergio@facinf.uho.edu.cu

J.M. Fernández-Luna & J.F. Huete
Dpto. de Ciencias de la Computación e I.A.
CITIC – UGR Universidad de Granada, Spain
{jmfluna,jhg}@decsai.ugr.es

ABSTRACT

The classical strategy to evaluate the performance of a Recommender System is to measure the error in rating predictions. But when focusing on a particular dimension in a recommending process it is reasonable to assume that every prediction should not be treated equally, its importance depends on the degree to which the predicted item matches the deemed dimension or feature. In this paper we shall explore the use of weighted Mean Average Error (wMAE) as an alternative to capture and measure their effects on the recommendations. In order to illustrate our approach two different dimensions are considered, one item-dependent and the other that depends on the user preferences.

1. INTRODUCTION

Several algorithms based on different ideas and concepts have been developed to compute recommendations and, as a consequence, several metrics can be used to measure the performance of the system. In the last years, increasing efforts have been devoted to the research of Recommender System (RS) evaluation. According to [2], "the decision on the proper evaluation metric is often critical, as each metric may favor a different algorithm". The selected metric depends on the particular recommendation tasks to be analyzed. Two main tasks might be considered: the first one, with the objective of measuring the capability of a RS to predict the rating that a user should give to an unobserved item, and the second one is related to the ability of an RS to rank a set of unobserved items, in such a way that those items more relevant to the user have to be placed in top positions of the ranking. Our interest in this paper is the measurement of the capability of a system to predict user interest in an unobserved item, so we focus on *rating prediction.*

For this purpose, two standard metrics [3, 2] have been traditionally considered: the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE). Both metrics try to measure which might be the expected error of the system, RMSE being more sensitive to the occasional large error: the squaring process gives higher weight to very large errors. A valuable property of both metrics is that they take their values in the same range as the error being estimated, so they can be easily understood by the users.

But, these metrics consider that the standard deviation of the error term is constant over all the predictions, i.e. each prediction provides equally precise information about the error variation. This assumption, however, does not hold, even approximately, in every recommending application. In this paper we will focus on the weighted Mean Absolute Error, wMAE, as an alternative to measure the impact of a given feature in the recommendations[1]. Two are the main purposes for using this metric: On the one hand, as an enhanced evaluation tool for better assessing the RS performance with respect to the goals of the application. For example, in the case of recommending books or movies it could be possible that the accuracy of the predictions varies when focusing on past or recent products. In this situation, it is not reasonable that every error were treated equally, so more stress should be put in recent items. On the other hand, it can be also useful as a diagnosis tool that, using a "magnifying lens", can help to identify those cases where an algorithm is having trouble with. For both purposes, different features shall be considered which might depend on the items, as for example, in the case of a movie-based RS the genre, the release date, price, etc. But also, the dimension might be user-dependent considering, for example, the location of the user, the users' rating distribution, etc.

This metric has been widely used for evaluation of model performance in several fields as meteorology or economic forecasting [8]. But, few have been discussed about its use in the recommending field; isolately several papers use small tweaks on error metrics in order to explore different aspects of RS [5, 7]. Next section presents the weighted mean absolute error, illustrating its performance considering two different features, user and item-dependent, respectively. Lastly we present the concluding remarks.

2. WEIGHTED MEAN ABSOLUTE ERROR

The objective of this paper is to study the use of a weighting factor in the average error. In order to illustrate its functionality we will consider simple examples obtained using four different collaborative-based RS (using Mahout implementations): i) Means, that predicts using the average rat-

Acknowledgements: This work was jointly supported by the Spanish Ministerio de Educación y Ciencia and Junta de Andalucía, under projects TIN2011-28538-C02-02 and Excellence Project TIC-04526, respectively, as well as the Spanish AECID fellowship program.

[1]A similar reasoning can be used when considering squared error, which yields to the weighted Mean Root Squared Error, wRMSE.

ings for each user; ii) LM [1], following a nearest neighbors approach; iii) SlopeOne [4], predicting based on the average difference between preferences and iv) SVD [6], based on a matrix factorization technique. The metric performance is showed using an empirical evaluation based on the classic MovieLens 100K data set.

A weighting factor would indicate the subjective importance we wish to place on each prediction, relating the error to any feature that might be relevant from both, the user or the seller point of view. For instance, considering the release date, we can assign weights in such a way that the higher the weight, the higher importance we are placing on more recent data. In this case we could observe that even when the MAE is under reasonable threshold, the performance of a system might be inadequate when analyzing this particular feature.

The *weighted Mean Absolute Error* can be computed as

$$wMAE = \frac{\sum_{i=1}^{U} \sum_{j=1}^{N_i} w_{i,j} \times abs(p_{i,j} - r_{i,j})}{\sum_{i=1}^{U} \sum_{j=1}^{N_i} w_{i,j}}, \qquad (1)$$

where U represents the number of users; N_i, the number of items predicted for the i^{th}-user; $r_{i,j}$, the rating given by the i^{th}-user to the item I_j; $p_{i,j}$, the rating predicted by the model and $w_{i,j}$ represents the weight associated to this prediction. Note that when all the individual differences are weighted equally $wMAE$ coincides with MAE.

In order to illustrate our approach, we shall consider two factors, assuming that $w_{i,j} \in [0, 1]$.

• **Item popularity**: we would like to investigate whether the error in the predictions depends on the number of users who rated the items. Two alternatives will be considered:

i+ The weights will put more of a penalty on bad predictions when an item has been rated quite frequently (the items has a high number of ratings). We still penalize bad predictions when it has a small number of ratings, but we do not penalize as much as when we have more samples, since it may just be that the limited number of ratings do not provide much information about the latent factors which influence the users ratings. Particularly, for each item I_i we shall consider its weight as the probability that this item were rated in the training set, i.e. $w_i = pr(I_i)$.

i– This is the inverse of the previous criterion, where we put more emphasis on the predictions over those items with fewer ratings. So the weights are $w_i = 1 - pr(I_i)$.

• **Rating distribution**: It is well known that the users does not rate the items uniformly, they tend to use high-valued ratings. By means of this feature we can measure whether the error depends on the ratings distribution or not. Particularly, we shall consider four different alternatives:

rS+ Considering the overall rating distribution in the system, putting more emphasis on the error in the predictions on those common ratings. So the weights are $w_i = pr_S(r_i)$, r_i being the rating given by the user to the item I_i.

rS– Inversely, we assess more weight to the less common ratings, i.e. $w_i = 1 - pr_S(r_i)$.

rU+ Different users can use a different pattern of rating, so we consider the rating distribution of the user, in such a way that those common ratings for a particular user will have greater weights, i.e. $w_i = pr_U(r_i)$.

rU– The last one assigns more weight to the less frequent rating, i.e. $w_i = 1 - pr_U(r_i)$.

Figures 1-A and 1-B present the absolute values of the MAE and wMAE error for the four RSs considered in this paper. Figure 1-A shows the results where the weights are positively correlated to the feature distribution, whereas Figure 1-B presents the results when they are negatively correlated. In this case, we can observe that by using wMAE we can determine that error is highly dependent on the users' pattern of ratings, and weaker when considering item popularity. Moreover, if we compare the two figures we can observe that all the models perform better when predicting the most common ratings. In this sense, they are able to learn the most frequent preferences and greater errors (bad performance) are obtained when focusing on less frequent rating values. Related to item popularity these differences are less conclusive. In some sense, the way in which the user rates an item does not depend of how popular the item is.

2.1 Relative Weights vs. Relative Error

Another different alternative to explore the benefits of using the wMAE metric is to consider the ratio between wMAE and MAE. In this sense, denoting as $e_{i,j} = abs(p_{i,j} - r_{i,j})$, we have that wMAE/MAE is equal to

$$wMAE/MAE = \frac{\sum_{i,j} w_{i,j} e_{i,j} / \sum_{i,j} e_{i,j}}{\sum_{i,j} w_{i,j} / N}.$$

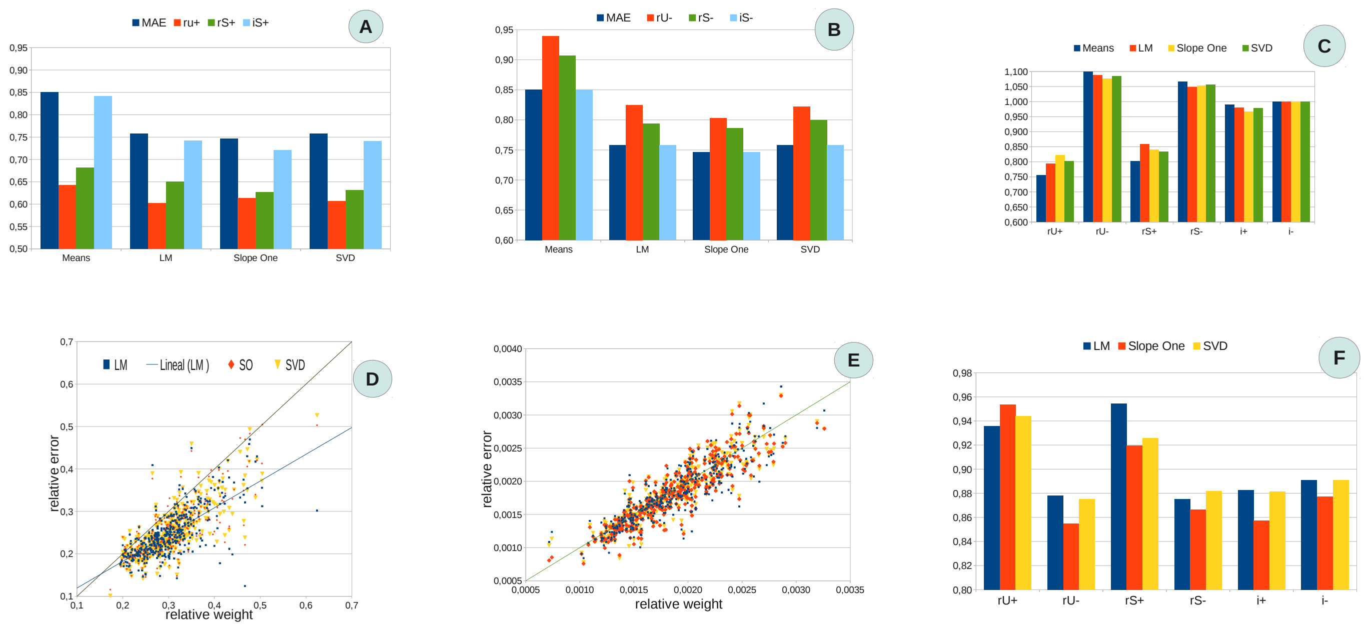
Taking into account that we restrict the weights to take its value in the $[0, 1]$ interval, the denominator might represent the average percentage of mass of the items that is related to the dimension under consideration whereas the numerator represents the average percentage of the error coming from this feature. So, when wMAE $>$ MAE we have that the percentage of error coming from the feature is greater than its associated mass, so the the system is not able to predict properly such dimension. When both metrics are equal this implies that the expected error is independent of the feature.

In Figure 1-C we present the values of the wMAE/MAE where, again, we can see that there exists a dependence between the rating and the error. The error associated to the common ratings are less than the relative importance of this feature in the system whereas for less common ratings the system is not able to perform good predictions, being greater the relative error than its associated weights. This situation does not hold when considering item popularity.

Figures 1-D and 1-E present an scatter plot that relates the relative weights (horizontal axis) to the relative error (vertical axis) for each user in the system and for each RS used[2]. Particularly, in Figure 1-D we are considering rU+ as weighting factor. Note that, since there are 5 possible ratings, the relative weight is equal to 0.2 when all the ratings are equally probable and its value increases with the importance of the most used ratings. In this figure we can see that both percentage of mass and the percentage of error are positively correlated, being wMAE/MAE $<$ 1 for most of the users. Moreover, there is a trend to improve the predictions for those users with higher relative mass (for example, we can see how the regression line for the LM model

[2]We have included all the users with at least 10 predictions.

Figure 1: Using wMAE in recommendations: absolute and relative values.



gets further away[3] from the line y=x). In some way we can conclude that recommendation usefulness of the rating distribution is consistent for all the users and RS models. On the other hand, Figure 1-E considers i+ as weights. In this case, although weights and error are positively correlated, there exists significant differences between different users. This result is hidden in the global measures.

2.2 Relative Comparison Among Models

Although wMAE might give some information about how the error has been obtained, there is no criterion about what a good prediction is. In order to tackle this situation we propose the use of the relative rather than the absolute error, i.e. the weighted Mean Absolute Percentage Error, wMAPE. Then, given two models, $M1$ and $M2$, the relative metric is defined as $wMAE_{M1}/wMAE_{M2}$. In this metric, the less the value, the greater the improvements. Thus, if we fix the model M2 to be a simple model (as the average rating) we obtain the wMAE values in Figure 1-F. From these values, we can obtain some conclusions as for instance that LM fits better the common user's preferences (rU+), whereas Slope One and SVD are more biased toward the overall rating distribution in the system (rS+). Similarly, we found that better improvements, with respect to the average ratings, are obtained when focusing on less frequent ratings. Finally, with respect to item popularity all the models obtain better improvements when considering the most popular items, although these differences are less significant.

3. CONCLUSIONS

In this paper we have explored the use of weighted Mean Average Error as a means to measure the RS's performance by focusing on a given dimension or feature, being able to uncover specific cases where a recommendation algorithm may be having suboptimal performance. This is a very useful way to know the origin of the errors found in the recommendations and therefore useful for improving the RSs, although its main problem is that it is not absolute as MAE.

[3]The other models perform similarly, but we have decided to not include these regression lines due to clarity reasons.

4. REFERENCES

[1] S. Cleger-Tamayo, J.M. Fernández-Luna and J.F. Huete. *A New Criteria for Selecting Neighborhood in Memory-Based Recommender Systems.* Proc. of 14th CAEPIA'11, pp. 423-432. 2011.

[2] A. Gunawardana and G. Shani. *A Survey of Accuracy Evaluation Metrics of Recommendation Tasks.* Journal of Machine Learning Research 10, pp. 2935-2962. 2009.

[3] J.L. Herlocker, J.A. Konstan, L.G. Terveen and J.T. Riedl. *Evaluating collaborative filtering recommender systems.* ACM Trans. Inf. Syst. 22, 1. 2004, pp. 5-53.

[4] D. Lemire and A. Maclachlan. Slope One Predictors for Online Rating-Based Collaborative Filtering. Proc. of SIAM Data Mining (SDM'05), 2005

[5] P. Massa and P. Avesani. Trust metrics in recommender systems. Computing with Social Trust, pp. 259-285 Springer 2009.

[6] B.M. Sarwar, G. Karypis, J. Konstan and J. Riedl. Incremental SVD-Based Algorithms for Highly Scaleable Recommender Systems. 5th International Conf. on Computer and Information Technology. 2002.

[7] T. Jambor and J. Wang. Goal-driven collaborative filtering: A directional error based approach. In Proc. ECIR'2010. pp. 407-419. 2010.

[8] C.J. Willmot. Statistics for the Evaluation and Comparison of Models. Journal of Geophysical Research, 90. pp. 8995-9005, 1985.

How Similar is Rating Similarity to Content Similarity?

Osman Başkaya
Department of Computer Engineering
Bahçeşehir University
İstanbul, Turkey
osman.baskaya@computer.org

Tevfik Aytekin
Department of Computer Engineering
Bahçeşehir University
İstanbul, Turkey
tevfik.aytekin@bahcesehir.edu.tr

ABSTRACT

The success of a recommendation algorithm is typically measured by its ability to predict rating values of items. Although accuracy in rating value prediction is an important property of a recommendation algorithm there are other properties of recommendation algorithms which are important for user satisfaction. One such property is the diversity of recommendations. It has been recognized that being able to recommend a diverse set of items plays an important role in user satisfaction. One convenient approach for diversification is to use the rating patterns of items. However, in what sense the resulting lists will be diversified is not clear. In order to assess this we explore the relationship between rating similarity and content similarity of items. We discuss the experimental results and the possible implications of our findings.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Experimentation, Measurement

Keywords

diversity, recommender systems, collaborative filtering

1. INTRODUCTION

Recommender systems help users to pick items of interest based on explicit or implicit information that users provide to the system. One of the most successful and widely used technique in recommender systems is called collaborative filtering (CF) [7]. CF algorithms try to predict the ratings of a user based on the ratings of that user and the ratings of other users in the system. The performance of collaborative filtering algorithms is typically measured by the error they make in predicting the ratings of users for items. Although accuracy of predictions is an important aspect of recommender systems, it is not the only one. Recently, increasing the diversity of recommendation lists have gained attention among researchers in the field [8, 2]. To be able to recommend a diverse set of items to a user is important with respect to user satisfiability because a recommendation list consisting of one type of item (e.g., movies only from the same genre) might not be very satisfactory even if the accuracy of rating prediction is high. But here there is one issue. We need to define a metric for measuring the diversity of a recommendation list. Then we can try to optimize the recommendation list based on this metric. One possible metric for measuring the diversity of a recommendation list of a particular user is described in [2]. This metric measures the diversity as the average dissimilarity of all pairs of items in a user's recommendation list. Formally, it can be defined as follows:

$$D(R) = \frac{1}{N(N-1)} \sum_{i \in R} \sum_{j \in R, j \neq i} d(i,j), \qquad (1)$$

where R is the recommendation list of a user and $N = |R|$. $d(i,j)$ is the dissimilarity of items i and j which is defined as one minus the similarity of items i and j.

We think that average dissimilarity is a reasonable way to measure the diversity of a list of items. However, the important part is how to define $d(i,j)$, i.e., the dissimilarity of two items which is unspecified in equation (1). The problem is not to choose a similarity metric such as Pearson or cosine. The problem is whether we can use the rating patterns (vectors) of items in order to measure their similarity. And if we use these rating patterns, in what respect the recommendation lists will be diversified? For example, if it is a movie recommender system, will the recommendation lists contain more movies from different genres or will the content of the movies get diversified?

In order to answer these questions we will compare rating similarity with two types of content similarities which we will define below. We hope that the results we discuss will shed some light on these types of questions and stimulate discussion on diversification.

2. RELATED WORKS

In hybrid recommendations content information is used in order to increase the accuracy of rating predictions especially for items whose ratings are too sparse. For example [3, 5, 6] use content information collected from sources such as

Wikipedia and IMDB in order to improve the accuracy of rating predictions. These works indirectly show that there is indeed some positive relationship between rating similarity and content similarity. Otherwise, it was not possible to increase the prediction accuracy using content information.

Another paper which comes close to our concerns is [1] Here, the authors propose a new algorithm for diversifying recommendation lists. Their algorithm uses rating patterns of movies for diversification. They evaluate the results by looking at how well the recommendation lists are diversified with respect to genre and movie series they belong. They report that the resulting lists' diversity increase in both respects (genre and series). However, to the best of our knowledge there are no direct comparisons between rating and content similarity. In this paper we examine directly these two types of similarities.

3. ITEM CONTENT GENERATION

In our experiments we use Movielens[1] (1M) data set. In order to compare movies' rating patterns to their contents we first need to generate movie content information. We use two sources of information to this end. One source of content information comes from Wikipedia articles corresponding to movies in the Movielens dataset. The other source of content information comes from genre information which are provided in the dataset. Details of content generation are given below.

3.1 Content Generation from Wikipedia

The Movielens dataset contains 3883 distinct movies and 6040 users. Some of these movies are not rated by any user. Also some of the movies have no corresponding entries in Wikipedia. After discarding these movies we are able to fetch 3417 (approximately 88% of all movies) movie articles from Wikipedia.

In this work we only use the text of each Wikipedia article (we do not use link structure or category information of articles). The text of a Wikipedia article consists of parts such as "Plot", "Cast", and "Release". We do not include "References" and "See also" parts of the text since they may contain information which is unrelated to the content of the movies. After extracting the text of each document we apply some basic preprocessing steps such as stemming and stopwords removal. We use a vector space model to represent text documents.

3.2 Genre Information

As a second source of content we use the genre keywords (such as adventure, action, comedy, etc.) provided by the Movielens dataset. Each movie in the dataset is associated with one or more genre keywords. We define the genre similarity between two movies using the Jaccard metric given below:

$$J(i,j) = \frac{|G_i \cap G_j|}{|G_i \cup G_j|} \qquad (2)$$

where G_i and G_j are genre sets of items i and j.

4. EXPERIMENTS

In the first set of experiments we try to understand the relation between movie rating patterns and content generated from the corresponding Wikipedia articles. We have two matrices: one is the Movie-User matrix which holds the ratings of users on movies and the other is the Movie-TFIDF matrix which holds the *tf-idf* weights for each document. For evaluation we use the following methodology. For each movie we find the most similar 100 movies using the Movie-User matrix (rating neighborhood) and the most similar 100 movies using Movie-TFIDF matrix (content neighborhood). We then find the number of common items in these two neighborhoods. It turns out that on average there are 14.74 common movies in the two neighborhoods. If we generate the neighborhoods randomly this value turns out to be around 2.80. Randomization tests show that this difference is significant ($p < 0.01$).

We run the same experiment with different neighborhood sizes (20 and 50) but the percentages of the number of common items in the rating and content neighborhoods turn out to be similar to the percentages we get when we use a neighborhood of size 100.

We also test whether there is a relationship between the number of ratings and the correspondence between rating and content similarity. To see this we find the rating and content neighborhoods of those movies which have similar number of ratings. To do this we divide the movies into rating intervals according to the number of ratings they have: movies which have ratings between 1-100, between 101-200, and so on. If an interval has less than 20 movies, we merge it with the previous one in order to increase the significance of the results. Figure 1 shows the average number of common items in the rating and content neighborhood sets of movies as a function of rating intervals. Interestingly, Figure 1 shows a clear linear correlation, i.e., as the number of ratings increases the number of common items in the content and rating neighborhood of movies also increases. One possible explanation of this positive linear correlation might be this. Generally, there is a positive relationship between the number of ratings and the popularity of a movie. This means that popular movies receive ratings from many different people with different tastes. Hence the rating patterns of popular movies reflect a diverse set of characteristics. Wikipedia movie articles also have rich contents reflecting different characteristics of movies. This might explain why a movie's rating neighborhood approaches to its content neighborhood as the number of ratings increase.

In the next set of experiments our aim is to understand the relationship between movie rating patterns and movie genres provided in the Movielens dataset. Genre keywords provide limited information compared to Wikipedia articles. Because Wikipedia articles contain terms that give information not only about the genre of a movie but also about the director, starring, musical composition, etc.

In order to measure the relationship between movie rating patterns and genres we applied a similar methodology. For each movie m we find the most similar 100 movies using the Movie-User matrix (that is the rating neighborhood) and find the Jaccard similarity (as defined in equation 2) between movie m and movies in its rating neighborhood. The average Jaccard similarity value turns out to be 0.43. If we generate the rating neighborhood randomly we find a Jaccard value around 0.17. Randomization tests show that this difference is significant ($p < 0.01$).

We also test whether there is a relationship between the number of ratings and genre similarity. Similar to the experiment we described above we divided the movies into rat-

[1]http://www.grouplens.org/node/73

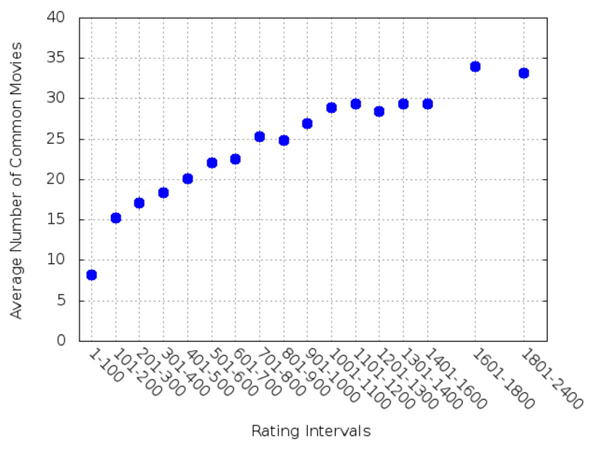


Figure 1: Average number of common movies as a function of rating intervals.

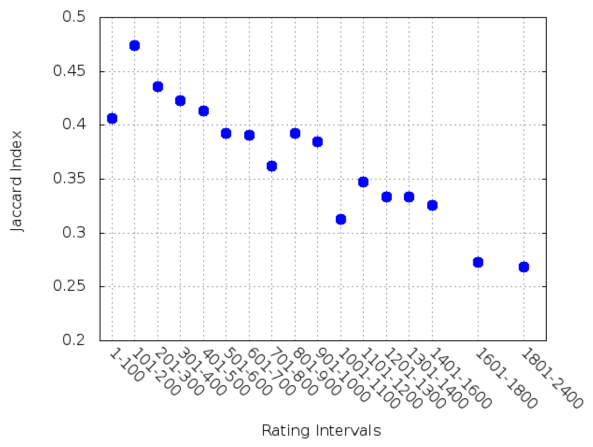


Figure 2: Average Jaccard index as a function of rating intervals.

ing intervals according to the number of ratings they have. Then for each movie m in a rating interval we calculate the Jaccard similarity value between the movie m and its rating neighborhood of 100 movies then calculate the averages per rating interval. Figure 2 shows these average values as a function of rating intervals. Here, we again have an interesting case. There is a *negative* linear correlation which means that the more a movie has ratings the more its rating similarity diverges from its genre similarity.

The reason underlying these results might be this. Movies which have limited number of ratings (unpopular movies) are generally watched by the fans of that genre. For example, a fan of sci-fi movies may also watch an unpopular sci-fi movie. So, unpopular movies generally get ratings from the same set of users who are fans of that movie's genre. And this makes the rating vectors of those movies (same genre movies) similar to each other. On the other hand if a movie is popular than it gets ratings from a diverse set of users which causes their rating neighborhoods diverge from its genre.

5. CONCLUSION

We should note at the outset that the conclusions presented here are not conclusive. Different experiments on different datasets and with different item types need to be done in order to drive more firm conclusions. However, we hope that these experiments and results will stimulate discussion and further research.

In this work we examined the relationship between rating similarity and content similarity of movies in the Movielens dataset. We examined two kinds of content: one of them is the *tf-idf* weights of movie articles in Wikipedia and the other is the genre keywords of movies provided by the Movielens dataset.

We found that to a certain degree there is a similarity between rating similarity and Wikipedia content similarity and also between rating similarity and genre similarity. However, we leave open to discussion the magnitude of these similarities. We also found that as the number of ratings of a movie increases its rating similarity approaches to its Wikipedia content similarity whereas its rating similarity diverges away from its genre similarity.

According to these results if diversification is done based on the rating patterns of movies then the recommendation lists will likely be diversified with respect to the content of movies to some extent. So, if no content information is available or it is difficult to get it, it might be useful to use rating patterns to diversify the recommendation lists.

To this analysis we plan to add latent characteristics of items generated by matrix factorization methods [4]. We plan to explore the correspondences among similarities defined over rating patterns, contents, and latent characteristics of items.

6. REFERENCES

[1] R. Boim, T. Milo, and S. Novgorodov. Diversification and refinement in collaborative filtering recommender. In *CIKM*, pages 739–744, 2011.

[2] N. Hurley and M. Zhang. Novelty and diversity in top-N recommendation - analysis and evaluation. *ACM Trans. Internet Techn*, 10(4):14, 2011.

[3] G. Katz, N. Ofek, B. Shapira, L. Rokach, and G. Shani. Using wikipedia to boost collaborative filtering techniques. In *RecSys*, pages 285–288, 2011.

[4] Y. Koren, R. M. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[5] A. Loizou and S. Dasmahapatra. *Using Wikipedia to alleviate data sparsity issues in Recommender Systems*, pages 104–111. IEEE, 2010.

[6] P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *AAAI/IAAI*, pages 187–192, 2002.

[7] J. B. Schafer, D. Frankowski, J. L. Herlocker, and S. Sen. Collaborative filtering recommender systems. In *The Adaptive Web*, pages 291–324, 2007.

[8] M. Zhang and N. Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *RecSys*, pages 123–130, 2008.

Modeling Difficulty in Recommender Systems

Benjamin Kille, Sahin Albayrak
DAI-Lab
Technische Universität Berlin
{kille,sahin}@dai-lab.de

ABSTRACT

Recommender systems have frequently been evaluated with respect to their average performance for all users. However, optimizing such recommender systems regarding those evaluation measures might provide worse results for a subset of users. Defining a difficulty measure allows us to evaluate and optimize recommender systems in a personalized fashion. We introduce an experimental setup to evaluate the eligibility of such a difficulty score. We formulate the hypothesis that provided a difficulty score recommender systems can be optimized regarding costs and performance simultaneously.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - Information filtering, Retrieval models, Search process, Selection process; H.3.4 [**Information Technology and Systems Applications**]: Decision support

General Terms

Algorithms, Design, Experimentation, Measurement, Human factors

Keywords

difficulty, recommender systems, user modeling, evaluation

1. INTRODUCTION

Evaluating a recommender system's performance represents a non-trivial task. The choice of evaluation measure and methodology depends on various factors. Modeling the recommendation task as rating prediction problem favors measures such as root mean squared error (RMSE) and mean absolute error (MAE) [7]. In contrast, mean average precision (MAP) and normalized discounted cumulative gain (NDCG) qualify as evaluation measures for an item ranking scenario [12]. In either case, two recommendation algorithms are compared with respect to their average performance on the full set of users. This entails that all users are treated equally. However, we argue that the difficulty of recommending items to users varies. Suppose we consider a recommender system with two users, Alice and Bob. Alice has rated a large number of items. Bob has recently started using the system and rated a few items. The system recommends a number of items to both of them. Alice and Bob rate the recommended items. Suppose we attempt to evaluate two recommender algorithms based on those ratings, denoted as R_1 and R_2. Assume that R_1 predicts Alice's ratings with an error of 0.8 and Bob's ratings with an error of 0.9. On the other hand, we observe R_2 to deviate 1.0 for Alice and 0.8 for Bob, respectively. Averaging the errors, we obtain 0.85 for R_1 and 0.9 for R_2. Still, R_2 predicts Bob's ratings better even though his preferences exhibit higher sparsity compared to Alice's. Besides the number of ratings, there are further factors discriminating how well an recommendation algorithm perform for a given user. We introduce the notion of difficulty in the context of recommender systems. Each user is assigned a difficulty value reflecting her expected evaluation outcome. Users with high errors or disordered item rankings receive a high difficulty value. Contrarily, we assign low difficulty values to users exhibiting low errors and well ordered item rankings. Recommender systems benefit from those difficulty value twofold. First, optimization can target difficult users who require such efforts. On the other hand, the recommender system can provide users with low difficulty values with recommendations generated by more trivial methods. Recommending most popular items represents such an approach. Second, difficulty values enable the system to estimate how likely a specific user will perceive the recommendations as adequate. Thereby, the system can control interactions with users, e.g. by asking for additional ratings to provide better recommendations for particularly difficult users.

2. RELATED WORK

Adomavicius and Tuzhilin reveal possible extensions to recommender systems [1]. They mention an enhanced understanding of the user as one such extension. Our work attempts to contribute to this by defining a user's difficulty. Hereby, the system estimates a user's likely perception of the recommendations. The methods performing best on the well-known *Netflix Prize Challenge* had applied ensemble techniques to further improve their rating prediction accuracy [3, 11]. Both do not state an explicit modeling of recommendation difficulty on the user-level. However, ensem-

Workshop on Recommendation Utitlity Evaluation: Beyond RMSE (RUE 2012), held in conjunction with ACM RecSys 2012 September 9, 2012, Dublin, Ireland.

ble techniques involve an implicit presence of such a concept. Combining the outcome of several recommendation algorithms does make sense in case a single recommendation algorithm fails for a subset of users. Such an effect is consequently compensated by including other recommendation algorithms' outcomes. Bellogin presents an approach to predict a recommender systems performance [4]. However, the evaluation is reported on the system level averaging evaluation measures over the full set of users. Our approach focuses on predicting the user-level difficulty. Koren and Sill introduced a recommendation algorithms that outputs confidence values [8]. Those confidence values could be regarded as difficulty. However, the confidence values are algorithm specific. We require the difficulty score's validity to generally hold. The concept of evaluation considering difficulty has been introduced in other domains. Aslam and Pavlu investigated estimation of a query's difficulty in the context of information retrieval [2]. Their approach bases on the diversity of retrieved lists of documents by several IR systems. Strong agreement with respect to the document ranking indicates a low level of difficulty. In contrast, highly diverse rankings suggest a high level of difficulty. He et al. describe another approach to estimate a query's difficulty [6]. Their method compares the content of retrieved documents and computes a coherence score. They assume that in case highly coherent documents are retrieved the query is clear and thus less difficult than a query resulting in minor coherency. Genetic Algorithms represent another domain where difficulty has received the research community's attention. However, the difficulty is determined by the fitness landscape of the respective problem [5, 6]. Kuncheva and Whitaker investigated diversity in the context of a classification scenario [9]. Concerning a query's difficulty, diversity has been found an appropriate measure [2]. We will adapt two diversity measures applied in the classification scenario for determining the difficulty to recommend a user items.

3. METHOD

We strive to determine the difficulty of the recommendation task for a given user. Formally, we are looking for a map $\delta(u) : \mathcal{U} \mapsto [0; 1]$ that assigns each user $u \in \mathcal{U}$ a real number between 0 and 1 corresponding to the level of difficulty. For one recommendation algorithm the difficulty for a given user could be simply determined by a (normalized) evaluation measure, e.g. RMSE. However, such a difficulty would not be valid for other recommendation algorithms. In addition, recommender systems optimized with respect to the item ranking would likely end up with different difficulty values. We adapt the idea of measuring difficulty in terms of diversity to overcome those issues. We assume the recommender systems comprises several recommendations methods, denoted $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$. Each such A_n generates rating predictions along with item rankings for a given user u. We choose RMSE to evaluate the rating predictions and NDCG to assess the item ranking, respectively. We measure a user's difficulty by means of the diversity of those rating predictions and item rankings. For that purpose, we adjusted two diversity metrics introduced by Kuncheva and Whitaker for classification ensembles [9]. We alter the pair-wise Q statistics to fit the item ranking scenario. Regarding the rating prediction we adapt the difficulty measure θ.

3.1 Q statistics

Applied to the classification ensemble setting, the Q statistics base on a confusion matrix. The confusion matrix confronts two classifiers in a binary manner - correctly classified versus incorrectly classified. We need to adjust this setting to fit the item ranking scenario. Table 1 illustrates the adjusted confusion matrix. We count all correctly ranked items as well as all incorrectly ranked items for both recommendations algorithms. Subsequently, the confusion matrix displays the overlap of those items sets. Equation 1 represents the Q statistic derived from the confusion matrix. Note that the Q statistic measures diversity in between two recommender algorithms. Hence, we need to average the Q statistic values obtained for all comparisons in between the available algorithms. Equation 2 computes the final diversity measure.

$$Q_{ij}(u) = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{10}N^{01}} \tag{1}$$

$$Q_{\mathcal{A}}(u) = \binom{|\mathcal{A}|}{2}^{-1} \sum_{A_i, A_j \in \mathcal{A}} Q_{ij} \tag{2}$$

3.2 Difficutly measure θ

Kuncheva and Whitaker introduce the difficulty measure θ as a non-pairwise measure [9]. θ allows us to consider several recommendation algorithms simultaneously. We iterate over the set of withhold items for the given user. At each step we compute the RMSE for all recommendation algorithms at hand. Thereby, we observe the variance in between all recommender algorithms' RMSE values. The average variance displays how diverse the user is perceived by the set of recommendation algorithms. Equation 3 calculates θ. $\mathcal{I}$ denotes the set of items in the target user's test set. The $\sigma(i)$ function computes the variances in between the recommendation algorithms' RMSE values for the given item.

$$\theta(u) = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \sigma(i) \tag{3}$$

3.3 Difficulty measure δ

We attempt to formulate a robust difficulty measure for recommender systems. Therefore, we propose a combination of the Q statistic and θ. As a result, both RMSE and NDCG are considered. The more recommendation algorithms we include the more robust the final difficulty value will be. Equation 4 displays a linear combination of both measures. The parameter $\lambda \in [0; 1]$ controls the weighting. $\lambda = 1$ allows to focus on the ranking task. The absence of rating values might require such a setting.

$$\delta(u|\mathcal{A}) = \lambda \cdot Q_{\mathcal{A}}(u) + (1 - \lambda) \cdot \theta(u) \tag{4}$$

4. EXPERIMENTAL SETTING

We will evaluate the proposed difficulty measure δ and subsequently outline the intended experimental protocol. Several data sets, such as *Movielens 10M, Netflix* and *Last.fm* provide the required data. We will implement item-based

	A_i: rank(k) $\leq$ rank(l)	A_i: rank(k) $>$ rank(l)
A_j: rank(k) $\leq$ rank(l)	N^{11}	N^{10}
A_j: rank(k) $>$ rank(l)	N^{01}	N^{00}

Where rank(k) $\leq$ rank(l) is true.

Table 1: Adjusted confusion matrix

and user-based neighborhood recommenders, matrix factorization recommenders along with slope-one recommenders. All those recommendation methods do not require data except user preferences. As a first step, we split each data sets in three partitions. The first partition will be used for training and the second partition to assign each user a difficulty score according to Equation 4. Finally, the third partition will be used as evaluation data set. The subset of users with comparably low difficulty score will receive recommendations based on a non-optimized recommendation method such as the most popular recommender. The subset of users with medium difficulty score will receive recommendations generated by slightly optimized recommender algorithms. Finally, the particularly hard users will receive recommendations generated by highly optimized recommenders. We will compare both required efforts and recommendation quality against approaches dealing with all users in the same ways. Such approaches will include optimizing recommenders to perform well averaged over the full set of users and recommending all users with trivial recommenders, for instance most popular recommendations. Our hypothesis is that the difficulty score will allow us to achieve lower costs along with a comparably high recommendation quality by selecting an appropriate recommendation algorithm for a given user. In addition, we will observe what user characteristics determine her difficulty.

5. CONCLUSION AND FUTURE WORK

We introduced the notion of difficulty in the context of recommending items to users. Measuring that task's difficulty on user-level allows a more personalized optimization of recommender systems. Users with comparably low difficulty scores receive adequate recommendations without much efforts. On the other hand, users with a comparably high difficulty score may be asked to provide additional data to improve the system's individual perception. A combination of the Q statistic and the difficulty measure θ - both adjusted to fit the recommendation scenario - allows us to measure the difficulty for a given user. The calculation requires a sufficiently large data set containing user preferences on items along with a set of implemented recommendation algorithms. We introduced an experimental setting that we will use to evaluate the presented methodology in section 4. In addition, we intend to apply pattern recognition techniques to find correlations between the difficulty score and user characteristics. For instance, the number of ratings is known to correlate with the difficulty for new users ("cold-start problem") and users with a large number of items ("power-users") [10].

6. ACKNOWLEDGEMENTS

This work is funded by the DFG (Deutsche Forschungsgemeinschaft) in the scope of the LSR project.

7. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transaction on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[2] J. A. Aslam and V. Pavlu. Query hardness estimation using jensen-shannon divergence among multiple scoring functions. In *Proceedings of the 29th European conference on IR research*, ECIR'07, pages 198–209, 2007.

[3] R. M. Bell and Y. Koren. Lessons from the netflix prize challenge. *SIGKDD Explor. Newsl.*, 9(2):75–79, 2007.

[4] A. Bellogin. Predicting performance in recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 371–374. ACM, 2011.

[5] M. Hauschild and M. Pelikan. Advanced neighborhoods and problem difficulty measures. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 625–632, 2011.

[6] J. He, M. Larson, and M. De Rijke. Using coherence-based measures to predict query difficulty. In *Proceedings of the IR research, 30th European conference on Advances in information retrieval*, ECIR'08, pages 689–694, 2008.

[7] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.

[8] Y. Koren and J. Sill. Ordrec: an ordinal model for predicting personalized item rating distributions. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 117–124, New York, NY, USA, 2011. ACM.

[9] L. Kuncheva and C. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51:181–207, 2003.

[10] A. Said, B. Jain, and S. Albayrak. Analyzing weighting schemes in collaborative filtering: Cold start, post cold start and power users. In *27th ACM Symposium On Applied Computing (SAC '12)*, New York, NY, USA, 2012. ACM.

[11] G. Takacs, I. Pilaszy, B. Nemeth, and D. Tikk. Major components of the gravity recommendation system. *SIGKDD Explorations*, 9(2), 2007.

[12] S. Vargas and P. Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 109–116, New York, NY, USA, 2011. ACM.

Insights on Social Recommender System

Wolney L. de Mello Neto
Vrije Universiteit Brussel
CoMo Lab
Brussels, Belgium
wdemello@vub.ac.be

Ann Nowé
Vrije Universiteit Brussel
CoMo Lab
Brussels, Belgium
ann.nowe@vub.ac.be

ABSTRACT

Recommender Systems (RS) algorithms are growing more and more complex to follow requirements from real-world applications. Nevertheless, the slight improvement they often bring may not compensate the considerable increase in algorithmic complexity and decrease in computational performance. Contrarily, context aspects such as social awareness are still not much explored. In view of that, this paper proposes insights on how to possibly achieve more efficient and accurate predictions for recommendations by exploring multiple dimensions of a RS architecture. A framework is designed, comprised of a Facebook application called *MyPopCorn* and some scenarios of user neighborhood RSs are proposed. The first one investigates how to recommend movies based on a narrowed subset of collaborative data, extracted from the social connections of the active user. Secondly, connections between users enable a solution for the cold-start problem. Preferences from social connections are aggregated, producing a temporary profile of the new user. Finally, a third dimension is explored regarding evaluation metrics. Results from traditional evaluation by offline cross-validation are compared to measuring prediction accuracy of online feedback data. These insights propose how community-based RS designs might take advantage of social context features. Results show that all three proposed solutions perform better assuming some conditions. Social neighborhoods can often provide representative data for collaborative filtering user-neighborhood techniques, improving a lot the RS performance in terms of computational complexity metric without compromising prediction accuracy. Assuming a user has a dense social network, the cold-start problem can be easily tackled. Finally, rating prediction accuracy performs better when evaluated online than by offline cross-validation.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.3.3 [**Information Search and Retrieval**]: Collaborative Filtering; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Recommender System, Collaborative Filtering, Social Recommenders, Cold-Start Problem, Evaluation Metrics

1. INTRODUCTION

Our generation faces several tough challenges within the current *peta-*, *exa-* or even *zettabyte* information era. Every day we deal with huge amounts of information whose manipulation and storage struggles even on high-end computer technologies. Shifting from the point of view of computer capacity to an average single person, the problem gets even worse due to human being limitations. Online services are examples of big data resources with increasing importance in our lives. About two years ago, Google's search engine used to process approximately half of the entire written works of mankind per day [6]. Nowadays, it is impossible to avoid such reality while working, studying, and entertaining yourself. Perhaps this information overload comes with high cost, nevertheless, high benefit as well.

Movie domain is a great context where information overload is a high potential pain point to be explored. Moreover, Netflix movie streaming service is a good motivation for this work due to two main reasons. Firstly, figures disclosed in [1] mention 75% of their sales come from recommendations. Secondly, [1] reveals the decision of not implementing commercially the algorithm with around 10% improvement in prediction accuracy, winner of US$ 1 million prize[8]. Taking these facts into account, what would be the most potential path to explore within the field of RSs? Is accuracy the most important metric to take into account? What about computational complexity and transparency? What about online instead of offline evaluation methods?

Rather than building upon complex RS methods, this paper investigates a social framework for developing state-of-the-art RS. Aiming at current main challenges, this paper proposes contributions on how to tackle some of its most relevant issues based on possibilities enabled by social context information. The three explored RS challenges are: *(i)* performance issues related to scalability of recommender systems; *(ii)* lack of knowledge about new users, known as cold start problem; and *(iii)* definition of good evaluation methods.

Some insights are discussed based on how social-graph data enable a good implementation of a user neighborhood RS algorithm, focusing not only on prediction accuracy but also on other metrics such as scalability, computational complexity and transparency. These insights lead to 3 hypotheses listed below:

i. A user's social neighborhood is sufficiently representative to provide efficient, in the sense of computational complexity, and effective recommendations, in terms of prediction accuracy;

ii. Social neighborhood connections can derive assumptions about new users taste, avoiding the cold-start problem;

iii. Online evaluation of transparent recommendations should be a valid metric within social RSs.

2. RELATED WORK

In the introduction of the latest survey in RS field, [15] highlights current challenges for RSs. Some of them are investigated hereby, such as follows:

Scalability In real-world applications, the number of instances might often steeply increase in multiple dimensions such as number of users, items and, in turn, user-item preference signals. Despite being a good scenario for some RS algorithms to achieve better accuracy, bigger datasets may lead to a great increase in computational complexity.

[7] proposes an evaluation of top-N recommendation algorithms. Item-based RS is proposed as an alternative for non-scalable user-based recommenders, since it performs better when there are many more users than items. Some other item-based RSs avoiding scalability problems within memory-based CF algorithms are compared in [16].

Regarding model-based CF techniques, [17] follows a reasoning that is similar to the solution presented in Section 4.1, since both look for a narrowed neighborhood which does not to compromise general performance. Whereas the cited papers are based on clustering techniques, our heuristic consists of narrowing the database to a subset of user social-graph connections. Although scalability is an intrinsic disadvantage to user-based RS, the proposition of a local neighborhood might overcome this drawback. User-based RS is adopted since it enables some features related to the social RSs, such as transparent explanations for each recommendation;

Data Sparsity It is among the main bottlenecks for RSs. The lack of information is a big problem, especially during first interactions of a new user. This scenario is defined as the cold start or new user problem, which is traditionally solved by requiring initial user information before any recommendation is given. Nevertheless, this interaction is time consuming, since the user has to look for a couple of items to rate. To improve that, [14] has compared 6 techniques to generate this first list of items, aiming to maximize the percentage of rated items out of all items presented to a new user.

Besides requiring this first interaction with the RS, one could think of a temporary user profile in order to enable initial recommendations. [11] explores trust networks and propose the incorporation of preferences from trusted users. Nevertheless, the new user still has to explicitly provide information about who are his/her trusted users. Our work retrieves implicit information from social networks, regardless trust measurements. The method consists of retrieving social connections and building a virtual profile based on aggregation methods, originally proposed for group RSs. [13] describes 10 aggregation methods and empirically concludes that social-based think is the best basis for generating an artificial preference profile. The author claims that *Least Misery*, *Average* and *Average without Misery* are the most human-like reasoning techniques, achieving very good results.

Transparency Users eventually question themselves about the reasoning behind a recommendation. They are more inclined to accept and evaluate better once they understand how an item has been suggested to him or her. Nevertheless, it is not always possible to provide such a transparent explanation. [9] presents a survey on content-based RS and compares them to CF techniques also in terms of transparency. The authors claim CF techniques are a black box, and it is indeed the truth for most cases. In the case of user-neighborhood RSs, although RSs could tell to the active user about people with close taste that influenced the recommendation, privacy issues may not allow such transparency. In view of this challenge, this paper counteracts the affirmation made by the previously cited survey. It is possible to give explanation on user-based collaborative filtering technique once one assumes not having privacy issues, a tractable scenario within social networks, where connections previously agree on sharing some information. Besides this proposal, some solutions to tackle CF limitations related to transparency are proposed in [4].

Evaluation One of the main modules of a RS design, evaluation strategy is a critical and subjective aspect to be shaped throughout the whole process of building and maintaining a RS. Even though most papers adopt accuracy as the most important metric, one should consider many other evaluation criteria, as presented in [5]. Computational complexity is one metric highlighted in the insight presented in Section 4.1. Transparency is enabled by social context, as discussed in Section 3.1.3. Besides exploring metrics, this paper also focus on questioning methods (see Section 4.3). Offline and online methods should be compared while measuring rating prediction accuracy.

2.1 Social Recommenders

In view of all issues previously listed and the fact some state-of-the-art architectures might not be that attractive for commercial purposes, this paper dives into a RS design that is gaining special attention: Social RSs. Also called community-based recommenders, the basic architecture embeds context data into either collaborative filtering or content-based algorithms, improving the RS performance. According to [15], community-based paradigm is still a hot topic and it is not possible to find a consensus about whether social recommenders have better performance. [19] presents a broad survey on social recommenders. One could see social data in two ways: *(i)* unweighted social graph; *(ii)* or a more complex weighted social-graph. The former has been selected for this paper experiments based on empirical conclusions made by [2] while comparing CF and Social Filtering. Similarities between friends were in average higher than the same correlation measurement between non-connected users. Moreover, both weighted and basic social RSs performed the same or better than pure collaborative filtering RSs for the referred case.

Further than looking at social connections, the latter is

Table 1: MyPopCorn and GroupLens datasets.

	Users	Ratings	Movies
MyPopCorn	129	14k	3k
GroupLens	72k	10M	10k

a trust-based RS that focuses on weighted relationships. A clear comparison between social RS and trust-based RS is defined in [10]. Moreover, [3] highlights the possibility of explaining recommendations based on social connections and the fact active users rate better the RS in case of existing such transparency. Finally, the social RS described hereby profits from an unweighted social graph.

3. FRAMEWORK

As claimed in [15, pg 15], the context in which a RS is developed and its expected features determine the optimal algorithm to be adopted. Parameters such as movie domain, social community context, rating strategy and sparse data were definitely crucial to come up with the final architecture described hereby. A Facebook application called *MyPopCorn*[1], the RS front-end, and a social based implementation of user neighborhood CF algorithm compose the current framework, to be presented in the two following sections.

3.1 MyPopCorn, a Facebook app as Front-End

The idea of building this movie recommender system and making it available on a social network is due to the fact social graph enables proposed recommendation experiments based on social neighborhoods. Moreover, the capability of recommending to an active user and receiving an online feedback on rating prediction accuracy on recommended items is decisive to benchmark the implemented algorithms.

MyPopCorn is a web movie recommender system. Some of its interfaces are composed as follows:

First screen presents a brief description of the main features before the user joins the application. After that, an active user can check statistics about top users and friends;

MyTaste is where a user can rate movies. Recommendation-wise, this is one of the main interactions with the user, in which RS collects data;

My Friends' Taste presents a list of friends and their respective number of ratings. The more ratings each friend has, the bigger his or her basket gets.

3.1.1 Social-Graph Data

The first collaborative data with ratings over movies were taken from GroupLens 10M dataset. From that point, the database was increased with ratings from users of *MyPopCorn*. Information about users, friendships are also made persistent into the same database. The dataset used for the experiments is summarized in Table 1.

In a very short timeframe, the application was accepted by a good number of users. Almost 130 active users have been exploring the application during 2 months time. Figure 1 illustrates all users who contributed for the experiments carried out into this paper. The more movies a user rates, the bigger the node is represented in the social graph. The average degree of connections in this graph was 10.543.

[1]http://mypopcorn.info/

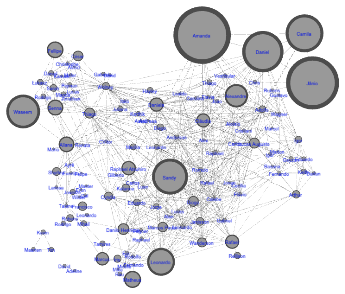


Figure 1: Social Graph representation of *MyPopCorn* database.

3.1.2 Rating Strategy

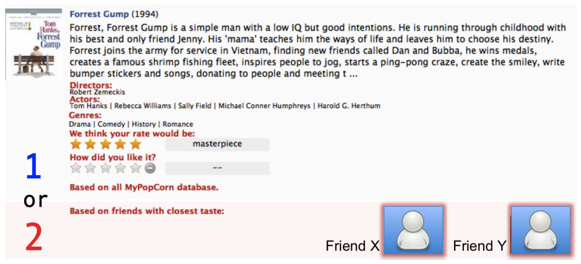
In *MyPopCorn*, the user can choose a rating from 1 to 5 'stars'. Asymmetric labels were defined for each of the 5 stars to achieve a more homogeneous judgment, namely *Bad*, *Regular*, *Good*, *Great* and *Masterpiece*. Test users reported good feedback on the proposed rating strategy claiming this discrete labeled design is certainly more intelligible, where users can have a hint of what each rating value may represent. While following such design, this research aims at reducing subjectivity that is intrinsic to rating process, the core interaction responsible for obtaining the main input of a Collaborative Filtering RS. This strategy also prevents the necessity of the RS to normalize user ratings.

3.1.3 Recommendation Strategy

Recommendations are generated from two implementations of user neighborhood recommenders, such as follows:

- Provided by a traditional user-based RS. The neighborhood calculated among all users in the database;
- Provided by a social-graph user-based RS. A social neighborhood is based on the set of active user friends, to be described in more details in the next section.

A shuffled list of recommendations generated by both RS implementations is presented to the user. Movie description and a continuous predicted value is presented. Therefore, recommendations are seen as a regression and not a classification problem within this framework. Finally, at the bottom of the frame one can see the explanation about each recommendation(see Figure 2). In the first example on light blue background, a message informs the recommendation was *"Based on all MyPopCorn database"*. Alternatively, the second message informs that is was *"Based on friends with closest taste"*, followed by the list of users *Friend X* and *Friend Y*.



Forrest Gump (1994)
Forrest, Forrest Gump is a simple man with a low IQ but good intentions. He is running through childhood with his best and only friend Jenny. His 'mama' teaches him the ways of life and leaves him to choose his destiny. Forrest joins the army for service in Vietnam, finding new friends called Dan and Bubba, he wins medals, creates a famous shrimp fishing fleet, inspires people to jog, starts a ping-pong craze, create the smiley, write bumper stickers and songs, donating to people and meeting t ...
Directors:
Robert Zemeckis
Actors:
Tom Hanks | Rebecca Williams | Sally Field | Michael Conner Humphreys | Harold G. Herthum
Genres:
Drama | Comedy | History | Romance
We think your rate would be:
★★★★★ masterpiece
How did you like it?
☆☆☆☆☆ --
Based on all MyPopCorn database.
Based on friends with closest taste:

Figure 2: Recommendation strategy in *MyPopCorn*.

This system is designed to give the most transparent recommendations possible. In view of that, the reasoning behind the RS can be better understood by presenting the real number as predicted rating value. Furthermore, explaining the recommendation with a list of users will transform a formerly impersonal recommendation into a social passive interaction between friends. Due to privacy issues, presenting this list is only possible for the social neighborhood approach, where content sharing among users is agreed in advance.

3.2 Movie RS Back-End

The final architecture of the social-graph recommender was developed on top of the user-based RS implementation provided in Mahout[2]. User neighborhood CF paradigm has close reasoning to social user behavior, being the most relevant criterion that influenced this design choice. In possession of information about users taste, this user-centered method focus on comparing similarity among users. Furthermore, friendship data will be essential to enable modifications on the original algorithm. Insights on how to profit from social context information in different dimensions will be addressed below.

4. INSIGHTS ON RS CHALLENGES

As the title suggests, solutions to the current RS challenges listed in Related Work are described in this section. Each of the following implemented scenarios tackle three main challenges previously mentioned, namely *computational complexity* issues of scalable user-neighborhood RSs; *sparse data* about new users, known as cold start problem; and definition of *optimal evaluation methods* for transparent and non-transparent recommendations.

4.1 Social Neighborhood

The idea of narrowing the dataset to a subset of users aims to tackle scalability constraints and increase real-time performance, two issues that are intrinsic to user-based RS [7]. Assuming that calculating an active user's neighborhood (comprised of k similar users) among his or her social connections might be representative enough, good recommendations could be achieved without the necessity of comparing a user preference vector with all other users in the database. This hypothesis is based on a related work comparing the correlation between users similarity and the binary fact of being or not being friends[2]. It was observed that similarities between friends are in average higher than the same correlation measurement between non-connected users.

Experiments were performed in order to investigate the three insights proposed above. A standard user-based neighborhood RS setup is incrementally modified from the current insight until the third one. This scenario focus on predicting ratings contained in a training set comprised of 5% of all 14.367 ratings provided by *MyPopCorn* users. The reason for not adding any rating from GroupLens into the training set of the standard neighborhood is allow a fair comparison between both neighborhoods. By applying two strategies, namely *Standard* full neighborhood and hereby proposed *Social* one, some hypotheses are tested: *(i)* Real-time recommendation performance will become much more efficient while adopting social neighborhood; *(ii)* Rating prediction accuracy from social neighborhood recommendations will be as much precise as in the standard method.

[2]Apache Mahout machine learning library

For the proposed experiment methods, standard neighborhood RS performs around 70k calculations, the number of all users in the merged dataset. In the case of social neighborhood, the number of comparisons is relative to the degree of each node (user) in the social graph, which varies from 0 to 49 for *MyPopCorn* dataset with an average degree of 10.543. Concerning average runtime, whereas prediction process for one rating takes around 950.55 ms for standard neighborhood, after narrowing the search space to the set of social connections, it takes in average 69.975 ms, 92.63% lower. Regarding accuracy, Figure 3 presents prediction accuracy error for this new neighborhood compared to the standard implementation. Both implementations were compared by varying the size of the neighborhood k while experimenting two values of threshold t=1 and t=2. This threshold defines the minimum number users in the neighborhood that rated a same candidate item. When t=2, the items rated by only one user in the neighborhood are not taken into account.

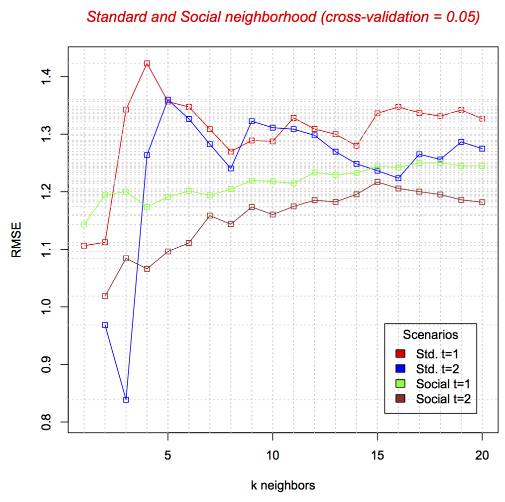


Figure 3: Standard and Social Neighborhoods prediction accuracy (*RMSE*).

The minimum $RMSE = 0.8385664$ was obtained by Standard neighborhood (k=3,t=2). Besides that, Social (k=2,t=2) achieved $RMSE = 1.018598$. Surprisingly, rating prediction accuracy also improved. Except for values of k neighbors equal to 2 and 3, Social Neighborhood outperforms, in average, the standard method, confirming the first hypothesis for this scenario. Besides that, the value of threshold t=2 performs better. The fact of accepting only items rated by at least two users might have increased the confidence on preference data, achieving better accuracy results. On the contrary, hypothesis 2 was surprisingly refuted. Instead of performing almost the same as in the original approach, Social Neighborhood can *significantly* outperform prediction accuracy for $k > 3$. While increasing the value of k, such social neighborhood enables a more accurate predictions and, probably, reaching higher serendipity.

Remark: This approach is not available for people with no or few friends, suffering from the cold start problem, to be solved next.

4.2 Social Aggregation for Cold-start Problem

One of the main issues related to RS, the cold-start problem or new-user problem prohibit some active users to receive recommendations. In the dataset used for all experiments, 21 users out of 129 have rated less than 10 movies, while others more than a thousand. These users with few ratings are almost unable to receive any recommendation.

Instead of adopting the classic approaches such as content-based or presenting a list to be rated as from the first user interaction, this paper proposes a solution based on social-graph information. It is based strategy from group RS based on aggregating user profiles. One could see this problem following the quote *"Tell me who your friends are and I will tell you who you are"*. This reasoning is also motivated by the work carried out in [2], where social filtering is explored and conclusions reinforce the suggested heuristic. Likewise, [?] developed a probabilistic RS and achieved good results in experiments where active users were recommended items based on the preferences of his or her social connections. On the contrary, the idea presented in this paper follows the same reasoning of absorbing social context data into the system to solve the cold-start problem, nevertheless, by different means (based on group RS) and in a different RS implementation technique (user neighborhood RS).

Among some aggregation techniques mentioned in the Related Work, *Average without Misery* is adopted, since it finds a balance between the *Least Misery* and *Average*. It preserves the main advantages of both aggregation strategies originally applied to group RS and now reflected in the aggregated virtual profile to be considered by our single-user RS. It follows the human-like reasoning in which a group of people tend to select items that please, in average, most persons involved. Moreover, it excludes items once rated below a defined threshold, as described by [13]. The same author proposed such aggregation for solving the cold-start problem in [12], although in a different RS paradigm. Experiments were run in order to test the following hypothesis: *(i) Recommendation accuracy for aggregated virtual social profile performs not much worse than cross-validation of real ratings. Hence, it would be a feasible solution to the cold-start problem.*

The social neighborhood method was adopted with parameters k=4 and t=1, so that the most number of predictions are enabled. The idea here is to investigate how many active users had the cold-start problem, meaning their neighborhoods were empty. While repeating the experiments from last section in 5% of MyPopCorn ratings dataset, around 103 users were in the testset. Nevertheless, RS could not estimate any rating for 13 users due to empty neighborhood issue. 6 users had no social connections, what can not be solved by the method proposed here. The remaining 7 users had their ratings predicted with accuracy error of *RMSE = 1.69588.*

One should raise the question that this is not much data, referring to the tiny set of 7 users. In view of that, another experiment has been run on 50% of ratings in *MyPopCorn* dataset. Ratings of 44 users experiencing the cold-start problem were hidden iteratively in order to be predicted by the RS. Foreach of the 44 users, the RS generated a virtual profile based on aggregating all ratings from their friends, including those removed in order to artificially cause the cold-start problem. Only 8 new users(18%) could not be helped by this method of aggregation due to the fact of having no social connections. Prediction accuracy error was *RMSE = 1.37461.*

Compared to the accuracy evaluated in the experiments of previous sections (*RMSE = 1.173435* for k=4, t=1), this proposed solution to the cold-start problem has decreased performance in around 20%, considering the *RMSE = 1.37461.* In view of that, the proposed solution is considered to be a good alternative for social RSs. Besides not compromising the prediction accuracy significantly, this method should be considered in terms of how efficient the RS can deal with new users that are not interested in providing many ratings as from the first interaction. Despite not being an objective metric, the ability of solving the cold-start should be incorporated into RS evaluation.

4.3 Comparison of Evaluation Methods

While the first insight focuses on the two objective evaluation metrics, namely prediction accuracy and computational complexity, this insight focuses on transparency, a subjective metric, and evaluation methods. The most popular evaluation metric throughout RS state-of-the-art, prediction accuracy benchmark is often based on offline cross-validation and error calculation over *Root Mean Squared Error - RMSE.* In view of that, this third and last section compares offline and online methods of calculating estimation accuracy together with more transparent recommendations based on social explanation. One hypothesis is that this online method might make offline approach suboptimal for the context of social recommenders. Instead of cross-validation, one should consider the social factor involved within online evaluation. Due to the strategy of recommending a list of movies whose predicted ratings might not be always high and to make it more transparent, the predicted value is presented to the active user. Assuming that not many people tend to converge with the RS prediction, this strategy will not bias the comparison. Actually, we believe there are people who also try to diverge from what has been predicted.

The current experiment intends to test the effect of explained recommendations, as previously described in [18], but now in the context of social RSs, as defined in the following hypothesis: (i) Assuming social RSs where recommendations based on social connections are explained, rating estimation accuracy achieve better results if evaluated online, instead of offline.

Besides *RMSE*, metrics such as novelty or serendipity were taken into account while choosing higher values of k other than the ones that reached minimum accuracy, shown in Figure 3. Although the same number of recommendations with standard and social neighborhood were generated, active users gave more feedback on the social ones. 119 online feedbacks were provided, as presented in Table 2 in comparison with the traditional offline method.

As Table 2 shows, Standard Neighborhood method achieved a prediction accuracy of 1.0646 and Social Neighborhood RS setup achieved better rating prediction accuracy of *RMSE* = 0.9952. Both of them presented an improvement when evaluated online other than offline. The decrease in *RMSE* was of 14.16% and 6.64%.

Hypothesis was confirmed by the numbers shown in Table 2. Surprisingly, online evaluation accuracy with Standard Neighborhood improved better (14.16%) than 6.64% gain achieved by Social Neighborhood strategy. Finally, results have shown that, in average, RSs tend to present better

Table 2: Online evaluation of social and standard neighborhood.

	Std. N.	**Social N.**
Setup	k=8, t=2	k=4, t=2
OFFLINE *RMSE*	1.240429	1.066049
ONLINE *RMSE*	1.064686	0.995211
Improvement	14.16%	6.64%

accuracy results in online evaluations than offline for both explained and non-explained recommendations.

5. CONCLUSIONS

This paper first discussed the computational requirements intrinsic to user neighborhood RS, by nature a non-scalable algorithm. Based on the two most important evaluation metrics, state space reduction enabled a decrease of *92.63%* in computational complexity, while not compromising accuracy. Instead, the latter also improved.

Social graph was essential to enable a solution to the cold-start problem. Tested with success in group RS, *Average without Misery* enabled creation of virtual profiles based on active users network. Results confirmed the proposed hypothesis, indicating this solution as a good alternative to this issue while presenting a decrease on prediction accuracy of only *20%* by cross-validation.

Another important achievement was caused by transparent recommendations. Results from the third insight turn prediction accuracy by cross-validation an even more questionable benchmark method. Both neighborhood formation methods presented a considerable improvement of *6.64%* and *14.12%*. While choosing online evaluation methods, one could have better conclusions about the RS quality.

6. ACKNOWLEDGMENTS

This research is part of a master studies sponsored by *Monesia: **MO**bility **N**etwork **E**urope-**S**outhamerica: an **I**nstitutional **A**pproach*, an Erasmus Mundus External Cooperation Window.

Thanks Lucas Carvalho, researcher at Federal University of Sergipe - Brazil, for cooperating on the development of the Facebook application named MyPopCorn.

7. REFERENCES

[1] X. Amatriain and J. Basilico. Netflix recommendations: Beyond the 5 stars, 2012.

[2] G. Groh and C. Ehmig. Recommendations in taste related domains: collaborative filtering vs. social filtering. In *Proceedings of the 2007 international ACM conference on Supporting group work*, pages 127–136. Citeseer, 2007.

[3] I. Guy, N. Zwerdling, D. Carmel, I. Ronen, E. Uziel, S. Yogev, and S. Ofek-Koifman. Personalized recommendation of social software items based on social relations. In *Proceedings of the third ACM conference on Recommender systems*, pages 53–60. ACM, 2009.

[4] J. Herlocker and J. Konstan. Explaining collaborative filtering recommendations. *of the 2000 ACM conference on*, pages 241–250, 2000.

[5] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[6] D. Infographic. Visualizing the petabyte age, 2010.

[7] G. Karypis. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 247–254. ACM, 2001.

[8] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[9] P. Lops, M. Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. *Recommender Systems Handbook*, pages 73–105, 2011.

[10] H. Ma, D. Zhou, C. Liu, M. Lyu, and I. King. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296. ACM, 2011.

[11] P. Massa and P. Avesani. Trust-aware collaborative filtering for recommender systems. *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, pages 492–508, 2004.

[12] J. Masthoff. Modeling the multiple people that are me. *User Modeling 2003*, pages 146–146, 2003.

[13] J. Masthoff. Group modeling: Selecting a sequence of television items to suit a group of viewers. *User Modeling and User-Adapted Interaction*, 14(1):37–85, 2004.

[14] A. Rashid, I. Albert, D. Cosley, S. Lam, S. McNee, J. 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.

[15] F. Ricci, L. Rokach, and B. Shapira. Introduction to recommender systems handbook. *Recommender Systems Handbook*, pages 1–35, 2011.

[16] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[17] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the Fifth International Conference on Computer and Information Technology*, pages 158–167, 2002.

[18] N. Tintarev and J. Masthoff. Designing and evaluating explanations for recommender systems. *Recommender Systems Handbook*, pages 479–510, 2011.

[19] P. Victor, M. Cock, and C. Cornelis. Trust and recommendations. *Recommender Systems Handbook*, pages 645–675, 2011.

User Evaluation of Fusion-based Approach for Serendipity-oriented Recommender System

Kenta Oku
College of Information Science and Engineering,
Ritsumeikan University
1-1-1 Nojihigashi, Kusatsu City
Shiga, Japan
oku@fc.ritsumei.ac.jp

Fumio Hattori
College of Information Science and Engineering,
Ritsumeikan University
1-1-1 Nojihigashi, Kusatsu City
Shiga, Japan
fhattori@is.ritsumei.ac.jp

ABSTRACT

In recent years, studies have focused on the development of recommender systems that consider measures that go beyond simply the accuracy of the system. One such measure, serendipity, is defined as a measure that indicates how the recommender system can find unexpected and useful items for users. We have previously proposed a fusion-based recommender system as a serendipity-oriented recommender system. In this study, we improve upon this system by considering the concept of serendipity. Our system possesses mechanisms that can cause extrinsic and intrinsic accidents, and it enables users to derive some value from such accidents through their sagacity. We consider that such mechanisms are required for the development of the serendipity-oriented recommender system. The key idea of this system is the fusion-based approach, through which the system mixes two user-input items to find new items that have the mixed features. The contributions of this paper are as follows: providing an improved fusion-based recommender system that adopts a fusion-based approach to improve serendipity; practically evaluating the recommender system through user tests using a real book data set from Rakuten Books; and showing the effectiveness of the system compared to recommender systems on websites such as Amazon from the viewpoint of serendipity.

Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: Information Filtering

General Terms

Experimentation

Keywords

Recommender systems, Serendipity-oriented recommender systems, Serendipity

1. INTRODUCTION

In recent years, several studies have focused on the development of recommender systems that consider measures beyond simply the accuracy of the system, such as the novelty, diversity, and serendipity [1][2]. This is because these studies have found that users are not always satisfied with recommender systems with only high accuracy—they desire for the systems to consider various other viewpoints, too.

In an attempt to satisfy this need, in this study, we focus on the serendipity. Serendipity means "the ability to make unexpected and valuable discoveries by accident." We thus define a serendipitous item as something unexpected and valuable, and we believe that such an item can diversify users' interest regardless of their experiences, thus making their lives richer. This study therefore aims to develop a serendipity-oriented recommender system that provides users with serendipitous items.

First, it is necessary to gain some insight into the original meaning of the word "serendipity." The word "serendipity" originated from a story called "The Three Princes of Serendip" [3], which tells the story of three princes. These princes discovered a series of novel things during the course of various and unexpected events on their journeys, which they attributed to their luck. Horace Walpole, who read this story, stated that "the princes were always making discoveries, by accidents and sagacity," to describe which he coined the word "serendipity," which means "the ability to make unexpected discovery by accidents and sagacity" [4]. In light of Walpole's definition, we believe that a serendipity-oriented recommender system should possess an interface that has mechanisms that output "unexpected discoveries" based on the input of "accidental events" experienced by the users and the sagacity of the users.

In addition, [4] states that accidents are of two types: "extrinsic" and "intrinsic." For example, a well-known serendipitous discovery is that of gravity—it is stated that "Newton had an inspiration of the notion of universal gravitation at the sight of an apple that fell from a tree"[5]. In this event, the apple falling from the tree can be considered an "extrinsic accident," that is, one that occurs regardless of the action of a person. Another example of a serendipitous discovery is that made by Koichi Tanaka, which won him the Nobel Prize in Chemistry in 2002. Although he realized that he had accidentally used glycerin instead of acetone as a sample, he continued his experiments in order to observe the results. This led to him discovering an unknown phenomenon. In this event, the discovery of the unknown phenomenon can be considered an "intrinsic accident," that is, one that results from the action of a person with the positive expectation of something. It is of great importance to derive some value from these accidents. In this light, a person's sagacity plays a crucial role.

The above-described examples suggest that a serendipity-oriented recommender system should have an interface consisting of the following mechanisms:

(a) A mechanism that causes extrinsic accidents.

(b) A mechanism that causes intrinsic accidents.

(c) A mechanism that enables users to derive some value from accidents through their sagacity.

In this study, we have proposed a fusion-based recommender system to satisfy these requirements. The key idea of this system is adopting a fusion-based approach for discovering serendipitous items by mixing two user-input items together. As described at the beginning of this section, we define a serendipitous item as an unexpected and valuable item. Specifically, the following items are relevant to serendipitous items:

- Items that can excite the user's interest for the first time although he/she does not know about them and he/she would not be able to discover them by himself/herself.
- Items that can excite the user's interest for the first time although he/she thought that he/she was not interested in them.
- Items that can attract the user's interest after being displayed by the system.

We also define a high-serendipity recommender system that can recommend more serendipitous items to users.

By using our proposed fusion-based recommender system, a user can mix two items together in the system interface to create something new from something existing in a manner analogous to mixing colors, ingredients, or sounds. The act of mixing also entails the following:

a) We can intuitively expect mixed results from a combination of inputs. On the other hand, some combinations can yield unexpected results.

b) Because our curiosity may be aroused by the intuitive comprehensibility and unexpectedness of the act of mixing, we might feel like being creative and mixing various combinations of inputs.

Characteristic (a) corresponds to the mechanism that causes intrinsic accidents because unexpected results may be produced by mixing materials together with the expectation of some positive results. Characteristic (b) corresponds to the mechanism that enables us to derive some value from accidents through our sagacity in that we can select valuable inputs from among the given inputs.

Figure 1 shows the interface of the fusion-based recommender system for book recommendation. When the user clicks [Random], [Search], [Popular], and [New] buttons, the system randomly provides the user with corresponding books from the book database. Randomly providing books corresponds to the mechanism that causes extrinsic accidents.

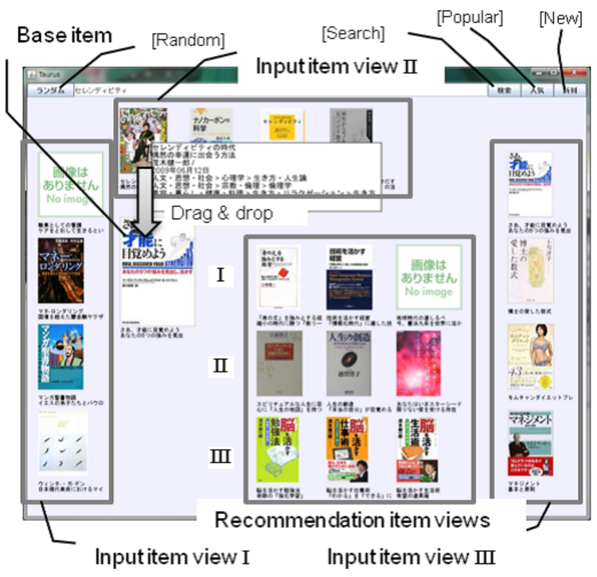


Figure 1: Interface of Fusion-based Recommender System.

The user can also select an interesting book as a material from the displayed books based on his/her sagacity, and then drag-and-drop it into a base book, which is also selected by the user. The system then provides the user with books possessing mixed features of the two books. Although the user can select books to mix with some expectation, some book combinations may yield unexpected results. This may cause intrinsic accidents. The user can repeatedly and creatively use the system to see various mixing results until he/she is satisfied with the results. In this process, serendipitous items are interactively provided to the user.

We have already developed a predecessor to the proposed fusion-based recommender system[6]. In this study, we have improved upon the system interface and internal processing based on the deeper idea of serendipity, and we have evaluated this system from the viewpoint of practical use.

The contributions of this study are as follows:

- developing the improved fusion-based recommender system that adopts a fusion-based approach for improving the serendipity;
- experimentally evaluating the practical usability of the recommender system using a real book data set from Rakuten Books;
- showing the effectiveness of the system compared to recommender systems on websites such as Amazon from the viewpoint of serendipity.

2. RELATED WORK

Herlocker et al. [1] suggested that recommender systems with high accuracy do not always satisfy users. Therefore, they suggested that recommender systems should be evaluated not only by their accuracy but also by various other metrics such as novelty, diversity, and serendipity.

Several studies have already focused on serendipity in the context of recommendation. Ziegler et al. [7][8] suggested that diversifying recommendation lists improves user satisfaction. Toward this end, they proposed topic diversification based on an intra-list similarity metric. Sarwar et al. [9] suggested that serendipity might be improved by removing obvious items from recommendation lists. Berkovsky et al. [10] proposed group-based recipe recommendations. They suggested that recipes loved by a group member are likely to be recommended to others, which may increase serendipity.

Hijikata et al. [11] and Murakami et al. [2] proposed recommendation methods that predict novelty or unexpectedness. The former study proposed collaborative filtering, which predicts unknown items for a target user based on known/unknown profiles explicitly acquired from the user, and showed that such filtering can improve novelty by providing unknown items to the user. The latter study proposed a method that implicitly predicts unexpectedness based on a user's action history. They introduced a preference model that predicts items the user likes and a habit model that predicts items habitually selected by the user. The method estimates the unexpectedness of recommended items by considering the differences between the models. The disadvantage of these methods is that they need to obtain models or profiles for an individual user. Our proposed system, however, does not have these requirements. It can instantly recommend serendipitous items based on items the user has just selected.

Murakami et al. [2] and Ge et al. [12] introduced measures for evaluating the unexpectedness and serendipity of recommender systems.

The former study assumed that unexpectedness is the distance between the results produced by the system to be evaluated and those produced by primitive prediction methods. Here, primitive methods include recommendation methods based on user profiles or action histories. Based on this notion, they proposed *unexpectedness* for measuring the unexpectedness of recommendation lists and $unexpectedness\_r$ to take into account the rankings in the lists. The latter study also propose unexpectedness following the notion of the former study.

In our previous study[6], we evaluated our recommender system based on Murakami et al.'s evaluation metrics. However, we did not evaluate the system through tests involving real users to determine its serendipity. In contrast, in this study, we evaluate our proposed fusion-based recommender system through experiments involving real users.

3. FUSION-BASED RECOMMENDER SYSTEM

In this section, we describe our proposed fusion-based recommender system. This system has an interface that consists of the aforementioned mechanisms for recommending serendipitous items (Figure 1).

As shown in Figures 1 , a user selects a base item from items displayed in views and drags-and-drops another material item onto the base item. Then, the system mixes these two items and outputs recommended items that have features of both, which we define as fusion. The user can repeatedly perform fusion by reselecting the base items and researching the material items until he/she obtains acceptable results. During this process, the user may interactively discover serendipitous items.

In Section 3.1, we describe the book database used as the recommendation content in this study. In Section 3.2, we describe the system interface and the user interactions related to the above mechanisms. Finally, in Section 3.3, we show fusion methods as the internal processing of the fusion.

3.1 Book database

In this study, we consider books as the recommendation content; in the future, of course, we intend to apply the system to various contents such as music, movies, and recipes. We collected Japanese book data using Rakuten Books book search API[1] from Rakuten Books[2]. We obtained data for 667,218 books between Dec. 27, 2011, and Feb. 10, 2012.

The book data consists of the attributes of *isbn*, *title*, sub_title, *author*, $sales\_date$, $item\_url$, $review\_count$, $review\_average$, $books\_genre\_id$. We created a *book* table consisting of these attributes, in addition to the following tables:

- $book - phrase(\underline{isbn}, \underline{phrase}, idf)$
- $book - author(\underline{isbn}, \underline{author})$
- $book - genre(\underline{isbn}, \underline{genre\_id})$

Here, the $book - phrase$ table contains phrases from $book.title$ and $book.sub\_title$ for each book. In Section 3.1.1, we explain how phrases are extracted. The $book - author$ table contains the authors of each book. The $book - genre$ table contains the genre id of each book. Rakuten Books has 800 genres such as "novels and essays" and "sciences, medical sciences, and technologies," each of which consist of four-level categories. The genre id is a unique id that corresponds to each genre.

3.1.1 Phrase extraction from book data

The system extracts phrases using Chasen[3], a Japanese morphological analyzer, from $book.title$ and $book.sub\_title$ for each book. We heuristically selected "nouns," "verbs," "adjectives," "adverbs," and "unknown words" as target parts of speech. Here, the system extracts also compound words such as "cognitive psychology" that are treated as one phrase.

3.2 System interface

Figure 1 shows the interface of the proposed system, which implements mechanisms (a), (b), and (c) mentioned above.

(a) Mechanism that causes extrinsic accidents.

The system implements [random], [search], [popular], and [new] buttons, which cause extrinsic accidents. When the user clicks each button, the system randomly searches for k corresponding books from the book database. The books are displayed in input item views I, II, and III in Figure 1. Table 1 lists the processes that are called when each button is clicked.

When the user moves the mouse cursor over the books displayed in the views, the book information ("title," "sub title," "authors," "publication date," and "genres") are shown in a pop-up window. When the user right-clicks the books, he/she can view detailed information from the site of Rakuten Books through an external browser.

[1]Rakuten Books book search API (in Japanese): http://webservice.rakuten.co.jp/api/booksbooksearch/
[2]Rakuten books (in Japanese): http://books.rakuten.co.jp/book/
[3]Chasen (in Japanese): http://chasen.naist.jp/hiki/ChaSen/

Table 1: Search processing by each button.

Button	Processing	Target view
Random	Searching k books from the book database at random.	Input item view I
Search	Searching k books at random from books whose *title* or *sub_title* includes keywords input in the text box.	Input item view II
New	Searching k books at random from books satisfying $review\_count \times review\_average \geq \theta$.	Input item view III
Popular	Searching k books at random from books saled last one month.	Input item view III

(b) Mechanism that causes intrinsic accidents.

The system implements a fusion mechanism as an interface that causes intrinsic accidents.

The user can select a base item by double-clicking a book from among the books in the input item view or recommendation item view. The base item is considered as the basis when performing fusion.

The user can select a material item from among the books in the same two views. The material item is used for performing fusion with the base item. When the user drags-and-drops the material item onto the base item, fusion of the two items is performed. The system then displays the items outputted by the fusion in the recommendation item view. In Section 3.3, we define three fusion methods. The system displays items outputted by each fusion method in the corresponding recommendation item view I, II, or III.

(c) Mechanism that enables users to derive some value from accidents through their sagacity.

In mechanism (b), the user can select a base item and a material item from among the books deemed interesting in the views. Such intuitive selection of books may correspond to his/her sagacity.

Here, the type of book that can be selected depends on the user. When performing fusion, the user can select items that are suitable for his/her preferences as well as items that are considered interesting.

3.3 Fusion method

As shown in Section 3.2 (b), fusion is performed using the base and the material item when the user drags-and-drops the material item onto the base item. We define the following three methods as fusion methods. In this section, *bookA*, *bookB*, and *book* denote the base item, material item, and recommended item, respectively.

$phrase - phrase$ *fusion.*

The $phrase - phrase$ fusion method searches for a maximum of m books whose $book.title$ or $book.sub\_title$ includes at least one phrase from the phrase list $bookA.phraseList$ in $bookA$ and at least one phrase from the phrase list $bookB.phraseList$ in $bookB$. The searched books are shown in recommendation item view I. Figure 2 (a) shows an example of fusion for $bookA$—"Equation loved by a doctor"—and $bookB$—"Magic for cleaning up giving palpitations of life." In this case, the system displays "Magic doctor" based on "doctor" in $bookA$ and "magic" in $bookB$.

$phrase - genre$ *fusion.*

The $phrase-genre$ fusion method searches for a maximum

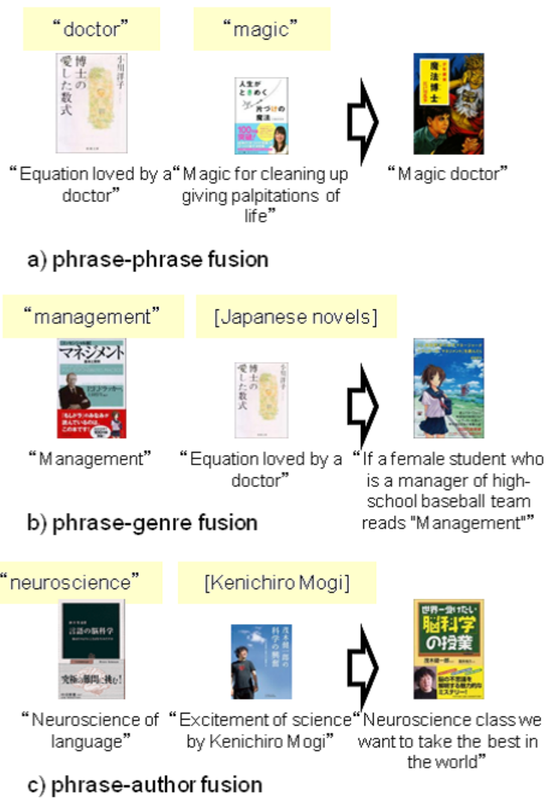

"doctor"
"magic"
"Equation loved by a doctor"
"Magic for cleaning up giving palpitations of life"
"Magic doctor"
a) phrase-phrase fusion
"management"
[Japanese novels]
"Management"
"Equation loved by a doctor"
"If a female student who is a manager of high-school baseball team reads "Management""
b) phrase-genre fusion
"neuroscience"
[Kenichiro Mogi]
"Neuroscience of language"
"Excitement of science by Kenichiro Mogi"
"Neuroscience class we want to take the best in the world"
c) phrase-author fusion

Figure 2: Example of each fusion method.

of m books whose $book.title$ or $book.sub\_title$ includes at least one phrase from the phrase list $bookA.phraseList$ in $bookA$ and whose $book.genre\_id$ corresponds to at least one genre from the genre list $bookB.genre\_idList$ in $bookB$. The searched books are shown in recommendation item view II. Figure 2 (b) shows an example of the fusion of $bookA$—"Management"—and $bookB$—"Equation loved by a doctor." In this case, the system displays "If a female student who is a manager of high-school baseball team reads 'Management'," whose $book.title$ or $book.sub\_title$ includes "management" and whose $book.genre\_id$ corresponds to $bookB.genre\_id$ (i.e., "[novels and essays – Japanese novels]").

$phrase - author$ *fusion.*

The $phrase - author$ fusion method searches for a maximum of m books whose $book.title$ or $book.sub\_title$ includes at least one phrase from the phrase list $bookA.phraseList$ in $bookA$ and whose $book.author$ corresponds to at least one author from the author list $bookB.authorList$ in $bookB$. The searched books are shown in recommendation item view III. Figure 2 (c) shows an example of the fusion of $bookA$—"Neuroscience of language"—and $bookB$—"Excitement of science by Kenichiro Mogi." In this case, the system displays "Neuroscience class we want to take the best in the world," whose $book.title$ or $book.sub\_title$ includes "neuroscience" and whose $book.author$ corresponds to $bookB.author$ (i.e., "[Kenichiro Mogi]").

4. EXPERIMENTS

In this section, we show the experimental results of user tests of our proposed fusion-based recommender system. We implemented this system using Java and Processing as the evaluation system. In the experiments, we selected books as recommendation contents and created the book database described in Section 3.1 using MySQL.

4.1 Experimental method

Nine subjects (eight males and one female) participated in our study. Their age is from 20 to 23. They had average computer skills and used the Internet regularly (every day/nearly every day). They also used online shopping websites such as Amazon very rarely (a few times so far) or rarely (a few times a month). They read books rarely (a few times a month) or moderately (once to three times a week).

The experimental procedure is as follows:

(1) We explain the recommender system to be used to each subject and provide them with the task "Find three books you want to read on holidays."

(2) Each subject carries out the task using the assigned system (without time limitation).

(3) If the subject finds suitable books, he/she marks them (at most 3 books). We call these books the main recommended books.

(4) If the subject finds books that are not suitable but are interesting, he/she marks them (any number of books). We call these books the sub-recommended books.

(5) The subject finishes the task when he/she finds three main recommended books. However, he/she can finish the task if he/she is satisfied or satiated with even less than three books.

(6) After the task is finished, the subject answers all the questions listed in Table 2 for each recommended book.

(7) The subject performs the same steps for each recommender system.

Section 4.2 discusses the recommender systems used in the experiments. Each subject uses the various recommender systems in a different order to cancel any effect that might otherwise be produced.

Table 2 lists the questions about the recommended books. Here, the subjects answered Q1 using a three-level scale {3:unknown, 2:known but never read, 1:have been ever read}, and Q2 to Q4 using a five-level scale {5:strongly agree, 4:agree, 3:neither agree nor disagree, 2:disagree, 1:strongly disagree}. With regard to "by myself" in Q4, we explained to the subjects that "if you think that you can easily find the book by using existing search engines (e.g., Google, Yahoo!) or by using a genre or keyword search at online/real book stores or libraries by yourself, the book is regarded as 'findable book by myself'."

After all tasks were finished, the subjects answered questions, "this system excited my interest and enabled me to discover somthing new," which is related to serendipity of the recommender systems using the same five-level scale.

4.2 Comparative systems

We choose Amazon[4], a large online store with recommender systems, for comparison with our proposed system.

[4]amazon.co.jp (Japanese site): http://www.amazon.co.jp/

Table 2: Questions for recommended books.

No.	Question
Q1	I did not know this book.
Q2	I have been interested in this book before the system presented it to me.
Q3	This book excited my interest for the first time.
Q4	I think that I could not find this book by myself.

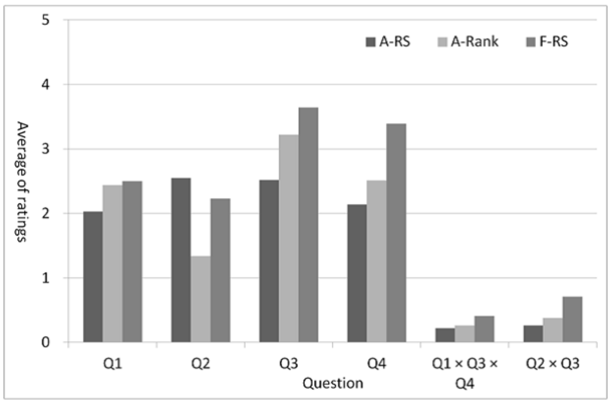


Figure 3: Separate evaluation of sub-recommended book.

We considered two types of systems—Amazon search and recommend (A-RS) and Amazon ranking (A-Rank)—as baseline systems. In this section, we explain the utilization of the baseline systems and the proposed system.

Amazon search and recommend (A-RS).

The subjects are allowed to only use keyword and genre search method on the Amazon site, following which they can use the recommendation list (a list shown under "Customers Who Bought This Item Also Bought"). We encouraged the subjects to refer to the entire recommendation list because toward the end, the list potentially includes unexpected but interesting books. Amazon's recommendation method is implemented by item-based collaborative filtering[13].

Amazon ranking (A-Rank).

The subjects are allowed to only refer to the ranking of "Best Sellers" and "New Releases." They are also allowed to refer to the ranking in each category.

Fusion-based recommender system (F-RS).

We explained the system interface, described in Section 3.2, and how it is used to the subjects in advance. However, we did not explain the details of the internal processing of the fusion method, described in Section 3.3, because we would like to observe whether the subjects can gradually understand the same through trial and error.

Here, we used $k = 4, \theta = 1000$, and $m = 3$, as mentioned in Section 3.2 and Section 3.3.

4.3 Results

4.3.1 Evaluation of sub-recommended books

We analyzed what type of books were marked as sub-recommended books. Figure 3 shows the overall results of the subjects' ratings for Q1–Q4 from Table 2 about sub-recommended books. The figure shows the averages of the ratings for each recommender system.

As described in Section 1, the first definition of serendipitous items is "items that can excite the user's interest for the first time although he/she does not know about them and he/she would not be able to discover them by himself/herself." From this viewpoint, we evaluated the systems based on not only the discoverability but also whether the recommended items excited the users' interest. Therefore, from the viewpoint of serendipity, we analyzed how many items that satisfied the conditions of books that "Q1: I did not know this book," "Q4: I think that I could not find this book by myself," and "Q3: This book excited my interest for the first time" could be found by each system. If the rating of a book for Q1 $= 3$, Q4 ≥ 4, and Q3 ≥ 4, we assign it a score of "1," otherwise we assign a score of "0." Figure 3 shows the averages. We found significant differences between the average of F-RS and those of A-RS and A-Rank by a t-test with a significance level of 5%.

The second definition of serendipitous items is "items that can excite the user's interest for the first time although he/she thought that he/she was not interested in them." From this viewpoint, we analyzed how many items that satisfied the conditions of books that "Q2: I have not been interested in this book" and "Q3: This book excited my interest for the first time" could be found by each system. If the rating of a book for Q2 ≤ 2 and Q3 ≥ 4, we assign it a score of "1," otherwise we assign a score of "0." Figure 3 shows the averages. We found significant differences between the average of F-RS and that of A-RS by a t-test with a significance level of 1%. In addition, we found significant differences between the average of F-RS and that of A-Rank with a significance level of 5%.

Although A-RS recommends books related to the browsed book through item-based collaborative filtering, there is little possibility of the recommended book being largely against the user's interest because of its high accuracy. Meanwhile, because A-Rank recommends popular books, the user may already know the recommended books if they belong to genres the user is interested in. On the other hand, the fusion-based recommender system can recommend books that are occasionally against the user's interest depending on the selection of the material item. This is why the system showed high discoverability, although this involves some risks. In addition, because the recommended books are still relevant to the base item, the user may be interested in them. This is why the fusion-based recommender system was superior from the viewpoint of serendipity.

4.3.2 Evaluation of systems

We focus on the question about serendipity, "this system excited my interest and enabled me to discover something new." The average of the subjects' ratings were 3.00 for A-RS, 2.67 for A-Rank, and 4.22 for F-RS. From this viewpoint, the proposed system significantly outperformed A-RS and A-Rank with a significance level of 5%. This result indicates that the proposed system can provide serendipitous items related to "items that can attract the user's interest after being displayed by the system," which is one of the definitions of serendipitous items.

5. CONCLUSION

In this study, we improved upon our fusion-based recommender system based on the deeper idea of serendipity. This system possesses mechanisms that cause extrinsic and intrinsic accidents and enables users to derive some value from accidents through their sagacity. The key idea of the system is the fusion-based approach, through which the system mixes two user-input items to find new items that have the mixed features.

We experimentally evaluated the fusion-based recommender system through user tests using a real book data set from Rakuten Books. The experimental results showed the effectiveness of this system compared with the recommender systems used on the Amazon website from the viewpoint of serendipity. We would like to enhance its interfaces and make the fusion methods more intuitive and understandable for the users.

6. ACKNOWLEDGEMENT

This work was supported by a Grant-in-Aid for Young Scientists (B) (23700132).

7. REFERENCES

[1] J.L. Herlocker, J.A. Konstan, L.G. Terveen, and J.T. Riedl. Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems (TOIS), Vol. 22, No. 1, pp. 5–53, 2004.
[2] Tomoko Murakami, Koichiro Mori, and Ryohei Orihara. Metrics for evaluating the serendipity of recommendation lists. New Frontiers in Artificial Intelligence, pp. 40–46, 2008.
[3] Elizabeth Jamison Hodges. Three Princes of Serendip. Atheneum, 1964.
[4] Shigekazu Sawaizumi, Osamu Katai, Hiroshi Kawakami, and Takayuki Shiose. Use of serendipity power for discoveries and inventions. Intelligent and Evolutionary Systems, Studies in Computational Intelligence, Vol. 187, pp. 163–169, 2009.
[5] Royston M. Roberts. Serendipity: Accidental Discoveries in Science. Wiley, 1989.
[6] Kenta Oku and Fumio Hattori. Fusion-based recommender system for improving serendipity. In Proceedings of the Workshop on Novelty and Diversity in Recommender Systems (DiveRS 2011), at the 5th ACM International Conference on Recommender Systems (RecSys 2011), pp. 19–26, 2011.
[7] Cai-Nicolas Ziegler, Georg Lausen, and Lars Schmidt-Thieme. Taxonomy-driven computation of product recommendations. In Proceedings of the Thirteenth ACM conference on Information and knowledge management - CIKM '04, p. 406, New York, New York, USA, 2004. ACM Press.
[8] C.N. Ziegler, S.M. McNee, J.A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In Proceedings of the 14th international conference on World Wide Web, pp. 22–32, New York, New York, USA, 2005. ACM.
[9] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th international conference on World Wide Web, pp. 285–295, New York, New York, USA, 2001. ACM.
[10] Shlomo Berkovsky and Jill Freyne. Group-based recipe recommendations: analysis of data aggregation strategies. In Proceedings of the fourth ACM conference on Recommender systems, pp. 111–118. ACM, 2010.
[11] Y. Hijikata, T. Shimizu, and S. Nishida. Discovery-oriented collaborative filtering for improving user satisfaction. In Proceedings of the 13th international conference on Intelligent user interfaces, pp. 67–76. ACM, 2009.
[12] Mouzhi Ge, C. Delgado-Battenfeld, and D. Jannach. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In Proceedings of the fourth ACM conference on Recommender systems, pp. 257–260. ACM, 2010.
[13] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing, Vol. 7, No. 1, pp. 76–80, January 2003.

Case Study Evaluation of Mahout as a Recommender Platform

Carlos E. Seminario
Software and Information Systems Dept.
University of North Carolina Charlotte
cseminar@uncc.edu

David C. Wilson
Software and Information Systems Dept.
University of North Carolina Charlotte
davils@uncc.edu

ABSTRACT

Various libraries have been released to support the development of recommender systems for some time, but it is only relatively recently that larger scale, open-source platforms have become readily available. In the context of such platforms, evaluation tools are important both to verify and validate baseline platform functionality, as well as to provide support for testing new techniques and approaches developed on top of the platform. We have adopted Apache Mahout as an enabling platform for our research and have faced both of these issues in employing it as part of our work in collaborative filtering. This paper presents a case study of evaluation focusing on accuracy and coverage evaluation metrics in Apache Mahout, a recent platform tool that provides support for recommender system application development. As part of this case study, we developed a new metric combining accuracy and coverage in order to evaluate functional changes made to Mahout's collaborative filtering algorithms.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval–*Information filtering*

General Terms

Algorithms, Experimentation, Measurement

Keywords

Recommender systems, Evaluation, Mahout

1. INTRODUCTION

Selecting a foundational platform is an important step in developing recommender systems for personal, research, or commercial purposes. This can be done in many different ways: the platform may be developed from the ground up, an existing recommender engine may be contracted (e.g., OracleAS Personalization[1]), code libraries can be adapted, or a platform may be selected and tailored to suit (e.g., LensKit[2], MymediaLite[3], Apache Mahout[4], etc.). In some cases, a combination of these approaches will be employed.

For many projects, and particularly in the research context, the ideal situation is to find an open-source platform with many active contributors that provides a rich and varied set of recommender system functions that meets all or most of the baseline development requirements. Short of finding this ideal solution, some minor customization to an already existing system may be the best approach to meet the specific development requirements. Various libraries have been released to support the development of recommender systems for some time, but it is only relatively recently that larger scale, open-source platforms have become readily available. In the context of such platforms, evaluation tools are important both to verify and validate baseline platform functionality, as well as to provide support for testing new techniques and approaches developed on top of the platform. We have adopted Apache Mahout as an enabling platform for our research and have faced both of these issues in employing it as part of our work in collaborative filtering recommenders.

This paper presents a case study of evaluation for recommender systems in Apache Mahout, focusing on metrics for accuracy and coverage. We have developed functional changes to the baseline Mahout collaborative filtering algorithms to meet our research purposes, and this paper examines evaluation both from the standpoint of tools for baseline platform functionality, as well as for enhancements and new functionality. The objective of this case study is to evaluate these functional changes made to the platform by comparing the baseline collaborative filtering algorithms to the changed algorithms using well known measures of accuracy and coverage [6]. Our goal is not to validate algorithms that have already been tested previously, but to assess whether, and to what extent, the functional enhancements have improved the accuracy and coverage performance of the baseline out-of-the-box Mahout platform. Given the interplay between accuracy and coverage in this context, we developed a unified metric to assess accuracy vs. coverage trade-offs when evaluating functional changes made to Mahout's collaborative filtering algorithms.

[1]http://download.oracle.com/docs/cd/B10464_05/bi.904/b12102/1intro.htm
[2]http://lenskit.grouplens.org/
[3]http://www.ismll.uni-hildesheim.de/mymedialite/
[4]http://mahout.apache.org

2. RELATED WORK

Revisiting evaluation in the context of recommender platforms has received recent attention in the thorough evaluation of the LensKit platform using previously tested collaborative filtering algorithms and metrics, as reported in [2]. A comprehensive set of guidelines for evaluating recommender systems was provided by Herlocker et al [6]; these guidelines highlight the use of evaluation metrics such as accuracy and coverage and suggest the need for an ideal "general coverage metric" that would combine coverage with accuracy to yield an overall "practical accuracy" measure. Many of these evaluation metrics and techniques have also been covered recently in [12].

Recommender system research has been primarily concerned with improving recommendation accuracy [7]; however, other metrics such as coverage [10, 4] and also novelty and serendipity [6, 3] have been deemed necessary because accuracy alone is not sufficient to properly evaluate the system. Mcnee et al [7] states that recommendations that are most accurate according to the standard metrics are sometimes not the most useful to users and outlines a more user-centric approach to evaluation. The interplay between accuracy and other metrics such as coverage and serendipity creates trade-offs for recommender system implementers and this has been widely discussed in the literature, e.g., see [4, 3] and our previous work discussing trade-offs between accuracy and robustness [11].

3. SELECTING APACHE MAHOUT

To support our research in collaborative filtering, several recommender system platforms were surveyed, including LensKit, easyrec[5], and MymediaLite. We selected Mahout because it provides many of the desired characteristics required for a recommender development workbench platform. Mahout is a production-level, open-source, system and consists of a wide range of applications that are useful for a recommender system developer: collaborative filtering algorithms, data clustering, and data classification. Mahout is also highly scalable and is able to support distributed processing of large data sets across clusters of computers using Hadoop[6]. Mahout recommenders support various similarity and neighborhood formation calculations, recommendation prediction algorithms include user-based, item-based, Slope-One and Singular Value Decomposition (SVD), and it also incorporates Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) evaluation methods. Mahout is readily extensible and provides a wide range of Java classes for customization. As an open-source project, the Mahout developer/contributor community is very active; the Mahout wiki also provides a list of developers and a list of websites that have implemented Mahout[7].

3.1 Uncovering Mahout Details

Although Mahout is rich in documentation, there are implementation details on *how Mahout works* that could only be understood by looking at the source code. Thus, for clarity in evaluation, we needed to verify the implementation of baseline platform functionality. The following describes some of these details for Mahout 0.4 'out-of-the-box':

Similarity Weighting: Mahout implements the classic Pearson Correlation as described in [8, 5]. Similarity weighting is supported in Mahout and consists of the following method:

```
scaleFactor = 1.0 - count / (num + 1);
if (result < 0.0)
  result = -1.0 + scaleFactor * (1.0 + result);
else
  result = 1.0 - scaleFactor * (1.0 - result);
```

where *count* is the number of co-rated items between two users, *num* is the number of items in the dataset, and *result* is the calculated Pearson Correlation coefficient.

User-Based Prediction Algorithm: Mahout implements a Weighted Average prediction method similar to the approach described in [1], except that Mahout does *not* take the absolute value of the individual similarities in the denominator, however, it does ensure that the predicted ratings are within the allowable range, e.g., between 1.0 and 5.0.

Item-Based Prediction Algorithm: Mahout implements a Weighted Average prediction method. This approach is similar to the algorithm in [9], except that Mahout does *not* take the absolute value of the individual similarities in the denominator, however, it does ensure that the predicted ratings are within the allowable range, e.g., between 1.0 and 5.0. Also, Mahout does not provide support for neighborhood formation, e.g., similarity thresholding, for item-based prediction.

Accuracy Evaluation calculation: Mahout executes the recommender system evaluator specified at run time (MAE or RMSE) and implements traditional techniques found in [6, 12]. For MAE, this would be,

$$MAE = \frac{\sum_{i=1}^{n} \mid ActualRating_i - PredictedRating_i \mid}{n} \quad (1)$$

where n is the total number of ratings predicted in the test run.

3.2 Making Mahout Fit for Purpose

Through personal email communication with one of the Mahout developers, we were informed that Mahout intended to provide *basic* rating prediction and similarity weighting capabilities for its recommenders and that it would be up to developers to provide more elaborate approaches. Several changes were made to the prediction algorithms and the similarity weighting techniques for both the user-based and item-based recommenders in order to meet our specific requirements and to match the best practices found in the literature, as follows:

Similarity weighting: Defined as Significance Weighting in [5], this consists of the following method:

```
scaleFactor = count/50.0;
if (scaleFactor > 1.0)  scaleFactor = 1.0;
result = scaleFactor * result;
```

where *count* is the number of co-rated items between two users, and *result* is the calculated Pearson Correlation coefficient.

User-user mean-centered prediction: After identifying a neighborhood of similar users, a prediction, as documented in [8, 5, 1], is computed for a target item i and target user u as follows:

$$p_{u,i} = \overline{r_u} + \frac{\sum_{v \epsilon V} sim_{u,v}(r_{v,i} - \overline{r_v})}{\sum_{v \epsilon V} \mid sim_{u,v} \mid} \quad (2)$$

[5]http://easyrec.org/
[6]http://hadoop.apache.org/
[7]https://cwiki.apache.org/MAHOUT/mahout-wiki.html

where V is the set of k similar users who have rated item i, $r_{v,i}$ is the rating of those users who have rated item i, $\overline{r_u}$ is the average rating for the target user u over all rated items, $\overline{r_v}$ is the average rating for user v over all co-rated items, and $sim_{u,v}$ is the Pearson correlation coefficient.

Item-item mean-centered prediction: A prediction, as documented in [1], is computed for a target item i and target user u as follows:

$$p_{u,i} = \overline{r_i} + \frac{\sum_{j \epsilon N_u(i)} sim_{i,j}(r_{u,j} - \overline{r_j})}{\sum_{j \epsilon N_u(i)} \mid sim_{i,j} \mid} \quad (3)$$

where $N_u(i)$ is the set of items rated by user u most similar to item i, $r_{u,j}$ is u's rating of item j, $\overline{r_j}$ is the average rating for item j over all users who rated item j, $\overline{r_i}$ is the average rating for target item i, and $sim_{i,j}$ is the Pearson correlation coefficient.

Item-item similarity thresholding: This method was added to Mahout and used in conjunction with the item-item mean-centered prediction described above. Similarity thresholding, as described in [5], defines a level of similarity that is required for two items to be considered similar for purposes of making a recommendation prediction; item-item similarities that are less than the threshold are not used in the prediction calculation.

Coverage and combined accuracy/coverage metric: As suggested in [6], the easiest way to measure coverage is to select a random sample of user-item pairs, ask for a prediction for each pair, and measure the percentage for which a prediction was provided. To calculate coverage, code changes were made to Mahout to provide, for each test run, the total number of rating predictions requested that were unable to be calculated as well as the total of number of rating predictions requested that were actually calculated; the sum of these two numbers is the total number of ratings requested. Coverage was calculated as follows:

$$Coverage = \frac{Total\#RatingsCalculated}{Total\#RatingsRequested} \quad (4)$$

Code changes were also made to calculate a combined accuracy and coverage metric as defined in Section 4.

4. ACCURACY AND COVERAGE METRIC

The metrics selected for this case study, accuracy and coverage, were chosen because they are fundamental to the utility of a recommender system [10, 6]. Although other metrics such as novelty and serendipity can, and should, be used in conjunction with accuracy and coverage, our objective was to evaluate the very basic requirements of a recommender system. Our implementation of coverage, referred to as prediction coverage in [6], measures the percentage of a dataset for which the recommender system is able to provide predictions. High coverage would indicate that the recommender system is able to provide predictions for a large number of items and is considered to be a desirable characteristic of the recommender system [6]. A combination of high accuracy (low error rate) and high coverage are indeed desirable by users and system operators because it improves the utility or usefulness of the system from a user standpoint [10, 6].

What constitutes 'good' accuracy or coverage, however, has not been well defined in the literature: studies such as [10, 4, 5] and many others, endeavor to maximize accuracy (achieve lowest possible value) and/or coverage (achieve highest possible value) and view these metrics on a relative basis, i.e., how much the metric has increased or decreased beyond a baseline value based on empirical results. Furthermore, the interplay between accuracy and coverage, i.e., coverage decreases as a function of accuracy [4, 3], creates a trade-off for recommender system implementers that has been discussed previously but not been developed thoroughly. Inspired by the suggestion in [6] to combine the coverage and accuracy measures to yield an overall "practical accuracy" measure for the recommender system, we developed a straightforward "AC Measure" that combines both accuracy and coverage into a single metric as follows:

$$AC_i = \frac{Accuracy_i}{Coverage_i}, \quad (5)$$

where i indicates the ith trial in an evaluation experiment.

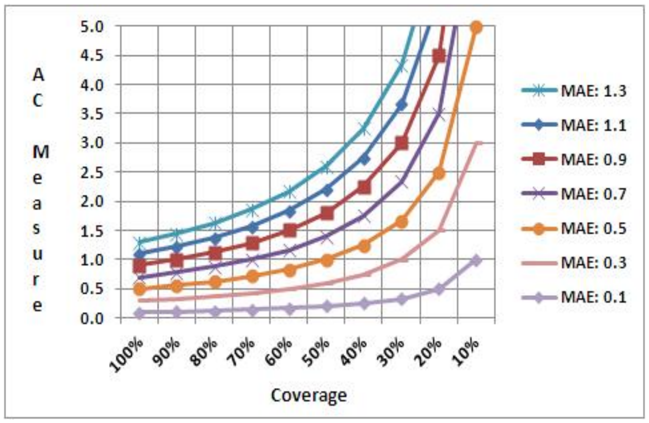


Figure 1: Illustration of the AC Measure

The AC Measure simply adjusts (upward) the Accuracy according to the level of Coverage metrics found in an experimental trial and is agnostic to the accuracy metric used, e.g., MAE or RMSE. Using a family of curves for the Mean Absolute Error (MAE) accuracy metric, Figure 1 illustrates the relationship between accuracy, coverage, and the AC Measure. As an example, following the "$MAE : 0.5$" curve we see that at 100% coverage, the AC Measure is 0.5, and at 10% coverage, the AC Measure has increased to 5. The intuition behind this metric is that when the recommender system is able to provide predictions for a high percentage of items in the dataset, the accuracy metric more closely indicates the level of system performance; conversely, when the coverage is low, the accuracy metric is "penalized" and is adjusted upwards. We believe that the major benefit of the AC Measure is that it formulates a solution for addressing the trade-off between accuracy and coverage and can be used to create a ranked list of results (low to high) from multiple experimental trials to find the best (lowest) AC Measure for each set of test conditions. The simplified visualization of the combined AC Measure shown in Figure 1 is an additional benefit. For our evaluation purposes, the use of a combined metric was ideal in addressing the inherent trade-offs between accuracy and coverage, especially in the cases where accuracy is found to be high when coverage is low; we posit that the AC Measure will also be useful for other researchers performing evaluations using accuracy and coverage.

5. EXPERIMENTAL DESIGN

The objective of this case study was to understand Mahout's baseline collaborative filtering algorithms and evaluate functional changes made to the platform using accuracy and coverage metrics. The main intent of making functional changes to Mahout recommender algorithms was to bring the Mahout algorithms in line with best practices found in the literature. Therefore, the overall hypothesis to be tested in this case study was that the modified algorithms improve Mahout's 'out-of-the-box' prediction accuracy for both user-based and item-based recommenders while maintaining reasonable coverage.

5.1 Datasets and Algorithms

The data used in this study were the MovieLens datasets downloaded from GroupLens Research[8]: the 100K dataset with 100,000 ratings for 1,682 movies and 943 users (referred to as ML100K in this study) and the 10M dataset with 10,000,000 ratings for 10,681 movies and 69,878 users (referred to as ML10M in this study). Ratings provided in these datasets consist of integer values between 1 (did not like) to 5 (liked very much).

For User-based (see §3.1), Mahout uses Pearson Correlation similarity (with and without similarity weighting), Neighborhood formation (similarity thresholding or kNN), and Weighted Average prediction. This was tested against a modified algorithm (see §3.2) consisting of Pearson Correlation similarity (with and without similarity weighting), Neighborhood formation (similarity thresholding or kNN), and Mean-centered prediction. For Item-based (see §3.1), Mahout uses Pearson Correlation similarity (with and without similarity weighting), no Neighborhood formation, and Weighted Average prediction. This was tested against a modified algorithm (see §3.2) consisting of Pearson Correlation similarity (with and without similarity weighting), Neighborhood formation (similarity thresholding), and Mean-centered prediction.

5.1.1 Test Cases

In order to test the overall hypothesis, the following test cases were developed and executed for both user-based and item-based recommenders using the ML100K and ML10M datasets:

1. Mahout Prediction, No weighting
2. Mahout Prediction, Mahout weighted
3. Mahout Prediction, Significance weighted
4. Mean-Centered Prediction, No weighting
5. Mean-Centered Prediction, Mahout weighted
6. Mean-Centered Prediction, Significance weighted

5.1.2 Accuracy and Coverage Metrics

We used Mahout's MAE evaluator to measure the accuracy of the rating predictions. For prediction coverage, we used dataset training data to estimate the rating predictions for the test set; the random sample of user-item pairs in our testing was 30K pairs for ML100K and 25K pairs for ML10M (see §3.2). AC Measures were calculated for all test cases.

5.1.3 Dataset Partitioning

The Mahout evaluator creates holdout [9] partitions according to a set of run-time parameters. For the tests using the ML100K dataset, the training set was 70% of the data, the test set was 30% of the data, and 100% of the user data was used; a total 30K rating predictions from 943 users were requested for each test set. For the tests using the ML10M dataset, the training set was 95% of the data, the test set was 5% of the data, and 5% of the user data was used; a total 25K rating predictions from 3180 users were requested for each test set.

5.1.4 Test Variations

Various similarity thresholds and kNN neighborhood sizes were executed for each test case in order to understand and evaluate the corresponding behavior of the recommenders. For User-based recommender testing, similarity thresholds of 0.0, 0.1, 0.3, 0.5, and 0.7 and kNN neighborhood sizes of 600, 400, 200, 100, 50, 20, 10, 5, and 2 were tested. For Item-based recommender testing, in addition to using no similarity thresholding, similarity thresholds of 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, and 0.7 were tested.

6. RESULTS AND DISCUSSION

6.1 ML10M Results

Figures 2 and 3 show the results of test cases 1 through 6 for user and item-based algorithms, respectively[10]. The key results of the experiment, for both user-based and item-based algorithms unless otherwise noted, were as follows:

1. MAE for mean-centered prediction with significance weighting is a significant improvement ($p<0.01$) over MAE for Mahout prediction, regardless of weighting, across similarity thresholds (except item-based at similarity threshold of 0.7) and kNN neighborhood sizes (except user-based at kNN of 2, not shown).

2. Mahout similarity weighting does not significantly improve ($p<0.01$) Mahout prediction MAE over prediction with no similarity weighting (except Mahout prediction for user-based and item-based at a similarity threshold of 0.4, not shown). This would indicate that Mahout similarity weighting is not very effective as a weighting technique, especially as compared to significance weighting.

6.2 ML100K Results

The results and trend lines for the ML100K experiment are similar to ML10M. The key results, for both user-based and item-based algorithms unless otherwise noted, were:

1. MAE for mean-centered prediction with significance weighting is a significant improvement ($p<0.01$) over MAE for Mahout prediction, regardless of weighting, across similarity thresholds and kNN neighborhood sizes (except user-based at kNN of 400).

2. Mahout similarity weighting does not significantly improve ($p<0.01$) Mahout prediction MAE over prediction with

[8]http://www.grouplens.org

[9]Holdout is a method that splits a dataset into two parts, a training set and a test set, and the partitioning is performed by randomly selecting some ratings from all, or some, of the users. The selected ratings constitute the test set, while the remaining ones are the training set.

[10]The following curves are superimposed over each other because the values are very similar: MAE results for mean-centered prediction (no weighting and Mahout weighted), MAE results for Mahout prediction (No weighting and Mahout weighted), Coverage results for Mahout prediction and mean-centered prediction (No weighting and Mahout weighted), Coverage results for Mahout prediction and mean-centered prediction (both Significance weighted).

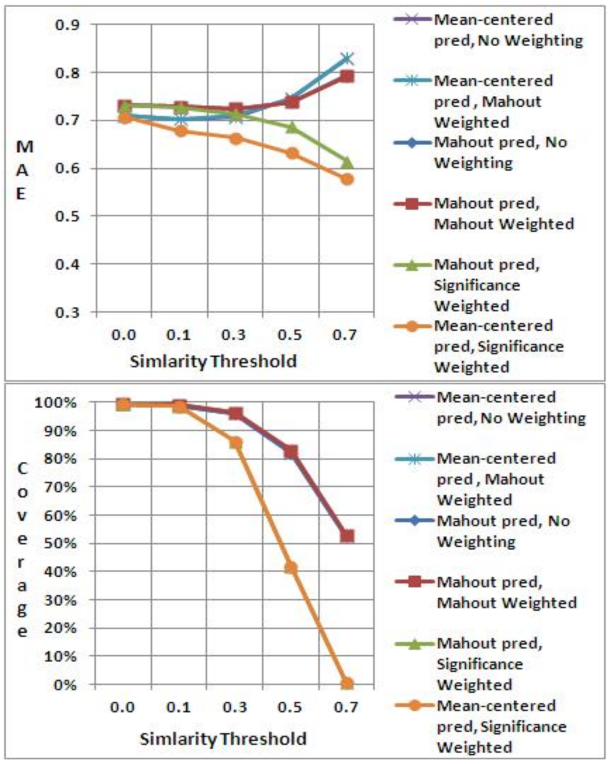


Figure 2: User-based Mahout Recommender Results for ML10M, Test cases 1 through 6

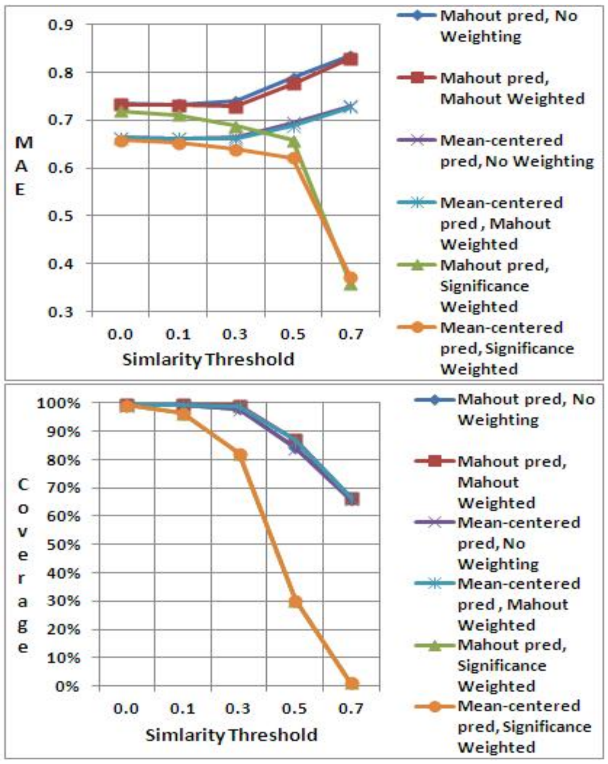


Figure 3: Item-based Mahout Recommender Results for ML10M, Test cases 1 through 6

no similarity weighting (except Mahout prediction for user-based and item-based at a similarity threshold of 0.4).

6.3 Discussion

As hypothesized, results for both of the ML100K and ML10M experiments show significant improvements in MAE using the mean-centered prediction algorithm with significance weighting compared to the Mahout baseline prediction algorithm. However, when coverage is considered, the "best" MAE results may need a second look. Can an MAE of 0.5 or less be considered "good" when the associated coverage is in the single digits? In this case, the recommender system may only be able to provide recommendations to a very small subset of its users and is a situation that must be avoided by system operators. To help address the accuracy vs. coverage trade-off, combined measures such as the AC Measure (Section 4), can help by considering both accuracy and coverage simultaneously. For the ML10M experiment, we determined that the lowest MAE for the User-based algorithm using mean-centered prediction with significance weighting was 0.578 at a similarity threshold of 0.7 and coverage of 0.833%; the AC Measure for this result is calculated as 69.42. Similarly, the lowest MAE for the Item-based algorithm using mean-centered prediction with significance weighting was 0.371 at a similarity threshold of 0.7 and coverage of 1.02%; the AC Measure for this result is calculated as 36.32. In each of these cases, the exceedingly high values for the AC Measure indicate that these results are not very desirable in a recommender system.

Figures 4 and 5 show the AC Measure results for user and item-based algorithms using ML10M, respectively. Rather than show all 30 results for each algorithm (5 similarity thresholds x 2 prediction methods x 3 weighting types), we show only the results with calculated AC Measure values less than 1.0; therefore, the lowest MAE results reported above for user-based and item-based algorithms are clearly beyond the range of this chart. We found that the best combined accuracy/coverage results were found at higher levels of coverage and lower levels of similarity threshold, i.e., the best (lowest) AC Measure for user-based was 0.688 at a similarity threshold of 0.1 and for item-based was 0.665 at a similarity threshold of 0.0, both using mean-centered prediction and significance weighting. We can also see that, with few exceptions, mean-centered prediction is improved over the Mahout prediction for the same similarity weighting and similarity threshold. We observed similar results using the ML100K dataset where the best (lowest) AC Measure for user-based was 0.765 and for item-based was 0.746, both at a similarity threshold of 0.0 and both using mean-centered prediction and significance weighting. These results demonstrate that the "best" MAE may not always be the lowest MAE, especially when coverage is also considered; furthermore, recommender system settings such as similarity weighting and neighborhood size also need to be considered during system evaluation.

Other observations of our experiments that match results reported in [5] and serve to validate our evaluation and increase our confidence in the results are: (a) In general, significance weighting improves prediction MAE, as compared to predictions using Mahout similarity weighting or no similar-

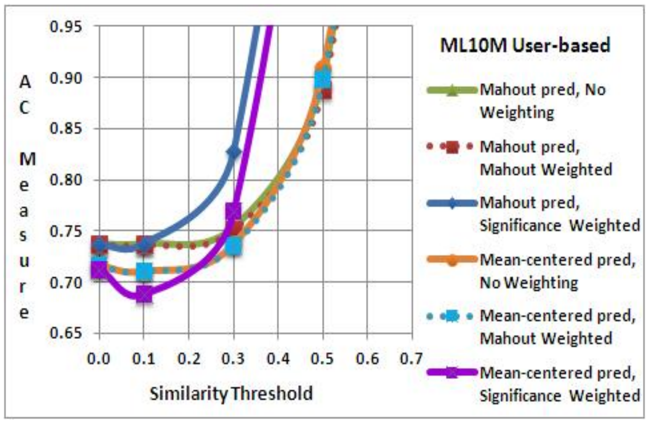


Figure 4: AC Measure for selected User-based results (lower is better)

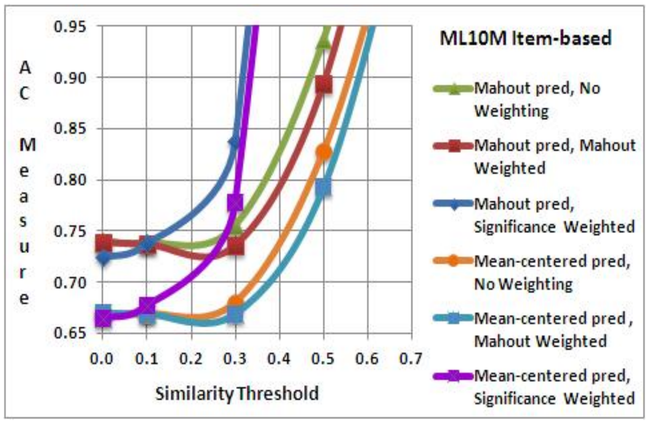


Figure 5: AC Measure for selected Item-based results (lower is better)

ity weighting; (b) As the similarity threshold increases, MAE for mean-centered prediction with significance weighting improves and coverage degrades, whereas MAE and coverage both degrade for Mahout prediction with Mahout weighting; (c) Coverage decreases as neighborhood size decreases.

7. CONCLUSION

Our case study of Mahout as a recommender system platform highlights evaluation considerations for developers and also shows how straightforward functional enhancements improves the performance of the baseline platform. We evaluated our changes against current Mahout functionality using accuracy and coverage metrics not only to assess baseline results, but also to provide a view of the trade-offs between accuracy and coverage resulting from using different recommender algorithms. We reported cases where the lowest MAE accuracy results were not necessarily always the 'best' when coverage results were also considered, and we instrumented Mahout for a combined accuracy and coverage metric (AC Measure) to evaluate these trade-offs more directly. We believe that this case study will provide useful guidance in using Mahout as a recommender platform, and that our combined measure will prove useful in evaluating algorithm changes for the inherent trade-offs between accuracy and coverage.

8. REFERENCES

[1] C. Desrosiers and G. Karypis. A comprehensive survey of neighborhood-based recommendations methods. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*. Springer, 2011.

[2] M. D. Ekstrand, M. Ludwig, J. A. Konnstan, and J. T. Riedl. Rethinking the recommender research ecosystem: Reproducibility, openness, and lenskit. In *Proceedings of the 5th ACM Recommender Systems Conference (RecSys '11)*, October 2011.

[3] M. Ge, C. Delgado-Battenfeld, and D. Jannach. Beyond accuracy: Evaluating recommender systems by coverage and serendipity. In *Proceedings of the 4th ACM Recommender Systems Conference (RecSys '10)*, September 2010.

[4] N. Good, J. B. Schafer, J. A. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*, July 1999.

[5] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the ACM SIGIR Conference*, 1999.

[6] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.

[7] S. Mcnee, J. Riedl, and J. Konstan. Accurate is not always good: How accuracy metrics have hurt recommender systems. In *Proceedings of the Conference on Human Factors in Computing Systems(CHI 2006)*, April 2006.

[8] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the ACM CSCW Conference*, 1994.

[9] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the World Wide Web Conference*, 2001.

[10] B. M. Sarwar, J. A. Konstan, A. Borchers, J. Herlocker, B. Miller, and J. Riedl. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *Proceedings of the ACM 1998 Conference on Computer Supported Cooperative Work (CSCW '98)*, November 1998.

[11] 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)*, May 2012.

[12] G. Shani and A. Gunawardana. Evaluating recommendation systems. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*. Springer, 2011.

Evaluating Various Implicit Factors in E-commerce

Ladislav Peska
Department of software engineering
Charles University in Prague
Malostranske namesti 25, Prague, Czech Republic
peska@ksi.mff.cuni.cz

Peter Vojtas
Department of software engineering
Charles University in Prague
Malostranske namesti 25, Prague, Czech Republic
vojtas@ksi.mff.cuni.cz

ABSTRACT

In this paper, we focus on the situation of a typical e-commerce portal employing personalized recommendation. Such website could, in addition to the explicit feedback, monitor many different patterns of implicit user behavior – implicit factors. The problem arises while trying to infer connections between observed implicit behavior and user preferences - while some connections are obvious, others may not.

We have selected several often used implicit factors and conducted online experiment on travel agency web site to find out which implicit factors could replace explicit ratings and (if there are more of them) how to combine their values. As utility functions determining recommending efficiency was selected click through rate and conversions rate.

Our experiments corroborate importance of considering more implicit factors and their different weights. The best individual results were achieved by means of the *scrolling* factor, the best combination was *Prior_to* method (lexicographical ordering based on factor values).

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval - *Information Filtering*

General Terms

Measurement, Human Factors.

Keywords

Recommender systems, implicit factors, user feedback, e-commerce success metrics

1. INTRODUCTION

Recommending on the web is both an important commercial application and popular research topic. The amount of data on the web grows continuously and it is nearly impossible to process it directly by a human. The keyword search engines were adopted to cope with information overload but despite their undoubted successes, they have certain limitations. Recommender systems can complement onsite search engines especially when the user does not know exactly what he/she wants. Many recommender systems, algorithms or methods have been presented so far. We can mention Amazon.com recommender [12] as one of the most popular commercial examples. Recommender systems varies in both type (Collaborative, Content-based, Context, hybrid, etc.), input (user feedback types, object attributes, etc.) or output. We suggest [17] for detailed recommender systems taxonomy.

The explicit feedback (given by the user consciously e.g. rating objects with stars) is often used in research and also in some commercial applications. Although it is quite easy to understand and refers very well to the user's preference, it also has drawbacks. The biggest ones are its scarcity and unwillingness of some users to provide any explicit feedback [7]. Contrary to the explicit feedback, the implicit feedback (events triggered by a user unconsciously) can provide abundant amount of data, but it is much more difficult to understand the true meaning of such feedback.

The rest of the paper is organized as follows: review of some related work is in section 2. In section 3 we describe our model of user preferences and in section 4 method how to learn it. Section 5 contains results of our online experiment on a travel agency website. Finally section 6 concludes our paper and points to our future work.

1.1 Motivation

In this paper we focus on an e-commerce website employing personalized object recommendation – e.g. travel agency. On such site we can record several types of user implicit feedback such as page-view, actions or time spent on page, purchasing related actions, click through or click stream, etc. Each of these factors is believed to be related to the user's preference on an object. However this relation can be non-trivial, dependant on other factors, etc. In this work, we focus on if and how such relations could be compared against each another. Our second aim is how to use or combine them in order to improve recommendations.

1.2 Contribution

The main contributions of this paper are:

- Evaluation of recommendation based on various implicit factors using typical e-commerce success metrics.
- A generic model that combines various types of user feedback.
- Experiments with several combining methods (average, weighted aggregation and prioritization).
- Gathered data for possible future off-line experiments.

2. RELATED WORK

The area of recommender systems has been extensively studied recently. Much effort has been made for creating different recommendation algorithms e.g. [3], [4], [5] and designing whole recommender systems e.g. [6], [15] and [16]. Our work is prependable to some of those systems as we can supply them with a single-value object rating based on more implicit factors instead of using explicit user's object rating or only single implicit factor.

A lot of recommendation algorithms aims to do decompose the user's preference on the object into the preference of the object's attributes [3], [4], [5] and [15], which can be a future extension to our work.

Some authors employ context information while deciding about true meaning of the user feedback e.g. Eckhardt et al. [2] proposes that good rating of an object is more relevant when the object appears among other good objects. Joachims et al. [8] proposes "Search Engine Trust Bias" while observing that the first result of a search engine search has higher click through rate than the second one, even if the results were swapped – so the less relevant result was shown at the first place.

Important for our research is the work of Kiessling et al. on the Preference SQL system e.g. [10]. The Preference SQL is an extension of SQL language allowing user to specify directly preferences (or so called "soft constraints") and to combine them in order to receive best objects. We use three described combination operators: *Prior to (hierarchical), Ranking* and *Pareto* in our model of user preference.

Several authors studied various aspects of implicit feedback: quite common are studies about comparing implicit and explicit feedback e.g. Claypool et al. [1] using adapted web browser or Jawaheer et al. [7] on an online music server. Using only an implicit feedback based utility function is a common approach when it is impossible to get explicit feedback [6], [14]. Lee and Brusilovsky proposed job recommender directly employing negative implicit feedback [11]. In our case we have focused on e-commerce recommenders, so we have used two typical e-commerce utility functions – *Click Through Rate* and user *Conversion Rate*. In contrast to several studies e.g. [1] who studied behavior of closed, small group of users (who installed special browser) on the open web, we have focused on the single website and all its users which in result let us to gather more feedback data and introduce more various feedback factors.

For our experiments, we use the UPComp [13] recommender deployable into the running e-commerce applications. Compared to our previous work [14], we have conducted larger on-line experiment, revised utility functions in our learning method and introduced new model of user preference.

3. MODELS OF USER PREFERENCE

We assume that any feedback is in the form *Feedback(user, object, feedback type, value)*. At this stage of our research, we do not employ preference relations or feedback related to the object groups (e.g. categories) and object attributes.

We based our models on work of Kiessling et al. and their model of user preferences in Preference SQL [10]. The authors defined several patterns on how to express preferences (soft conditions) on a single attribute e.g. "prize ***around*** 2000" or "***Highest*** distance", etc. Each soft condition assigns to each object value from [0, 1] interval. Then they defined three types of operators combining soft conditions together:

- *Preferring Operator*: preferring one (or more) condition against others.
- *Ranking Operator* to combine conditions by a ranking function. At this time we use weighted average as a ranking.
- *Pareto Operator* for combining equally important conditions, or conditions where their relation is unknown. We plan to use this operator in our future work.

In our research, we have replaced the soft conditions by the implicit factors forming the *Preference algebra model*. Each implicit factor value has assigned preference value from [0, 1] interval – currently we simply linearly normalize the space between highest and lowest factor values. Those preference values can be then freely combined with the operators e.g.:

Scrolling PRIOR TO Avg(*Time*, *MouseClicks*)

We will demonstrate behavior of our model on a small two-dimensional example: Table 1 contains four sample objects and their scrolling and time on page feedback for fixed user (data already normalized into [0, 1]). They are visualized on Figure 1: as it can be seen, we will receive different top-k for their various combinations.

Table 1: example objects and their scrolling and time on page implicit factor values.

Object	Amount of scrolling	Time on page
Object1	1.0 (e.g 10 times)	0.4 (e.g. 200sec)
Object2	0.7 (e.g 7 times)	1.0 (e.g. 500sec)
Object3	0.8 (e.g 8 times)	0.6 (e.g. 300sec)
Object4	0.4 (e.g 4 times)	0.3 (e.g. 150sec)

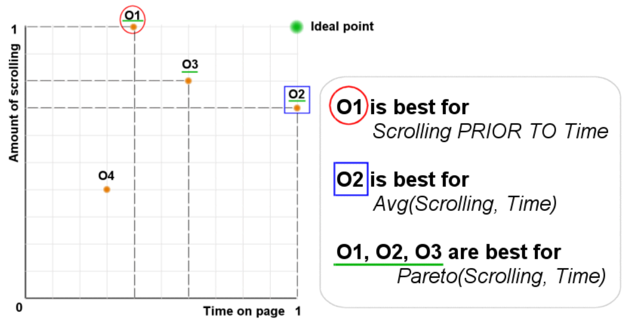


Figure 1: Combining single implicit factor values into the preference for objects from Table 1.

4. LEARNING PREFERENCE MODEL

The idea behind our learning model is following: If we use a fixed recommendation methods supplied with various implicit factor data and then compare the effectivity of the recommendations, we can estimate how successful each implicit factor is.

For the purpose of our experiment, we have divided our learning model into two phases: in the first phase, we have learned successfulness of the considered implicit factors (see Table 2 for their list and description). In the second phase we have implemented several methods combining various implicit factors together based on the *Preference algebra model*.

Table 2: Description of the considered implicit factors for arbitrary fixed user and object

Factor	Description
PageView	Count(*OnLoad()* event on object detail page)
MouseActions	Count(*OnMouseOver()* events on object detail page)
Scroll	Count(OnScroll() events on object detail page)
TimeOnPage	Sum(time spent on object detail page)
Purchase	Count(Object was purchased)
Open	Count(Object detail page accessed via link from recommending area)
Shown	Count(Object shown in recommending area)

In both phases we have measured success of the recommendations according to the two widely used e-commerce success metrics:

- Conversion rate - #buyers / #users
- Click through rate (CTR) - #click through / #shown objects by the recommending method

As we stand on the side of the e-shop owner, we determine that the main task for the recommender system is to increase the shop owner's profit. It is possible to measure the profit directly as an utility function, however we did reject this method for now and use only conversion rate measuring overall goal (purchase) achievements. In this stage of our work we mainly focus on convincing user to buy any product rather then convince him/her to buy product B instead of A (see table 3 – the overall conversion rates are rather low and need to be improved prior to the other goals).

As the conversion rate should evaluate the overall success of the whole system, the CTR refers directly to the success of the recommendation itself.

5. EXPERIMENT

We have conducted an online experiment on the SLAN tour travel agency website[1] to confirm our ideas. We have exchanged the previous random recommendations on the category pages for our methods. The experiment lasted for 2 months in February and March 2012. We have collected data from in total 15610 unique users (over 200 000 feedback events). We first describe in Figure 2 the simplified diagram of the travel agency e-shop. We recognize four important states of user interaction with the e-shop:

- User is creating conjunctive query *Q* (either implicitly e.g. by viewing category pages or explicitly via search interface).
- The (possibly very large) set of objects *OQ* is response to *Q*. The objects are recommended at this state. We recommend some objects from *OQ* to the user (membership in *OQ* set is necessary condition, each recommended object from *OR* has to fulfill).
- User is viewing detail of the selected object *o*. We believe that most of the interesting user feedback should be recorded in this phase.
- User purchased the object *o*, *w*hich is the criterion of success for us.

[1] http://www.slantour.cz

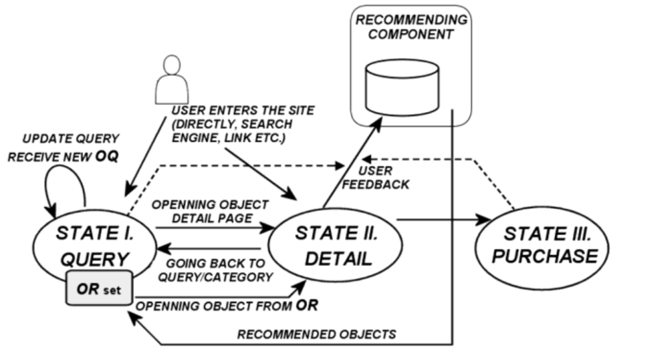

RECOMMENDING COMPONENT
USER ENTERS THE SITE (DIRECTLY, SEARCH ENGINE, LINK ETC.)
UPDATE QUERY RECEIVE NEW OQ
USER FEEDBACK
OPENNING OBJECT DETAIL PAGE
STATE I. QUERY
OR set
STATE II. DETAIL
STATE III. PURCHASE
GOING BACK TO QUERY/CATEGORY
OPENNING OBJECT FROM OR
RECOMMENDED OBJECTS

Figure 2: The simplified state diagram of an e-commerce site: User enters the site in *STATE I.* or *II*. He/she can either navigate through category or search result pages – updating query *Q*, receiving new recommended objects *OQ* and *OR* (*STATE I.*) or proceeds to the detail of an object (*STATE II.*). The object can be eventually purchased (*STATE III.*).

The Figure 3 depicts the schema of our experiment.

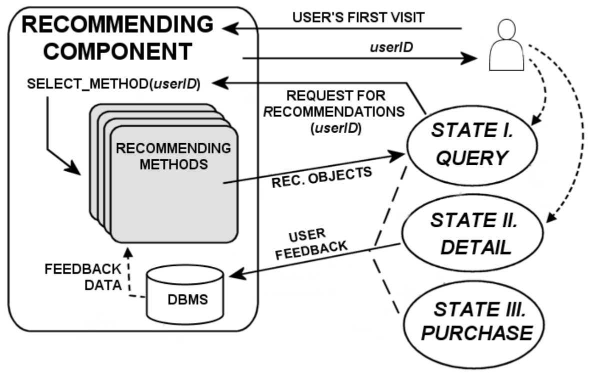


Figure 3: General schema of our experiment. When user visits the website for the first time, he receives *userID*, whenever he access page with recommendations, the component selects the recommending method according to the *userID.* The experiment results for each method are computed from user feedback (Click throughs, purchases).

5.1 UPComp recommender

The UPComp (user preference component) is an independent e-commerce recommender. It consists of a database layer storing user feedback, server-side computing user preference and recommendations and client-side which captures the user events and shows recommended objects. Among UPComp main advantages belong:

- Easy deployable to a various e-commerce systems regardless to the domain of objects.
- Large (extendible) set of recorded user behavior.
- Several recommending methods which can be combined together.

In the current experimental setting, we have used only a small portion of UPComp capabilities (*ObjectRating* and *Collaborative* methods, recommending objects for known category). For more complex description see [13].

5.2 Single implicit factors

For the first learning phase we have created a total of seven variants of *ObjectRating* recommending method, each based on one implicit factor (*PageView()*, *MouseActions()*, *Scrolling()*, *TimeOnPage()*, *Purchases()*, *ClickThrough()* and *ClickThrough()/Shown() rate*). Each variant of *ObjectRating* method used the same recommendation algorithm, but based on only one feedback type data. We have also added *Random()* method recommending random objects from the current category as a baseline. Each unique user received recommendations based only on one of these methods all the time he visited the website. The method is determined as *userID* mod *K*, where *K* is number of possible methods.

The *ObjectRating* method calculates for each object (*o*) the object rating as the sum of feedback values of given type (*f*) from all users *U*. The score is then normalized into [0, 1] (see pseudo SQL code below).

```
SELECT (SUM(value) / MAX(SUM(value)) as ObjectRating
        FROM Feedback
        WHERE Object = o and FeedbackType = f
```

We have selected this simple method, because we wanted to avoid the problems suffered by more complex methods (e.g. *Cold Start Problem*). On the other hand, this decision decreases variability of recommendations, so we want to use also other methods in our future work.

Table 3. shows results of the first phase of our experiment. Anova test proves statistically significant differences in Click through rate (p-value < 0.001), but the differences in the Conversion rate were not statistically significant (probably due to relatively small number of purchases – 106 buyers in total).

Rather surprising is the supreme position of the *Scrolling()* method comparing to the e.g. Claypool et al. [1]. However in contrast to the Claypool et al. the most of our object detail pages overflows typical browser visible area. However important controls like purchase button are visible in top of the page, scrolling is necessary to see some additional information like accommodation details, all hotel pictures, trip program, etc. On sites with bookmark-style design with no or a little scrolling needs, opening an in-page bookmark should be considered as a similar action to our scrolling event. Also time spent on page seems to improve recommendations (despite the results of e.g. Kelly and Belkin [9]).

Table 3. Results of the experiment's first phase. ***** significant improvement over *Random()* (TukeyHSD, 95% confidence).

Method	Conversion rate	Click through rate (CTR)
Random() (baseline)	0.97%	3.02%
PageView()	1.34%	4.11%*
MouseActions()	0.96%	4.15%*
TimeOnPage()	1.71%	4.50%*
Scrolling()	1.98%	4.94%*
Purchases()	1.39%	4.06%
ClickThrough()	0.84%	4.32%*
ClickThrough/Shown()	1.70%	4.38%*

5.3 Combining implicit factors

Following to the first phase, we have defined our three main tasks and perform experiments to receive at least initial answers/results for them:

T1. Measure whether combined methods produce better recommendations than the single-factor ones.

T2. Measure whether various combination functions affect recommendation effectivity.

T3. How to use our results in more complex recommending methods.

Table 4: Results of combined methods: *AVG* stands for average, in *Weighted_AVG* we use the factor's placement in the CTR results as weight, similarly *Prior_to* prioritize first factor against second, etc. * significant improvement over *Random()* (TukeyHSD). ** significant impr. over *AVG(best 3 factors)* (TukeyHSD). *** significant impr. over *Scrolling()* (t-test).

Method	Conversion rate	CTR
Random() (baseline1)	0.97%	3.19%
Scrolling() (baseline2)	1.07%	4.36% *
AVG(all factors)	1.41%	4.54% *
AVG(best 3 factors)	1.35%	3.95%
Weighted_AVG(best 3 factors)	1.49%	4.95% *,**
Prior_to(best 3 factors)	1.05%	5.12% *,**,***
Collaborative+ Weighted_AVG (all factors)	0.95%	4.64% *

Again Conversion rate unfortunately did not provide us with any significant results, so we have focused on the CTR. The combined methods overall achieved better results than the *Scrolling()*, but only the *Prior_to()* was significantly better. Almost every method outperforms *Random()* recommendation.

For the Task 2, we have compared Weighted average, Priorization and Average methods on the best three implicit factors, where both *Weighted average* and *Priorization* methods receives significantly better results than *Average* in Click through rate. Both Prior_to and Weighted_AVG significantly outperformed AVG method, from which can be concluded that there are important differences in various single implicit factors performance and that combination function should weight somehow the single factors performance. However even though the *Prior_to* CTR results were better than *Weighted_AVG*, the difference was not significant enough, so we can not yet make a conclusion about which combination method is the best.

For the third task, we have slightly changed our experiment schema (see Figure 2), where we have exchanged the *ObjectRating()* method for *UserObjectRating(User, Object, Feedback type)* calculating object rating separately for each relevant user (see pseudo SQL code below).

```
SELECT (SUM(value) / MAX(SUM(value)) as ObjectRating
  FROM Feedback
  WHERE User = u and Object = o and FeedbackType = f
```

UPComp then calculated standard user-to-user collaborative filtering. The method results (see Table 4, *Collaborative+ Weighted_AVG*) were though rather moderate. The method outperforms *AVG*, *Scrolling* and *Random* in CTR, however the difference was not significant enough and other simple methods (e.g. *Prior_to*) achieved better results. One of the possible problems was the higher computational complexity of this method resulting in higher response time which could reduce the user's interest in the objects presented in recommending area. This method can be in future compared / replaced with e.g. object-to-object collaborative filtering with precomputed similarity as described in [12].

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have discussed the problem of using more various implicit factors and how to formulate user's preference from them. We have adapted the *Preference algebra model* to this task, selected several possibly good implicit factors and organized a small online experiment to verify our ideas. The experiment results showed that the most of our proposed factors outperforms baseline recommendation and that it is important to use more various implicit factors combined accordingly to their performance.

The usage of e-commerce success metrics (especially CTR) to determine success of recommendations provided us with interesting results, so we plan to continue using Click through rate as a success metrics (conversions due to the relatively small number of purchases only in large scale experiments).

Our research on this field is in its early stage, so there is both space for more experiments (e.g. with negative implicit feedback, dependencies between various factors, temporal aspect of user's preference and behavior, etc.) and for possible improvements in our experimental settings (e.g. replacing recommending methods, extend the implicit factors set, etc.).

However our main task should be to move from such experiments into a working recommender system based on implicit preferences with various (dynamic) importances.

7. ACKNOWLEDGMENTS

The work on this paper was supported by Czech projects SVV-2012-265312, MSM 0021620838 and GACR 202-10-0761.

REFERENCES

[1] Mark Claypool, Phong Le, Makoto Wased, and David Brown. 2001. Implicit interest indicators. *In Proceedings of the 6th international conference on Intelligent user interfaces* (IUI '01). ACM, New York, NY, USA, 33-40.

[2] Eckhardt A., Horváth T., Vojtáš P.: PHASES: A User Profile Learning Approach for Web Search. *In Proc. of WI 2007,* Silicon Valley, CA, IEEE Computer Society, pp. 780-783

[3] Alan Eckhardt, Peter Vojtáš: Combining Various Methods of Automated User Decision and Preferences Modelling. MDAI '09 172-181. Springer-Verlag Berlin, Heidelberg, 2009.

[4] Alan Eckhardt, Peter Vojtáš. 2009. How to learn fuzzy user preferences with variable objectives. In proc. of IFSA/EUSFLAT Conf. 2009: 938-943.

[5] Jill Freyne, Shlomo Berkovsky, and Gregory Smith. 2011. Recipe recommendation: accuracy and reasoning. In *Proceedings of the 19th international conference on User modeling, adaption, and personalization* (UMAP'11). Springer-Verlag, Berlin, Heidelberg, 99-110.

[6] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *Proc. of ICDM '08*. IEEE Computer Society, Washington, DC, USA, 263-272.

[7] Gawesh Jawaheer, Martin Szomszor, and Patty Kostkova. 2010. Comparison of implicit and explicit feedback from an online music recommendation service. In *Proc. of* HetRec'10. ACM, New York, NY, USA, 47-51.

[8] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. 2007. Evaluating the accuracy of implicit feedback from clicks and query reformulations in Web search. *ACM Trans. Inf. Syst.* 25, 2, Article 7 (April 2007). DOI=10.1145/1229179.1229181

[9] Kelly, D. & Belkin, N. J. Display time as implicit feedback: understanding task effects *Proceedings of the 27th ACM SIGIR conference on Research and development in information retrieval*, ACM, 2004, 377-384

[10] Kießling, W.; Endres, M. & Wenzel, F. The Preference SQL System - An Overview. IEEE Data Eng. Bull., 2011, 34, 11-18

[11] Lee, D. H. & Brusilovsky, P. Reinforcing Recommendation Using Implicit Negative Feedback *In Proc. of UMAP 2009*, Springer, LNCS, 2009, 422-427

[12] Linden, G.; Smith, B. & York, J. Amazon.com recommendations: item-to-item collaborative filtering *Internet Computing,* IEEE, 2003, 7, 76 - 80

[13] Ladislav Peska, Alan Eckhardt, and Peter Vojtas. 2011. UPComp - A PHP Component for Recommendation Based on User Behaviour. *In Proceedings of WI-IAT '11*, IEEE Computer Society, Washington, DC, USA, 306-309.

[14] Ladislav Peska and Peter Vojtas. 2012. Estimating Importance of Implicit Factors in E-commerce. *To appear on WIMS 2012*, http://ksi.mff.cuni.cz/~peska/wims12.pdf

[15] Pizzato, L.; Rej, T.; Chung, T.; Koprinska, I. & Kay, J. RECON: a reciprocal recommender for online dating *Proc. of RecSys'10*, ACM, 2010, 207-214

[16] Symeonidis, P.; Tiakas, E. & Manolopoulos, Y. Product recommendation and rating prediction based on multi-modal social network. *Proc. of RecSys'11*, ACM, 2011, 61-68

[17] Bo Xiao and Izak Benbasat. 2007. E-commerce product recommendation agents: use, characteristics, and impact. *MIS Q.* 31, 1 (March 2007), 137-209.