Proceedings of the

Workshop on the Practical Use of Recommender Systems, Algorithms and Technologies (PRSAT 2010)

held at the

4th ACM Conference on Recommender Systems (RecSys 2010)

30 September 2010

Barcelona, Spain

Jérôme Picault[1], Dimitre Kostadinov[1], Pablo Castells[2], Alejandro Jaimes[3]

[1] Bell Labs, Alcatel-Lucent, France

[2] Universidad Autónoma de Madrid, Spain

[3] Yahoo! Research, Spain

Preface

User modeling, adaptation, and personalization techniques have hit the mainstream. The explosion of social network websites, on-line user-generated content platforms, and the tremendous growth in computational power of mobile devices are generating incredibly large amounts of user data, and an increasing desire of users to "personalize" (their desktop, e-mail, news site, phone). The potential value of personalization has become clear both as a commodity for the benefit or enjoyment of end-users, and as an enabler of new or better services – a strategic opportunity to enhance and expand businesses.

An exciting characteristic of recommender systems is that they draw the interest of industry and businesses while posing very interesting research and scientific challenges. In spite of significant progress in the research community, and industry efforts to bring the benefits of new techniques to end-users, there are still important gaps that make personalization and adaptation difficult for users. Research activities still often focus on narrow problems, such as incremental accuracy improvements of current techniques, sometimes with ideal hypotheses, or tend to overspecialize on a few applicative problems (typically TV or movie recommenders – sometimes simply because of the availability of data). This restrains de facto the range of other applications where personalization technologies might be useful as well.

This workshop contrived for a new uptake on past experiences and lessons learned. We proposed an analytic outlook on new research directions, or ones that still require substantial research, with a special focus on their practical adoption in working applications, and the barriers to be met in this path.

The topics of interest were related to:

- Limits of recommender systems: main bottlenecks, research dead ends and myths in recommender systems; missing technology pieces for wider adoption; social (privacy, culture) issues
- Analytical view of personalization experiences: case studies of recommender system implementations & deployments; evaluation and user studies of recommender systems; scalability in large recommender systems; lessons learnt from your past experience; obstacles to massive deployment of recommendation solutions in industrial environments
- Recommendation in broader systems
- Next needs in recommender systems: new business models related to recommendation; new paradigms to provide recommendations; new areas for recommendations; users' expectations about future recommender systems

This workshop brought together approximately 45 researchers and practitioners (including people from Microsoft, Amazon, Bloomberg, Telefónica, Netflix, Hitachi, eBay, YouTube, Strands, IBM, and many SMEs). Twelve papers were submitted to the workshop; nine were accepted, illustrating different facets of recommender systems that are important for a wider adoption, such as bringing more "realistic" algorithms that cope with problems related to feedback elicitation and serendipity, scalability, openness, handling multiple users, etc. The discussions sessions tried to confront the points of view of researchers and industry w.r.t. recommender systems. Several gaps in terms of concerns have been identified, among which user interfaces, scalability, and real-time issues, which are still under-represented topics in the research community.

Jérôme Picault
Dimitre Kostadinov
Pablo Castells
Alejandro Jaimes

Organising Committee

Jérôme Picault	Bell Labs, Alcatel-Lucent, France
Dimitre Kostadinov	Bell Labs, Alcatel-Lucent, France
Pablo Castells	Universidad Autónoma de Madrid, Spain
Alejandro Jaimes	Yahoo! Research, Spain

Programme Committee

David Bonnefoy	Pearltrees
Makram Bouzid	Alcatel-Lucent Bell Labs
Iván Cantador	Universidad Autónoma de Madrid
José Carlos Cortizo	Universidad Europea de Madrid & BrainSins
Alexander Felfernig	Graz University of Technology & ConfigWorks
Ido Guy	IBM Haifa Research Lab
Paola Hobson	Snell
Rubén Lara	Telefónica I+D
Kevin Mercer	BBC
Andreas Nauerz	IBM Deutschland Research & Development GmbH
Michael Papish	Media Unbound, Inc.
Igor Perisic	LinkedIn Corporation
Myriam Ribière	Alcatel-Lucent Bell Labs
Neel Sundaresan	eBay Research Labs
Marc Torrens	Strands, Inc.
Andreas Töscher	Commendo Research & Consulting GmbH
Xiaohui Xue	SAP

Table of Contents

Top 10 Lessons Learned Developing and Deploying Real World Recommender Systems

Marc Torrens
Strands, Inc.
torrens@strands.com

ABSTRACT

Strands develops products that help people find information online that they want and need. Strands offers production recommendation services for eCommerce, interactive tools for personal finance management, and personal interest and lifestyle-oriented social discovery solutions. Strands also operates moneystrands.com, a personal finance management platform, and strands.com, a training log and information source for active people. In this talk, Strands Chief Innovation Officer, Marc Torrens, PhD, will discuss Strands' "Top 10 Lessons Learned" from our experience building recommender systems and interactions with customers deploying our systems. The lessons learned will range from customer relations and marketing ("It must make 'strategic' sense"), to business planning ("Don't wait too long to get ready to scale"), to technical ("Cold start? Be Creative"). As recommender technology becomes ubiquitous online, and even overshadows search in many commercials settings, Strands has found these "Top 10 Lessons Learned" continue to be valuable guidelines.

Proposal and Evaluation of Serendipitous Recommendation Method Using General Unexpectedness

Takayuki Akiyama
Hitachi, Ltd., Central Research Laboratory
1-280, Higashi-Koigakubo,
Kokubunji-shi, Tokyo
185-8601 Japan
Tel: +81-42-323-1111 ext. 4302
takayuki.akiyama.hv@hitachi.com

Kiyohiro Obara
Hitachi, Ltd., Central Research Laboratory
1-280, Higashi-Koigakubo
Kokubunji-shi, Tokyo
185-8601 Japan
Tel: +81-42-323-1111 ext. 3612
kiyohiro.obara.pc@hitachi.com

Masaaki Tanizaki
Hitachi, Ltd., Central Research Laboratory
1-280, Higashi-Koigakubo
Kokubunji-shi, Tokyo
185-8601 Japan
Tel: +81-42-323-1111 ext. 4068
masaaki.tanizaki.tj@hitachi.com

ABSTRACT
Recommender systems support users in selecting items and services in an information-rich environment. Although recommender systems have been improved in terms of accuracy, such systems are still insufficient in terms of novelty and serendipity, giving unsatisfactory results to users. Two methods of "serendipitous recommendation" are therefore proposed. However, a method for recommending serendipitous items accurately to users does not yet exist, because what kinds of items are serendipitous is not clearly defined. Accordingly, a human preference model of serendipitous items based on actual data concerning a user's impression collected by questionnaires was devised. Two serendipitous recommendation methods based on the model were devised and evaluated according to a user's actual impression. The evaluation results show that one of these recommendation methods, the one using general unexpectedness independent of user profiles, can recommend the serendipitous items accurately.

Categories and Subject Descriptors
H.1.2 [**Models and Principles**]: User/Machine Interface – *Human factors, Human information processing.*

General Terms
Human Factors

Keywords
Recommender systems, user preference, content-based, serendipity, unexpected.

1. INTRODUCTION
In recent years, the amount of information accessible to users is increasing and becoming more diversified because of the growth of information technology and the expansion of commercial use of IT. Under this circumstance, although users can select various items (such as information, TV programs, and books) they cannot select the best of those items from a vast amount of items including many useless items.

To solve this problem, so-called "recommender systems"—for recommending suitable items to users by monitoring a user's action and extracting information concerning a user's preferences—are becoming necessary for "item-providing services" such as internet shopping sites and department stores. In the future, recommender systems will recommend items by monitoring all a user's preferences. Users will get information suitable for their needs, and they will have an opportunity to discover new items. Moreover, service providers will be able to provide services continuously because users will use their systems more frequently.

Recommendation technology is one way to retrieve information that suits a user's preferences. In information-retrieval theory, useful information is categorized as two types: that which users recognize as useful, and that which users do not recognize as useful but is actually useful [1]. We suppose that the items users like are categorized as the same two types; accordingly, in this paper, the second type of items is defined as "serendipitous items."

In general, typical recommender systems use either of two strategies: a content-based approach or collaborative filtering [2]. The content-based approach recommends items similar to users selected items by calculating the similarity between items by using feature vectors generated by extraction of a user's selection record. Collaborative filtering recommends items selected by multiple users whose selection histories are similar to the relevant user by calculating similarity between users' records.

These two methods recommend items similar to the ones that the user selected before. These items belong to the first type stated above because they are recognized as interesting items by users. For example, a typical recommendation recommends TV programs featuring actor A to users who frequently watch TV programs featuring actor A. Consequently, a user might get bored with typical recommendation because it always recommends similar items that a user already knows are interesting [2]. For that reason, recommending items belonging to the second type– namely, serendipitous ones—become necessary. For example, serendipitous recommendation recommends educational programs featuring performer A to users who do not usually watch educational programs but frequently watch performer A.

Nevertheless, typical recommendation methods cannot recommend such serendipitous items preferentially.

The purpose of this study is to realize serendipitous recommendation. Accordingly, actual data that users recognized as "serendipitous" was collected, and a user-preference model was established first. Serendipitous recommendation methods based on that model were devised and evaluated with actual data. The results of this evaluation verified the effectiveness of a serendipitous recommendation method using "general unexpectedness" that is independent from a user's profile.

2. RELATED WORKS AND MOTIVATION

In the early stage of developing recommendation systems, the accuracy of recommendation of the first-type items was improved. It was thought that this improved accuracy was enough to enhance user satisfaction. However, it is recognized that novelty and serendipity are important factors in satisfying a user, aside from simply suitability to a user's preference [2, 3, 4, 5].

There are several related works on serendipitous recommendation. Ziegler et al. supposed that serendipitous items exist in recommendation lists of different items in different categories more than in the lists of similar items, and they proposed a recommendation method to increase diversity of recommendation lists [6, 7]. They defined "intra-list similarity" as the similarity between all items in a recommendation list by calculating similarity between two items. Moreover, they increased diversity by inserting low-similarity items.

Approaches that recommend serendipitous items directly have also been proposed. Hijikata et al. proposed a method for improving novelty and serendipity by calculating the probability of known items by using the information about knowns or unknowns given explicitly by user [8]. Another method calculates the probability of "degree of interest" by using an evaluation of items selected by a user (namely, "interested" or "not interested"). The items whose degree-of-interest probabilities are nearly equal are taken as serendipitous and recommended [9].

Another proposed method considers the items that are different from the ones users use habitually as serendipitous and recommends those [10]. This method uses a preference model to predict items that users like and a habit model to predict items that users use habitually. It then recommends a recommendation list including serendipitous items by predicting the unexpectedness of items by calculating differences between the results of the preference model and the habit model.

As mentioned above, the only serendipitous recommendation methods proposed until now are based on researchers' own assumptions; no methods based on actual data regarding a user's actual impression of selected items have been devised. Moreover, many works suppose that serendipitous items mean unexpected items, and they do not treat items that are unexpected and interesting.

In this study, the authors clarified what kinds of items are actually serendipitous by collecting data concerning a user's actual impressions, made assumptions based on that actual data, and devised two serendipitous recommendation methods based on those assumptions.

3. MODELING SERENDIPITOUS ITEMS ACCORDING TO ANALYSIS OF ACTUAL DATA

3.1 User-preference model

The assumption of user preference was established first, and what kinds of items are serendipitous for users was verified by analyzing a user's actual impressions collected by questionnaires based on this assumption. The user-preference model established before the questionnaires were given is explained in the following. Figure 1 shows the concept of the model. In this model, items are arranged in feature vector space generated by features of items. Although this feature vector space is highly dimensional, for simplicity, two-dimensional space is introduced in Figure 1. Items that a user selected before exist in the area near the feature vector that the user recognizes and knows are interesting (so-called "recognized items" below because the user recognizes them as interesting and not surprising if recommended). In a distant area from that area, serendipitous items (namely, surprising and interesting items) are supposed to exist. In an area far from the recognized area, not-interesting items are supposed to exist. Broadly speaking, it is supposed that each user has several recognized areas in the feature vector space, because there may be several reasons that the user selected certain interesting items; for example, the reasons for selecting a drama and a documentary program may be different.

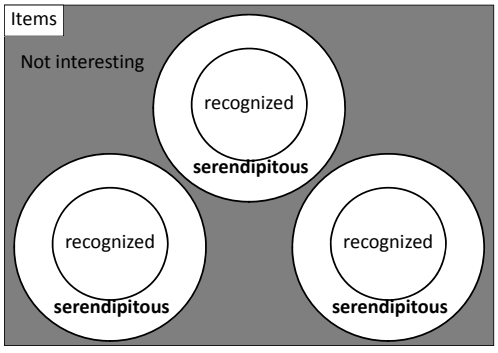


Fig. 1: Concept of user-preference model

3.2 Questionnaire

To collect users' actual impressions, a questionnaire was given to thirty users. The method is mentioned below. First, users read the information concerning a TV program selected randomly from TV programs over three months (31,433 programs), and then they classify these TV programs as recommended items into three categories: "recognized program" (first-type item), "serendipitous program" (second type) and "not-interesting program." An electric program guide (EPG) is used to provide the information concerning TV programs, which includes title, performer, and the other contents of programs.

In the questionnaire, three categories are available for choice by users. "Recognized program" means programs that users can expect from their own preference, for example, programs that users frequently watch. "Serendipitous program" means programs that users feel are interesting and surprising when recommended,

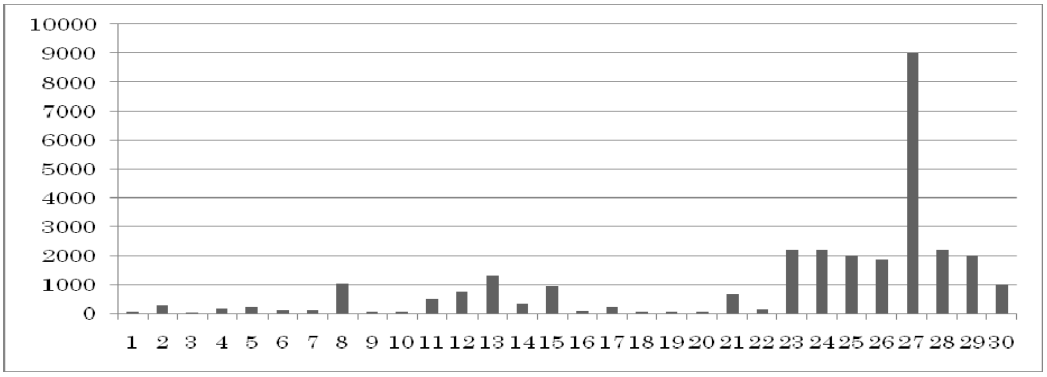


Fig. 2: Number of evaluated programs by each user

Vertical axis: Number of evaluated programs, Horizontal axis: User ID

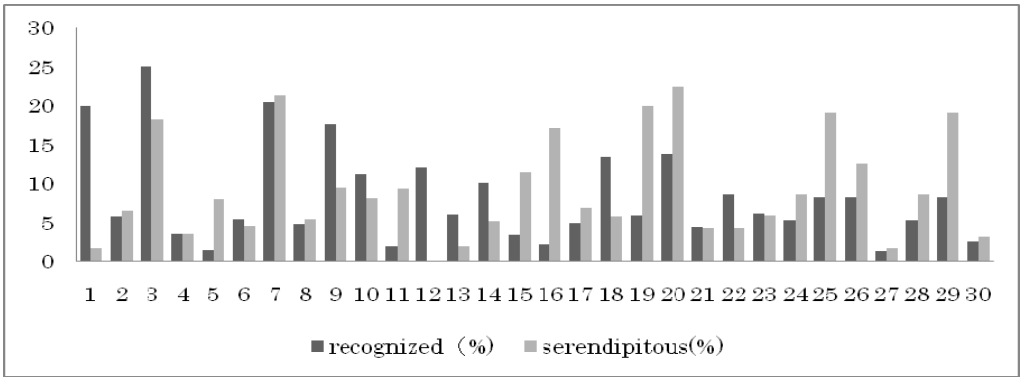


Fig. 3: Ratio of recognized programs and serendipitous programs in all programs for users

Vertical axis: Rate of each program in all programs, Horizontal axis: User ID

for example, programs that users do not expect from their own preferences but are interested in. "Not-interesting program" means programs that users are not interested in even though recommended.

It takes much time to answer this questionnaire (about one minute per program evaluation), so each user answered the questionnaire over one month, from ten to one hundred answers per day. We supposed that a user's preference does not change much over one month, because a series of TV programs lasts about three months.

All users live in Japan, twenty six work at Hitachi, Ltd., Central Research Laboratory and four are university students. Twenty five are male, and five are female. Fifteen are from twenty to thirty years old, eleven from thirty to forty, and the other four from forty to fifty. Each user evaluated about one thousand to five thousand programs.

3.3 Analysis method

The programs collected by questionnaire are first converted into term vectors extraction by morphological analysis of text information in the EPG. Each vector component contains two values, whether the EPG text includes the term or not. The recognized programs are then clustered to estimate the recognized area. For clustering, the distance between program Pi and program Pj is defined as

$$\mathrm{distance}(P_i, P_j) = \sum_{n=1}^{N} w_n \left| P_i(n) - P_j(n) \right| \quad (1)$$

where Pi(n) means the vector component of the *n*th term in program Pi, whether program Pi includes the *n*th term or not (1 or 0), w_n means the user's weight (a metric of user's preference) of the *n*th term. The user's own distance between programs is determined by introducing user's weight w_n.

Weight w_n of *n*th term *v* is calculated by TFIDF (product of term frequency and inverse document frequency) [11]. TFIDF is a metric of weighting characteristic terms occurring in observed documents by frequency in observed groups and in all groups. This metric is introduced to weight a user's preference as follows.

$$w_n = \mathrm{tfidf}(v \mid D) = \mathrm{tf}(v \mid D) \times \log\left(\frac{N_{all}}{N(v)}\right) \quad (2)$$

Here, D represents observed program, which means recognized programs here, tf(v|D) means the frequency that term *v* occurs in D, N_{all} means the total number of programs, and N(*v*) means the number of occurrences of term *v* in all programs.

3.4 Results

Figure 2 shows the number of programs evaluated by each user, and Figure 3 shows the ratio of recognized programs and serendipitous programs in all programs. Although each user has

various ratios, it is clear the rates of recognized programs are very low and there are a lot of inefficient programs. It is also clear that users who frequently watch TV programs evaluate more programs as recognized rather than serendipitous. On the other hand, the users who rarely watch TV programs evaluate more programs as serendipitous rather than recognized.

In regard to the questionnaire, most users said they feel serendipitous concerning the programs that they do not know before but are interesting (for example, interesting educational programs for users who do not watch educational programs) and the programs including an unexpected combination of interesting features (for example, educational programs featuring a comedian). However, surprising programs are not always unexpected programs, so the meanings of surprising would include other factors. Moreover, some users evaluated no programs as serendipitous, and some users cannot classify programs into the three types; consequently, it is difficult to evaluate their subjective impression quantitatively.

A clustering result of recognized programs is shown as dendrogram in Figure 4. The clustering method used is hierarchical clustering. The height of the cluster means average distance between programs belonging to the cluster and the cluster center calculated from Equation (1). The number of recognized areas is determined by cutting at a certain height of a cluster.

Figure 5 shows the ratio of average distance of recognized programs (radius of recognized area) and average distance of not-interesting programs (radius of not-interesting area) from the nearest center of the cluster with height of clusters. When the number of clusters increases, not-interesting programs are distributed outside of recognized area. On the other hand, when the number of clusters decreases, not-interesting programs are distributed inside the recognized area because the number of clusters is fewer than the true number of recognized areas.

Figure 6 shows the ratio of average distance of serendipitous programs (radius of serendipitous area) and the radius of a recognized area from the nearest center of the cluster with height of clusters. As the number of clusters increases, serendipitous programs are distributed outside of the recognized area.

Figure 7 plots the results from Figures 5 and 6. It is indicated that not-interesting programs are distributed outside the recognized area, and serendipitous programs are distributed far outside the recognized area.

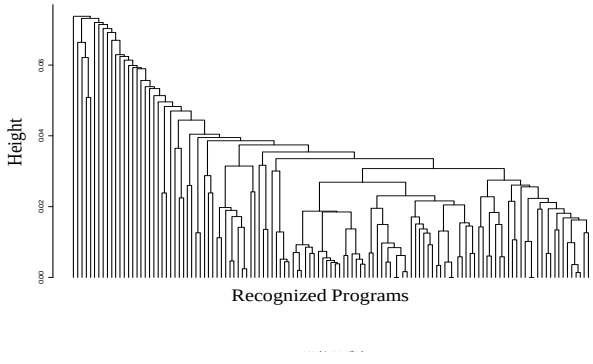


Fig. 4: Clustering result of recognized programs

(leaf nodes: recognized programs; vertical axis: height of cluster)

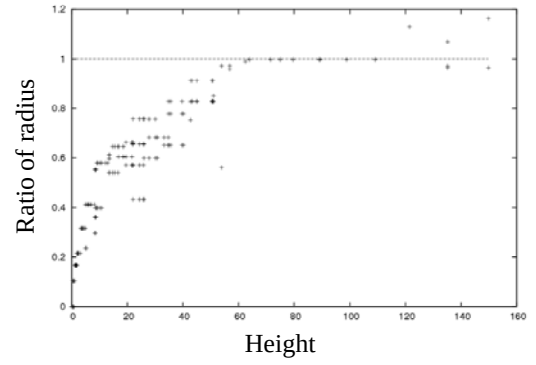


Fig. 5: Ratio of radiuses of recognized area and not-interesting area with height of cluster

(Denomination: radius of not-interesting area)

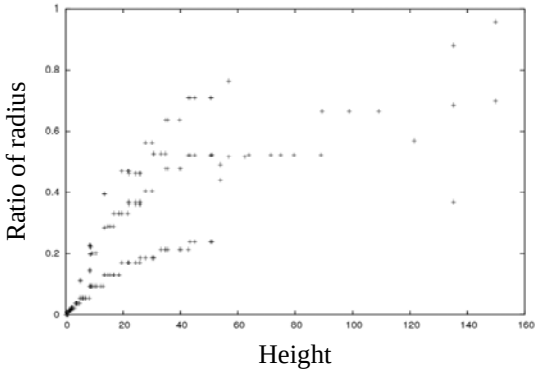


Fig. 6: Ratio of radiuses of recognized area and serendipitous area with height of cluster

(Denomination: radius of recognized area)

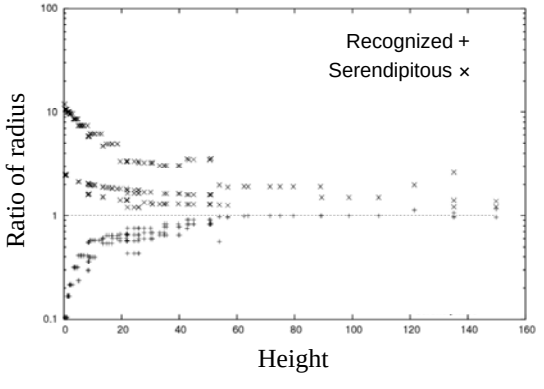


Fig. 7: Ratio of radiuses of not-interesting area, recognized area, and serendipitous area with height of cluster

(Denomination: not-interesting area)

3.5 Model based on analysis results

To summarize the results presented in this section, in the feature vector space generated by EPG texts, not-interesting programs are distributed outside the recognized area and serendipitous programs are distributed far outside the recognized area. This result does not support the assumption in Figure 1. We therefore suggest the structure of user preference as shown in Figure 8 instead of that shown in Figure 1.

Distance from the center of the recognized area means the number of terms in the program vector but not in the center because the program-vector components are described by only two values, whether each term in the contents of programs is included or not. In addition, the weight of terms is calculated as a user's preference by TFIDF. Therefore, even though the item includes many low-weight terms and is rarely watched, the distance from the recognized area is not far. And if the program includes high-weight terms belonging to the other recognized area, the distance from recognized area becomes far. Consequently, programs including many high-weight terms belonging to the other recognized area and not similar to the ones in the nearest recognized area are distributed in the intermediate region of recognized areas, and users treat them as serendipitous programs. This assumption expresses that "the contents makes users feel serendipity concerning an unexpected combination of program contents," which some users commented in the questionnaire.

Figure 9 shows the distribution of each type of program plotted against distance from one center of a recognized area. The solid line represents the distribution of not-interesting programs, the dotted line represents the distribution of serendipitous programs, and the dashed line represents distribution of recognized programs. The nearest peak of recognized programs to the origin represents the peak of the distribution of the recognized area, and the next-nearest peak represents several recognized areas. As shown in Fig. 7, not-interesting programs are distributed broadly both in the recognized area and the serendipitous area; consequently, it is difficult to distinguish only serendipitous programs accurately by distance between programs given by Equation (1).

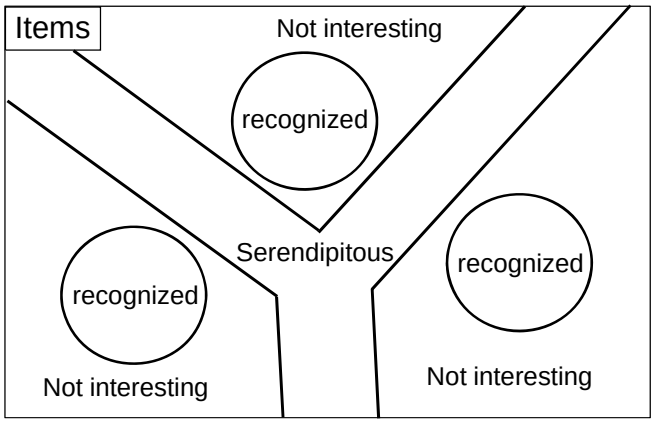


Fig. 8: User-preference model based on analysis results

4. PROPOSAL AND EVALUATION OF RECOMMENDATION METHODS

4.1 Proposed methods

4.1.1 Using distance between items

The distance between items used in this method is calculated from Equation (1) reflecting a user's preference. First, the proposed

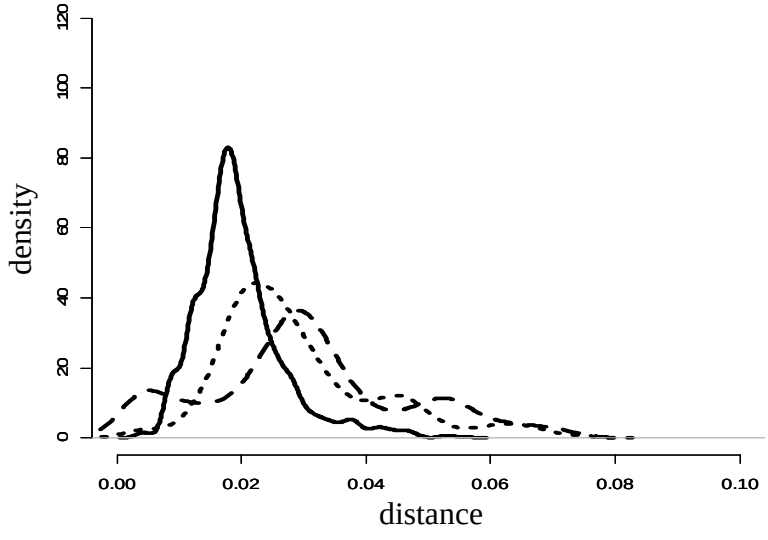


Fig. 9: Density of programs in each area with distance from center of recognized area

(Vertical axis: density of programs, horizontal axis: distance from center of recognized area,

solid line: not interesting programs, dotted line: serendipitous programs, dashed line: recognized programs)

recommender system learns features of programs according to the user's viewing history. In the same way as described in section 3, program vectors are defined by a term vector, whose component has two values. Second, the system splits watched programs (i.e., recognized programs) into several clusters by hierarchical clustering and finds the centers of recognized areas. The number of recognized areas is defined as 7 to 10 according to the results from the questionnaire. The system then calculates the distance of each not-watched program from the nearest center, and recommends the 10 longest programs. In short, the system recommends 10 highest score programs calculated according to

$$\mathrm{Score}(P_i) = \mathrm{distance}(P_i, C_{nearest}) \qquad (3)$$

Here, $C_{nearest}$ means the center of the nearest recognized area with program Pi.

This method may not recommend serendipitous programs accurately because not-interesting programs are distributed broadly. This method is referred to as the "first method" hereafter.

4.1.2 Using general unexpectedness

This method (hereafter, "second method") introduces "unexpectedness of programs" in addition to the distance used in the first method in order to capture a "surprising" factor. The results of the questionnaire indicate that the serendipitous programs have an unexpected aspect for users, as shown in Fig. 8. It is assumed that "unexpectedness" means something is hard to predict the program contents. Regarding a program-recommendation system, it is assumed that it is related to an unlikely combination of features. The second method treats highly unexpected and interesting programs as serendipitous programs. A general metric of difficulty of expecting programs for every user is defined by the sum of the tendencies of co-occurrence of the terms in the program.

$$\begin{aligned}&\mathrm{Expectedness}(P_i)\\ &= \frac{1}{|P_i|}\sum_{v,w\in P_i}\mathrm{Tendency\ of\ co\text{-}occurrence}(v,w) \qquad (4)\\ &= \frac{1}{|P_i|}\sum_{v,w\in P_i}\frac{Nvw}{Nv+Nw-Nvw}\end{aligned}$$

Tendency of co-occurrence (v, w) means tendency of co-occurrence of terms *v* and *w* in all programs. It makes it possible to evaluate quantitatively how unexpected a program is for users. N*v* means number of programs including term *v*, and N*vw* means number of programs including both term *v* and *w*. |Pi| means number of terms in program Pi and is a normalized factor. If the co-occurrence of the terms is low, expectedness will be low, and the program will be highly unexpected, so users would be unable to find it. Unexpectedness is defined as the inverse of expectedness (see Equation (6)), and 10 high-score programs (calculated according to the sum of squares of distance between programs and unexpectedness as below) are recommended.

$$\begin{aligned}\mathrm{Score}(P_i) = {} & \alpha \times \mathrm{distance}(P_i, C_{nearest})^2\\ & + (1-\alpha)\times \mathrm{Unexpectedness}(P_i)^2\end{aligned} \qquad (5)$$

$$\mathrm{Unexpectedness}(P_i) = \frac{1}{\mathrm{Expectedness}(P_i)} \qquad (6)$$

Parameter α controls the degree of combination of a user's preference and unexpectedness of programs. Simply put, equation (5) is a very simple linear combination of squares of distance and unexpectedness.

4.2 Evaluation method

4.2.1 Dataset

The results of the questionnaire implemented in the third section were used. Data of fourteen users who classified more than 100 programs into recognized or serendipitous programs were selected, because it was supposed that serendipitous recommendation becomes necessary after watching TV programs for about one month. (It was assumed that users get bored with typical recommendation after about one month and most users watch fifty TV programs per month). Each user evaluated from 1000 to 5000 programs, and the ratio of serendipitous programs in all evaluated programs is 7 to 8%.

4.2.2 Procedure

The three proposed methods are applied to each user. The procedure is mentioned below. First, the system learns recognized areas from fifty recognized programs. In this evaluation experiment, 50 recognized programs were prepared randomly as a training set from evaluated programs as recognized. Next, the system recommends ten high-score programs by using the proposed methods, random recommendation, and a method using only unexpectedness for each user from the remaining evaluated programs by using the recognized areas learned first. Random recommendation means recommending ten programs randomly, and the method using only unexpectedness calculates a program score according to unexpectedness only (α=0 in Equation (5)). This experiment was performed ten times, and each time different recognized programs were used and the accuracy of each method was compared.

4.2.3 Evaluation metrics

Our purpose is to recommend serendipitous programs. So we use detection rate and precision as evaluation metrics for the purpose of evaluating accuracy of the proposed methods to detect serendipitous programs. Detection rate means the probability of detecting a serendipitous program and precision means rate of serendipitous programs in recommendation list.

4.3 Results

Table 1 lists the evaluation results of the two proposed methods, random recommendation and only unexpectedness. Accuracy metrics are calculated as an average of users. Parameter α is set to 0.05, so the second method has the highest accuracy, .

The results in Table 1 show that detection rate and precision of random recommendation are low, so it suggests how difficult it is to recommend serendipitous programs. On the other hand, the accuracy of the second method (i.e., using unexpectedness of programs) is higher than the other methods, detection rate is 78.2% and precision is 21.6%. This result means that the second method recommends serendipitous programs accurately.

Table 1: Accuracy results

Method	Random	First	Second	Only unexpectedness
Detection Rate [%]	51.9	49.8	78.2	32.8
Precision [%]	7.98	7.51	21.6	5.21

While accuracy of the first method (i.e., using distance only) is the same as that of the random method, accuracy of the second method is much higher than the random one, and accuracy of the unexpectedness-only method is lower than that of the random one. This result shows it is possible to recommend serendipitous programs by using both distance reflecting a user's preference and unexpectedness of programs.

The first method recommends programs including not-interesting ones, because it recommends items that are not similar to recognized programs. On the other hand, the second method distinguishes "unexpected and interesting programs" and "unexpected but not-interesting programs" from programs with low similarity according to unexpectedness. Consequently, the accuracy of the second method is high.

Figure 10 shows the concept of user preference by distance and unexpectedness inferred from these results. Serendipitous programs and not-interesting programs are distant from the recognized area. According to the result "only unexpectedness" in Table 1, serendipitous programs exist in extra high-unexpectedness areas because they tend to have more combinations of terms whose tendency of co-occurrence is low. Moreover, in the right lower box, not-interesting programs may exist. It seems very possible that the user would already know the highly unexpected programs near to recognized programs and not select them, because "unexpectedness" is a general metric and does not depend on a user's record.

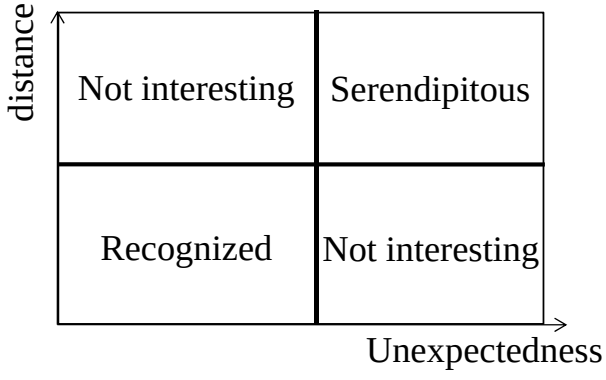


Fig. 10: Concept of user preference with distance and unexpectedness

Unexpectedness of programs calculated from tendency of co-occurrence of terms in the programs is introduced here. For example, users find programs by reading TV guides and EPGs on web sites. Therefore, programs that have rare contents in TV guides are supposed to be serendipitous. TV guides and EPG are not provided by users but by the surroundings of users, so we simply introduce unexpectedness independently from a user's characteristics. Examinees in this experiment are deemed to live in similar environments. The influence of unexpectedness for users living in totally different environments (e.g., living in different countries) might be significant. Unexpectedness may therefore be a frequency of contact with items similar to the relevant item which a user contacts with so far, with or without intention.

Finally, our proposed method is compared with the other related methods. It is hard to compare by accuracy because serendipity depends on user's subjective impression, so we compare these by requirements in Table. 2.

Table 2: Comparison of serendipitous recommendation

Requirement	Proposal	Different from Habit	Different from Interesting & Not	Collaborative
User's Impression	**Unnecessary**	**Unnecessary**	**Necessary**	**Unnecessary**
Other user's record	**Unnecessary**	**Unnecessary**	**Unnecessary**	**Necessary**
User's Habit	**Unnecessary**	**Necessary**	**Unnecessary**	**Unnecessary**
Information of Programs	**Necessary**	**Necessary**	**Necessary**	**Unnecessary**

Related works require some information concerning users, one requires a user's impression of recommended items, another requires other users' records, and another requires user's habits. The proposed recommendation method requires few evaluation values to learn a user's preference and does not depend on user's surroundings. On the other hand, it requires information concerning programs, but recently there is much information regarding programs on Internet reference sites like *Wikipedia*. In short, the proposal method has most broad utility regarding various systems because it is useful for both devices and servers. As for our future work, however, which method satisfies users must be verified by a user's subjective evaluation. On the other hand, we suggest using suitable terms for each user.

5. FUTURE WORK

Although the accuracy of our proposal serendipitous recommendation method was verified, the following three tasks remain as future work: improve accuracy, evaluate by more users, and tune performance of actual system

To improve accuracy, it is necessary to select the recognized area outside of which many serendipitous programs exist; in fact, there are some recognized areas outside of which serendipitous programs do not exist. By considering the radius and number of programs included in recognized areas, it is possible to select the best recognized area. Moreover, another approach to improving accuracy is to get rich information concerning programs via metadata and information on web sites.

It is also necessary to satisfy users by capturing user context with their spatial temporal information; for example, a user does not

want to watch a program in the morning but in the evening instead. It is also important to capture time-dependent user preferences, for example, users feel serendipity if a recommended program was not watched recently but has been watched in the past. With our recommendation method, a user's preference is described in a feature vector space generated by the user's selection history, so the structure of the space and distribution of user preference depends on time.

To make the user-preference model statistically strong, it is necessary to evaluate our proposed method by more users, because the concept of serendipity is supposed to depend strongly on user's subjective impression. Moreover, it is important to establish methods for evaluating a user's satisfaction quantitatively.

To introduce our recommendation method in an actual system, it is necessary to design an optimal data structure and speed up the method.

In this study, we verified the recommendation method by using TV programs, but this approach can be applied to recommend items like books and DVDs from a user's record of selecting TV programs. We plan to use this approach to capture the meanings of users like and dislike by collecting and analyzing user's records.

6. CONCLUSION

To realize serendipitous recommendation, a recommendation method for extracting a user's preference was proposed and evaluated. In particular, based on actual data obtained by giving a questionnaire to thirty users, a user-preference model using distance between programs was established. Based on this model, a serendipitous recommendation method using the distance and unexpectedness of programs was proposed. This method recommends a serendipitous program accurately at a detection rate is 78.2%. Moreover, it was found that the impression of unexpectedness depends on a user's living environment rather than his or her character. This result is an important fact in regard to understanding a user's preference in principle.

7. REFERENCES

[1] E. Toms: Serendipitous Information Retrieval, Proc. of DELOS Workshop, 2000

[2] Herlocker, J., et al.: Evaluating Collaborative Filtering Recommender Systems, ACM Transactions on Information Systems, Vol. 22, No. 1, pp. 5-53 (2004)

[3] K. Swearingen and R. Sinha: Beyond Algorithms: An HCI Perspective on Recommender Systems, ACM SIGIR Workshop on Recommender Systems (2001)

[4] S. M. McNee, J. Riedl, and J. A. Konstan: Making Recommendations Better: An Analysis Model for Human-Recommender Interaction, In proc. of ACM Special Interest Group on Computer-Human Interaction (ACM SIGCHI), pp. 997-1101 (2006)

[5] S. M. McNee, J. Riedl, and J. A. Konstan: Being accurate is not enough: How accuracy metrics have hurt recommender systems, In proc. of ACM Special Interest Group on Computer-Human Interaction (ACM SIGCHI), pp. 997-1101 (2006)

[6] C. N. Ziegler, G. Lausen, and L. S. Thieme: Taxonomy-driven Computation of Product Recommendations, In proc. of the 2004 ACM CIKM Conference on Information and Knowledge Management, pp. 406-415 (2004)

[7] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and George Lausen: Improving Recommendation Lists Through Topic Diversification, In proc. of World Wide Web Conference, pp. 22-32 (2005)

[8] Y. Hijikata, T. Shimizu, and S. Nishida: Discovery-oriented Collaborative Filtering for Improving User Satisfaction, In proc. of the 14th ACM International Conference on Intelligent User Interfaces(ACM IUI 2009), pp. 67-76 (2009)

[9] Leo Iaquinta, Macro de Gemmis, Pasquale Lops, Giovanni Semeraro, Michele Filannino, and Piero Molino: Introducing Serendipity in a Content-based Recommender System, Hybrid Intelligent Systems, 2008. HIS '08. Eighth International Conference on 10-12 Sept. 2008, pages 168 – 173

[10] T. Murakami, et al.: A Method to Enhance Serendipity in Recommendation and its Evaluation, Transactions of the Japanese Society for Artificial Intelligence, Vol. 24, Issue 5, pp. 428-436 (2009).

[11] Li-Ping Jing, et al.: Improved Feature Selection Approach TFIDF In Text Mining, Proceedings of the First International Conference on Machine Learning and Cybernetics (2002)

Enhancing Accuracy of Hybrid Recommender Systems through Adapting the Domain Trends

Fatih Aksel
Department of Computer Engineering
METU
fatih.aksel@ceng.metu.edu.tr

Ayşenur Birtürk
Department of Computer Engineering
METU
birturk@ceng.metu.edu.tr

ABSTRACT

Hybrid recommender systems combine several algorithms based on their hybridization strategy. Prediction algorithm selection strategy directly influence the accuracy of the hybrid recommenders. Recent research has mostly focused on static hybridization schemes which are designed as fixed combinations of prediction algorithms and do not change at run-time. However, people's tastes and desires are temporary and gradually evolve. Moreover, each domain has unique characteristics, trends and unique user interests. In this paper, we propose an adaptive method for hybrid recommender systems, in which the combination of algorithms are learned and dynamically updated from the results of previous predictions. We describe our hybrid recommender system, called AdaRec, that uses domain attributes to understand the domain drifts and trends, and user feedback in order to change it's prediction strategy at run-time, and adapt the combination of content-based and collaborative algorithms to have better results. Experiment results with datasets show that our system outperforms naive hybrid recommender.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - *Information filtering, Retrieval models, Selection process*; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Design, Experimentation, Algorithms

Keywords

Hybrid Recommender Systems, Switching Hybridization, Decision Tree Induction, Hybrid Recommender Framework, Adaptive Recommender Systems

1. INTRODUCTION

Content-based and collaborative filtering are the two major recommendation techniques that have come to dominate the current recommender system area. Content-based recommender system uses the descriptions about the content of the items (such as meta-data of the item), whereas collaborative filtering system tries to identify users whose tastes are similar and recommends items they like. Other recommendation technologies include hybrid techniques, knowledge-based approaches etc [6]. The popular Amazon item-to-item system [7] is one of the well-known recommender system that uses collaborative filtering techniques. NewsWeeder [12] and InfoFinder [11] are the pure content-based recommender systems that analyze the content of items, in their recommendation process.

Previous research in this area, has shown that these techniques suffer from various potential problems-such as, sparsity, reduced coverage, scalability, and cold-start problems [1, 6, 23]. For example; collaborative filtering techniques depend on historical ratings of across users that have the drawback, called cold start problem - an item cannot be recommended until it has been rated by a number of existing users. The technique tends to offer poor results when there are not enough user ratings. Content-based techniques can overcome the cold start problem, because new items can be recommended based on item features about the content of the item with existing items. Unfortunately, content-based approaches require additional information about the content of item, which may be hard to extract (such as, movies, music, restaurants). Every recommendation approach has its own strengths and weaknesses. Hybrid recommender systems have been proposed to gain better results with fewer drawbacks.

Most of the recommender system implementations focuses on hybrid systems that use mixture of recommendation approaches [6]. This helps to avoid certain limitations of content-based and collaborative filtering systems. Previous research on hybrid recommender system has mostly focused on static hybridization approaches (strategy) that do not change their hybridization behavior at run-time. Fixed strategy may be suboptimal for dynamic domains&user behaviors. Moreover they are unable to adapt to domain drifts. Since people's tastes and desires are transient and subject to change, a good recommender engine should deal with changing consumer preferences.

In this paper, we describe an *Adaptive Hybrid Recommender*

System, called AdaRec, that modifies its switching strategy according to the performance of prediction techniques. Our hybrid recommender approach uses adaptive *prediction strategies* that determine which *prediction techniques* (algorithms) should be used at the moment an actual prediction is required. Initially we used manually created rule-based strategies which are static. These static hybridization schemes have drawbacks. They require expert knowledge and they are unable to adapt to emerging trends in the domain. We now focus on prediction strategies that learn by themselves.

The paper is organized as follows. Related work is described in Section 2. In Section 3, we present the adaptive prediction strategy model for hybrid recommenders. We then describe our experimental recommender systems' architecture & learning module that dynamically adjusts recommendation strategy in response to the changes in domain. An initial evaluation of our approach, based on MovieLens dataset, is presented in Section 6.

2. RELATED WORK

Personalization techniques have been investigated extensively in several areas of computer science. Especially in the domains of recommender systems, personalization algorithms have been developed and deployed on various types of systems [1].

There have been several research efforts to combine different recommendation technologies. The BellKor system [4], statically combines weighted linear combination of more than a hundred collaborative filtering engines. The system uses the model based approach that first learns a statistical model in an offline fashion, and then uses it to make predictions and generate recommendations. The weights are learned by using a linear regression on outputs of the engine. The STREAM [3] recommender system, which can be thought of as a special case of the BellKor system, classifies the recommender engines in two levels: called level-1 and level-2 predictors. The hybrid STREAM system uses run-time metrics to learn next level predictors by linear regression. However combining many engines level by level results performance problems at run-time. Our approach combines different algorithms on a single hybrid engine with an adaptive strategy.

Some hybrid recommenders choice the best suited recommender engine for a specific case (user, item, input etc.). For example, the Daily Learner system [5], which is a personal web-based agent, selects the best recommender engine according to the confidence levels. But in order to handle different engines in a common point, confidence scores should be comparable.

The use of machine learning algorithms for user modeling purposes has recently attracted much attention. In [13], the authors proposed a hybrid recommender framework to recommend movies to users. The system uses a content-based predictor to enhance existing user data, and then provides personalized suggestions through collaborative filtering. In the content-based filtering part of the system, they get extra information about movies from the IMDB[1] and handle each movie as a text document. User and item profiles are built by using a Naive Bayes classifier that can handle vectors of bags of words; where each 'bag-of-words' corresponds to a movie-feature (e.g. title, cast, etc.). The Naive Bayes classifier is used to approximate the missing entries in the user-item rating matrix, and a user-based collaborative filtering is applied over this dense matrix.

In our system, we choice the Duine Framework[2] for our recommendation engine component, which is an open-source hybrid recommendation system [20]. The Duine framework allows users to develop their own prediction engines for recommender systems. The framework contains a set of recommendation techniques, ways to combine these techniques into recommendation strategies, a profile manager, and it allows users to add their own recommender algorithm to the system. It uses switching hybridization method in the selection of prediction techniques. The result of a Duine prediction engine is the retrieved set of information with added data about how interesting each piece of information is for the user [20, 17].

3. ADAREC: AN ADAPTIVE HYBRID RECOMMENDER SYSTEM

Hybrid recommendation systems combine multiple algorithms and define a *switching behavior* (strategy) among them. This strategy decides which technique to choose under what circumstances for a given prediction request. Recommender system's behavior is directly influenced by the prediction strategy. The construction of accurate strategy that suits in all circumstances is a difficult process. A well-designed adaptive prediction strategy offers advantages over the traditional static one.

In our approach we use the switching hybridization in order to decide which prediction technique is most suitable to provide a prediction. Prediction techniques, also called the predictors are combined into an upper prediction model that is called prediction strategy. The central concept in combining multiple predictors using the switching hybridization method is the prediction strategy. Prediction Strategy, which defines the algorithm selection strategy, changes the behavior of the recommender engines at run-time.

Most of the currently available personalized information systems focus on the use of a single selection technique or a fixed combination of techniques [20, 14]. However, application domains are dynamic environments. Users are continuously interacting with domain, new concepts and trends emerge each day. Therefore, user interests might change dynamically over time. It does not seem possible to adapt trends by using a static approach (static prediction strategy). Instead of static methods dynamic methods that can adapt to change on domains, could be more effective.

Different design approaches might be used for the prediction strategy adaptation. Rule based, case based, artificial neural networks or Bayesian are some of the learning techniques. Each technique has its own strengths and weaknesses. In this paper, we introduce self adaptive prediction strategy learning module which employs a strategy based on

[1] `http://www.imdb.com/`

[2] `http://duineframework.org/`

its attached learning technique. Learning module initializes prediction engine according to the specified machine learning technique. This prediction strategy adapts itself to the current context by using the previous performance results of the techniques. Different machine learning algorithms that induce decision trees or decision rules could be attached to our experimental design.

4. PREDICTION STRATEGY LEARNING

Duine Recommender offers extensive options for configuring various recommender algorithms. It provides a sample of most common recommendation algorithms that can be combined in algorithm strategies. In the Duine Recommender the knowledge that designs the selection strategy is provided manually by experts [8, 21]. However, the combination of different techniques in a dynamic, intelligent and adaptive way can provide better prediction results. The combination of techniques should not be fixed within a system and that the combination ought to be based on knowledge about strengths and weaknesses of each technique and that the choice of techniques should be made at the moment a prediction is required.

Hybridization of a recommender system employs using the best prediction technique from the available ones. The main purpose of a prediction strategy is to use the most appropriate prediction technique in a particular context. Adaptive prediction strategy depicts the selection rules of prediction techniques. Figure 1 shows a sample prediction strategy that decides when to use which predictors (gray nodes) by using the threshold values (arrows).

Prediction strategy employs the selection rules of the available prediction techniques. Our experimental hybrid recommender system has plenty of pre-defined prediction techniques. These prediction techniques are implemented by using different paradigms. Content-based and collaborative filtering are the two principal paradigms for computing recommendations [23]. In our system we have used content-based, collaborative filtering, knowledge based and case-based prediction techniques.

To make decisions about which predictor is suitable for the current context, threshold values, predictors' state and users feedback are used by the adaptive prediction strategy. The state of a predictor is described by the amount and at the quality of knowledge that is available to the predictor. In other words, the knowledge that is used by the prediction technique is the basis of its predictions. One of the objectives of the prediction strategy is to select the right prediction technique according to the current states of the predictors. The initial strategy is defined by using the expert knowledge. System starts with an initial prediction strategy. Later on the Learning Module adjusts the prediction strategy to the current systems' domain.

Decision trees and decision rules are model based strategies. Each node in a decision tree represents some attributes and each branch from a node corresponds to a possible value for that attribute. When trying to classify a certain instance, as seen in the Figure 1, one starts at the root of the decision tree and move down the branches of the tree according to the values of the attributes until a leaf node is reached. The

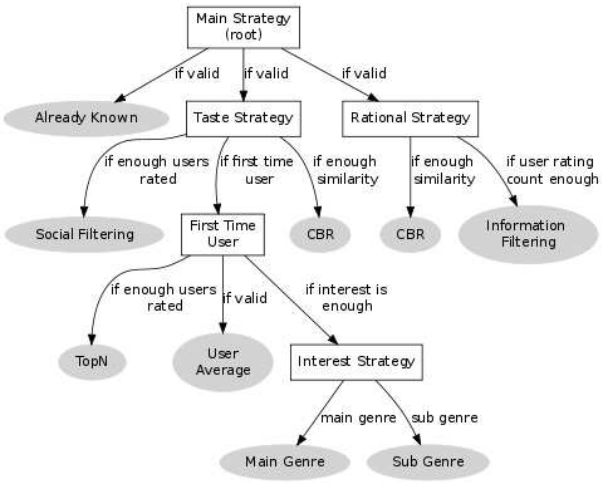


Figure 1: The Duine Recommender's prediction strategy depicted as decision tree. Grey nodes represent prediction algorithms and arrows represent the attributes&threshold values.

leaf nodes represent the decisions of the tree [20].

Adaptive strategy can be designed using by rule sets or trees that contain the knowledge on decisions. Decision rules can also be expressed as decision trees. Experts often describe their knowledge about a process using decision trees and decision rules as they are easy to interpret by other people. The decision trees are a good interface between experts and the recommender systems.

Adaptive prediction strategy is in the form of a decision tree. Another way to represent decision tree is using decision rules. Decision rules generally take the form of IF ... THEN ... rules, i.e. `IF attribute`$_1$ `= value`$_1$ `AND attribute`$_2$ `= value`$_2$ `THEN result`$_2$.

Depending on the nature of the domains (movie, music, book etc.) different attribute-value combinations can be used for prediction strategy design. In our proposed system, since we tested on MovieLens dataset, we choice these specific attributes that have meaningful correlations between movie domain and prediction techniques. We believe that, by measuring the changes on these attributes, we can capture the domain drifts and trends

1. *item ratings count*, the number of ratings that the current item has.

2. *item similar user count*, similar users count that have already rated the current item.

3. *similar item count*, the number of similar items count according to similarity measures.

4. *main genre interest*, main genre interest certainty of the current item among the users items.

5. *sub genre interest*, sub genre interest certainty of the current item among the users items.

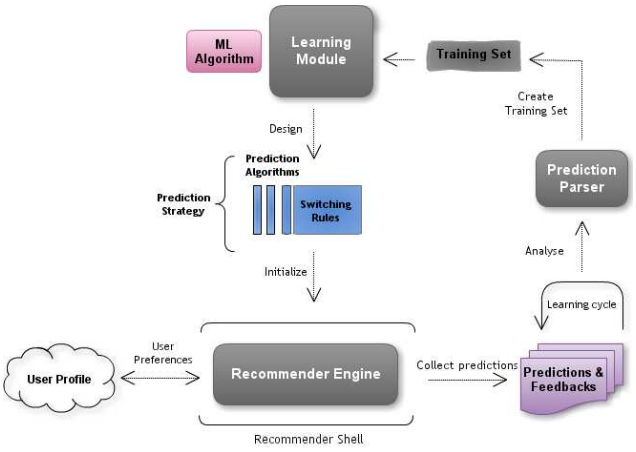


Figure 2: Overall architecture of the AdaRec System.

Decision trees/decision rules are constructed using the combination of the above five attributes. As shown in Figure 1 at each node of the decision tree a attribute is compared with a value, which is called *threshold value*. These five attributes are used to classify the prediction algorithms.

The Recommender System needs to work with the best suited prediction technique for its domain and users. In our system, we used and tested different prediction techniques. These are; *topNDeviation, userAverage, socialFiltering, CBR (Case Based Reasoning), mainGenreLMS, subGenreLMS, informationFiltering*. Because of the dynamic nature of the domain, these attributes create different forms of decision trees.

Each domain has unique characteristics including user behaviors, emerging trends etc. Recommender engine able to adapt itself to the changes in the domain by analyzing the changes. Also the system can capture the trends in the domain able to re-design its' attached prediction strategy. In our system, we used and tested different prediction techniques.

The quality of a decision tree depends on both the classification accuracy and the size of the tree[10]. After reviewing and testing many of the options, we decided to use two decision tree classifiers; *J48 (pruned C4.5 decision tree)*[18], which is the WEKA's[3] implementation of the decision tree learner based on the C4.5 decision tree algorithm, and *BF-Tree (best first-decision tree)*, which is a decision tree learner that uses a best first method of determining its branches. Also in order to compare the rule-based and tree induction methods we plugged and tested the *Conjunctive Rules* classifier, which is a simple rule learner that learns a set of simple conjunctive rules.

5. OVERVIEW OF THE ADAREC SYSTEM

Figure 2 depicts the architectural overview of the proposed AdaRec system. Our experimental framework is an extension of the open-source Duine Framework. System consists of two core parts, *Recommender Engine* and *Learning Module*.

Recommender Engine is responsible for generating the predictions of items based on the previous user profiles and item contents. It attempts to recommend information items, such as movies, music, books, that are likely to be of interest to the user. The recommender generate the predictions by using its attached prediction strategy. The implementation here uses the open source Duine Framework for the recommender engine.

Learning Module handles the new prediction strategy creation upon the previous instances and performance results of the prediction techniques on each learning cycle. It allows the building of new decision trees/decision rules based on the previous recorded instances.

Learning cycle is a logical concept that represents the redesign frequency of the prediction strategy. Each instance, based on the indicated count, and prediction algorithm's performance results are collected between two learning cycles. The learning module modifies the values of attributes in decision rules, which is also called threshold values according to the gathered results of the prediction techniques performance. The old prediction strategy is modified by using recommender engines' machine learning algorithm (rule tuning, rule adaptation, decision tree induction etc.). The modification of the threshold values allows recommender system to analyze&adapt the nature of the users and the domain.

The learning module first tests the accuracy of the each predictor in the system. Than the prediction strategy is redesigned by the learning module in order to improve proper use of predictors. Adaptive prediction strategy improves its' prediction accuracy by learning better when to use which predictors. The learning module adapts the hybrid recommender system to the current characteristics of domain.

Previous predictions and user feedbacks are fed to the training set of the next learning cycle. Inductive learning is used in learning from the training set. In our experiments we tested different (1K, 1.5K, 2K and 3K) instance sizes for training sets. The training set contains the instances from the previous learning cycle results. There are quite a few inductive learning techniques to choose from, including information theoretic ones (e.g. Rocchio classifier), neural networks (e.g. back-propagation), instance-based methods (e.g. nearest neighbour), rule learners (e.g. RIPPER), decision trees (e.g. C4.5) and probabilistic classifiers (e.g. naive Bayes) [14].

The User Profile, is the representation of the user in the system. For each active user a user model is stored in the user profile. User profile holds the knowledge (such as preferences, feedbacks, feature weights etc.) about users in a structured way. The Recommender Shell, encapsulates the Recommender Engine's interaction with other modules. The shell serves the created prediction strategies to the engine. The Prediction Parser, produces the performance results of the prediction algorithms based on the analyzing of the collected predictions & feedbacks. This module handles the decomposition of the prediction results and generates the

[3]Waikato Environment for Knowledge Analysis: `http://www.cs.waikato.ac.nz/ml/weka/`

training set of sample instances with current attributes.

User feedbacks and MAE (Mean Absolute Error) are the main criteria, which describe the trends in the domain. Adaptive prediction strategy learns its domain trends over time via unobtrusive monitoring and relevance feedback. In our proposed system, we focused self adaptive prediction strategy that classifies according to its' attached machine learning technique. This prediction strategy adapts itself to the current context by using the previous performance results of the techniques. Different machine learning algorithms that induce decision trees or decision rule sets could be attached to our experimental design.The architecture is open and flexible enough to attach different machine learning algorithms.

6. EXPERIMENTS

In this section we present a brief discussion of our experimental dataset, evaluation metric followed by the experimental results and discussion.

In order to assess the impact of our proposed adaptive recommender and different machine learning algorithms, we calculated prediction accuracy (MAE) of the system using different configurations of the machine learning schemes. Different MovieLens datasets are examined during the experiments.

The datasets are divided into temporal subsets according to their *time-stamp* values. Natural domain trends and changes in user interests are handled by using the subsets of the dataset.

6.1 Datasets

We used data for our recommender system from MovieLens[4], which is a web-based research recommender system that debuted in Fall 1997 [15].

In our experiments MovieLens one million ratings dataset is used, with 6040 users and 3900 movies pertaining to 19 genres. MovieLens dataset contains explicit ratings about movies and has a very high density. In order to train the recommender system, the MovieLens dataset is divided in to different temporal sets based on their distribution in time. When testing the ratings of first sets are used for recommender engine training [15, 9, 16].

6.2 Experimental Setup

Recommender systems researchers use a number of different measures for evaluating the success of the recommendation or prediction algorithms [19, 22]. For our experiments, we use a widely popular statistical accuracy metric named Global Mean Absolute Error (MAE), which is a measure of the deviation of recommendations from their true user-specified values. The MAE is defined as the average difference between the predicted ratings and the real user ratings, as defined within the test sets. Formally, MAE can be defined as:

$$MAE = \frac{\sum_{i=1}^{N} |p_i, r_i|}{N}$$

where p_i is the predicted value for item i and r_i is the user's rating.

[4] `http://www.movielens.umn.edu`

The aim of the experiments is to examine how the recommendation quality is affected by our proposed learning module. The present model of the Duine Framework is non adaptive but it supports predictor level learning. This original state of the framework is referred to *baseline*. As shown in the Figure 1, Duine recommender uses a static prediction strategy as its' hybridization scheme, which does not change at run-time. We want to compare the prediction quality obtained from the framework's baseline (non adaptive) to the quality obtained by our proposed experimental framework (adaptive). The approach will be considered useful if the prediction accuracy is better than the baseline. At the first iterations both systems are initialized with the same strategy, which is the default strategy of the Duine Framework.

The validation process is handled using the following procedure:

1. The ratings provided by the dataset are fed to the system one by one, in the logical order of the system (ordered by timestamps).
2. When a rating is provided during validation, prediction strategy is invoked to provide a prediction for the current user and the current item. The average prediction error can be used a performance indicator of the attached prediction strategy.
3. After the error has been calculated, the given rating is provided as feedback to the recommender system. The adaptive system collects the feedback as well as the current attributes of the system as instances.
4. Whenever the collected instances reached the learning cycle's instance count (1000 instances for example), the prediction strategy of the system will be redesigned by the adaptive system according to the instances.

This way, when the next learning cycle is processed, the adaptive system has learned from all the previously processed ratings. This process is repeated for all ratings at both adaptive and non-adaptive (baseline) systems in the test set. At the end MAE is calculated by averaging absolute errors within the baseline and the adaptive system, as described above.

6.3 Results & Discussion

Each prediction provided by the two different systems are examined. The prediction accuracy and the prediction error (MAE) are recorded. In experimenting with the MovieLens dataset, we considered both the proposed method, called *adaptive system*, and the existing method, called *baseline*.

In the experiments, different number of instances, such as 1K, 2K and 3K, are used. The purpose of different number of instances was to compare the influence of the instance size on algorithms at the same domain. In the adaptive system, C4.5, BF-Tree and Conjunctive Rules classifier algorithms are attached to the learning module and its results are recorded. We tuned the algorithms to optimize and configured to deliver the highest quality prediction without concern for performance.

Figure 3: Quality of prediction (MAE) using AdaRec (attached J48, BF-Tree and Conjunctive Rules with 1000 instances) vs Baseline & Best.

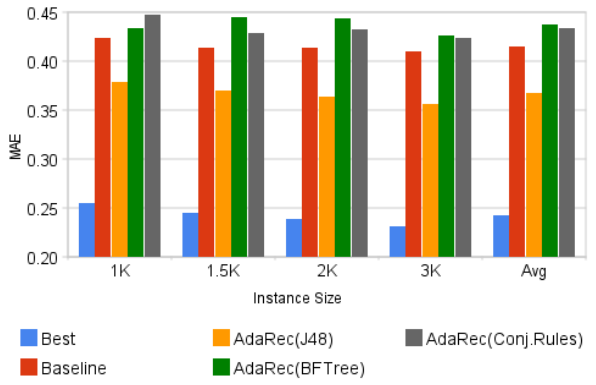


Figure 4: Prediction accuracy comparison of the AdaRec (attached J48, BF-Tree and Conjunctive Rules) vs Baseline & Best for different instance sizes.

We also plot the result of the best MAE (less is better) of the hybrid recommender in the current context at each iteration. The best MAE is referred as *best* at the charts. Therefore it is possible to compare the performance of the algorithms and the best possible result.

Figure 6 presents the prediction quality (average MAE) results of our experiments for the adaptive system as well as the original system referred to baseline. J48 algorithm is used in these experiments. In this chart, prediction quality is plotted for each of the iterations. On each iteration adaptive system re-designs its prediction strategy according to the previous iteration's performance result (feedbacks and results). It can be inferred from chart that, the prediction quality of the adaptive system performs better than the baseline. It can also be observed from the chart that the adaptive system adapts itself to the changes in the domain and users.

Figure 4 presents the prediction accuracy results of the different instance sizes. In the figure, we also plot the overall performance of ML algorithms as average. It can also be observed from the charts that as we increase the instance size of algorithms the quality tends to be superior (decreased MAE). In case of other algorithms it is expected that increasing the number of instances would mean small MAE values. The same trend is observed in the case of 2K and 3K instances.

Figure 5 presents the average MAE of hundred runs for 1K instance size. In this experiment we evaluate the impact of more runs for 1K instance size. It can be observed from the chart that changes in the MAE show the similar trends for both the baseline, adaptive and best systems. A harmony is achieved through time. The curves are similar in such a way that if one of them has a good prediction accuracy in one run, the others also have the good accuracy for that run.

In order to determine the impact of the instance size, we carried out an experiment where we varied the value of instance size (1K, 1.5K, 2K and 3K). For each of these training set/instance size values we run our experiments. Figure 6 presents the whole picture of the adaptive system's performance results. From the plot we observe that the quality of MAE increases as we increase the instance size.

Figure 3 presents the accuracy results of all used ML techniques for 1K instance size. It can be observed from the figure that the J48 attached AdaRec system performs better than the other systems. Also BF-Tree seems good enough to compete against the naive hybrid system. But BF-Tree algorithm needs some domain depended configuration adjustments. From the figure we also observe that the Conjunctive Rules algorithm underperforms among other ML algorithms. The rule learner algorithm seems not stable as the decision tree learners.

The results also show that, when using well tuned algorithms, the adaptive system is stable (better than the baseline) in obtaining the average prediction accuracy. This durability, which can be called the impact of learning, is established by the learning module. In order to adapt recommender engine to the current trends, the learning module re-designs the prediction strategy. The learning ability supports the recommender system adaptation to the changes.

7. CONCLUSION & FUTURE WORK

In this paper, we introduced an adaptive hybrid recommender system, called AdaRec, that combines several recommender algorithms in which the combination parameters are learned and dynamically updated from the results of previous predictions. Research study shows that traditional static hybrid recommender systems suffer from changing user preferences. In order to improve the recommendation performance, we handle domain drifts in our approach. The Learning Module re-designs its prediction (switching) strategy according to the performance of prediction techniques based on user feedbacks. As a result, the system adapts to the application domain, and the performance of recommendation increases as more data are accumulated. In the MovieLens dataset, the proposed adaptive system outperforms the baseline (naive hybrid system).

Initial experimental results show its potential impacts. Therefore, for the next step, we plan to further testing the learning module with various heterogeneous datasets. It would be interesting to examine the different domains other than movie

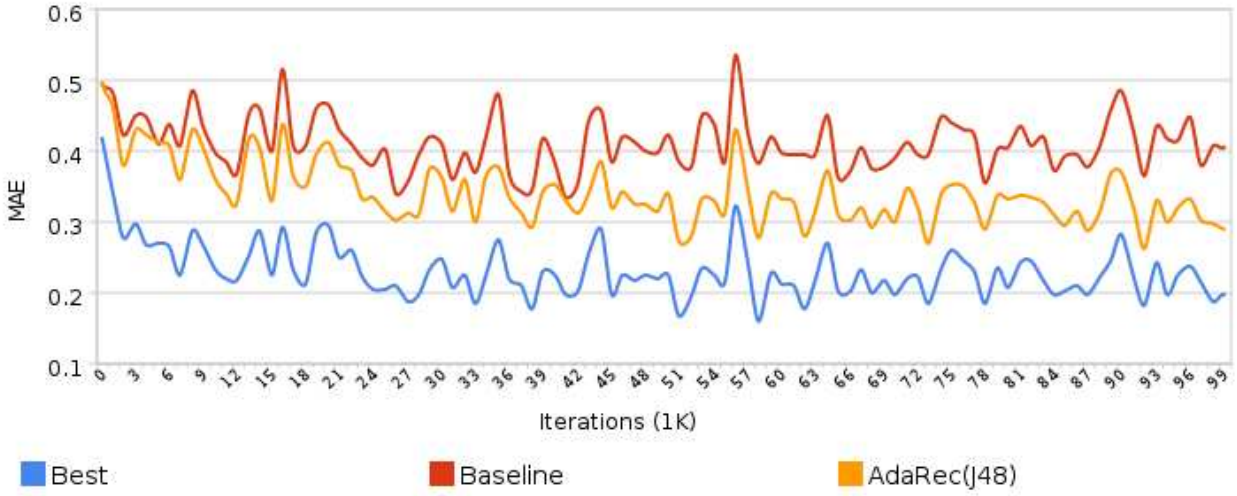


Figure 5: Quality of prediction using J48 (pruned C4.5), Baseline and Best according to 100 iterations. Previous 1000 instances are used for learning on each iteration.

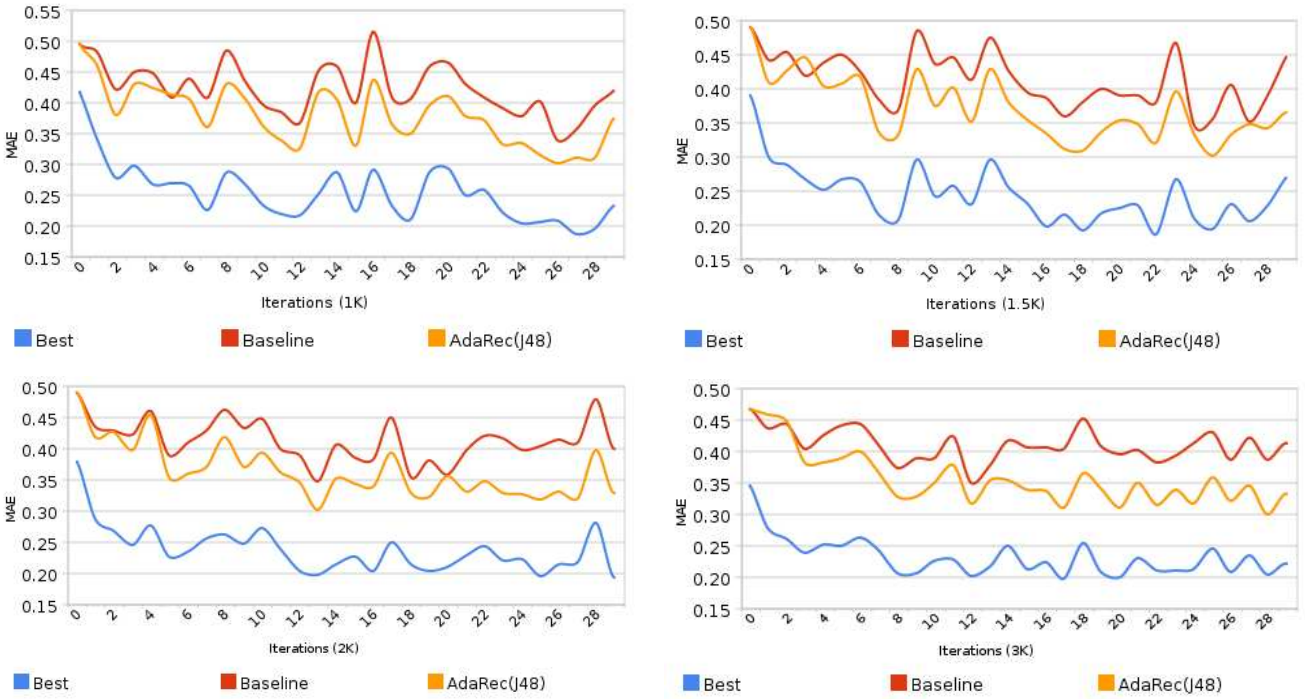


Figure 6: Comparison of the different instance sizes. 1K, 1.5K, 2K and 3K instances are used for learning cycle. 3K learning cycles' prediction quality yields better results than the others

(such as music, book, news etc.). Also, our future work will explore the effectiveness of other machine learning techniques for use in learning module. In our experiments we fixed the used attributes for domain monitoring. It would be also interesting to use dynamic attributes, which means to use different attributes on different iterations. We believe that with this adaptive learning module, a traditional hybrid recommender should have higher chance to allow its users to efficiently obtain an accurate and confident decision.

8. ACKNOWLEDGMENTS

We would like to acknowledge the anonymous reviewers of HaCDAIS'2010 and PRSAT'2010 workshops for their useful comments. An earlier version of this study has appeared as a short paper in the proceedings of HaCDAIS 2010 workshop [2].

9. 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 Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[2] F. Aksel and A. Birturk. An Adaptive Hybrid Recommender System that Learns Domain Dynamics. *International Workshop on Handling Concept Drift in Adaptive Information Systems: Importance, Challenges and Solutions (HaCDAIS-2010) at the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2010 (ECML PKDD 2010)*, page 49, 2010.

[3] X. Bao, L. Bergman, and R. Thompson. Stacking recommendation engines with additional meta-features. In *Proceedings of the third ACM conference on Recommender systems*, pages 109–116. ACM, 2009.

[4] R. Bell, Y. Koren, and C. Volinsky. The bellkor solution to the netflix prize. *KorBell Team's Report to Netflix*, 2007.

[5] D. Billsus and M. Pazzani. User modeling for adaptive news access. *User Modeling and User-Adapted Interaction*, 10(2):147–180, 2000.

[6] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[7] A. Gunawardana and C. Meek. A unified approach to building hybrid recommender systems. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, pages 117–124, New York, NY, USA, 2009. ACM.

[8] J. Herlocker and J. Konstan. Content-independent task-focused recommendation. *IEEE Internet Computing*, pages 40–47, 2001.

[9] The internet movie database. http://www.imdb.com/.

[10] D. Jensen and P. R. Cohen. Multiple comparisons in induction algorithms. In *Machine Learning*, pages 309–338, 1998.

[11] B. Krulwich and C. Burkey. Learning user information interests through extraction of semantically significant phrases. In *Proceedings of the AAAI spring symposium on machine learning in information access*, pages 100–112, 1996.

[12] K. Lang. Newsweeder: Learning to filter netnews. In *In Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[13] P. Melville, R. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Proceedings of the National Conference on Artificial Intelligence*, pages 187–192. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

[14] S. E. Middleton. *Capturing knowledge of user preferences with recommender systems*. PhD thesis, University of Southampton, May 2003.

[15] Movielens dataset. http://www.grouplens.org/node/73.

[16] Netflix dataset. http://www.netflixprize.com.

[17] Novay. Duine recommender — telematica instituut-novay, 2010. [Online; accessed 1-July-2010].

[18] S. Salzberg and A. Segre. Book review: "c4.5: Programs for machine learning" by j. ross quinlan. In *Machine Learning*. morgan kaufmann publishers, inc, 1994.

[19] 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. pages 345–354. ACM Press, 1998.

[20] M. V. Setten. *Supporting People in Finding Information- Hybrid Recommender Systems and Goal Based Structuring*. Telematica instituut fundamental research series no:016, Telematica Instituut, November 2005.

[21] M. V. Setten, M. Veenstra, and A. Nijholt. Prediction strategies: Combining prediction techniques to optimize personalization. In *Proceedings of the workshop Personalization in Future TV*, volume 2. Citeseer, 2002.

[22] U. Shardanand and P. Maes. Social information filtering: algorithms for automating "word of mouth". In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[23] C. Ziegler. *Towards Decentralized Recommender Systems*. Freiburg i. br, Albert-Ludwigs-Universität Freiburg,Germany, June 2005.

Supporting Consumers in Providing Meaningful Multi-Criteria Judgments

Friederike Klan
Institute of Computer Science
Friedrich-Schiller-University of Jena
friederike.klan@uni-jena.de

Birgitta König-Ries
Institute of Computer Science
Friedrich-Schiller-University of Jena
birgitta.koenig-ries@uni-jena.de

ABSTRACT

The huge amount of products and services that are available online, makes it difficult for consumers to identify offers which are of interest to them. Semantic retrieval techniques for Web Services address this issue, but make the unrealistic assumption that offer descriptions describe a service's capabilities correctly and that service requests reflect a consumer's actual requirements. As a consequence, they might produce inaccurate results. Alternative retrieval techniques such as collaborative filtering (CF) mitigate those problems, but perform not well in situations where consumer feedback is scarce. As a solution, we propose to combine both techniques. However, we argue that the multi-faceted nature Web Services imposes special requirements on the underlying feedback mechanism, that are only partially met by existing CF solutions. The focus of this paper is on how to elicit consumer feedback that can be effectively used in the context of Web Service retrieval and how to support users in that process. Our main contribution is an algorithm that suggests which service aspects should be judged by a consumer. The approach effectively adjusts to user's ability and willingness to provide judgments and ensures that the provided feedback is meaningful and appropriate in the context of a certain service interaction.

Categories and Subject Descriptors

H.3.3 [**Information Storage And Retrieval**]: Information Search and Retrieval; H.3.5 [**Information Storage And Retrieval**]: On-line Information Services

General Terms

Algorithms, Human Factors, Measurement

Keywords

recommend what to judge, multi-criteria judgments, personalized feedback elicitation

1. INTRODUCTION

The huge amount and heterogeneity of information, products and services that are available online, makes it difficult for consumers to identify offers which are of interest to them. Hence, new techniques that support users in the product search and selection process are required. In the past decade, semantic technologies have been developed and leveraged to approach this issue [3]. They provide information with a well-defined and machine-comprehensible meaning and thus enable computers to support people in identifying relevant content. This idea is not restricted to information, but also applies to functionality provided via the web as services. Semantic Web Services (SWS) provide a specific functionality semantically described in a machine-processable way over a well-defined interface. Similarly, service requesters may semantically express their service requirements. Having both, a semantic description of a consumer's needs as well as the published semantic descriptions of available Web Services, suitable service offers can be automatically discovered by comparing (matching) the given service request with available offer descriptions. Services might be automatically configured and composed and finally invoked over the web.

Existing semantic matchmaking and service selection approaches evaluate the suitability of available service offers exclusively by comparing the published offer descriptions with a given request description. They implicitly assume that offer descriptions describe a service's capabilities correctly and that service requests reflect a consumer's actual requirements. The first assumption might have been valid in a market with a small number of well-known and accredited companies. However, it is no longer true in today's market, where easy and cheap access to the Internet and the emergence of online marketplaces that offer easy to set up online storefronts enable virtually everyone to provide his own online shop accessible to millions of buyers. The situation becomes even more critical, since due to the huge number of offers, a hard competition and price war has been aroused that might cause some providers to promise more than they are able to provide. In our mind, the assumption that service requests reflect a consumer's actual requirements is also not realistic. This is due to the fact that, though SWS approaches provide adequate means to semantically describe service needs, they require the user to do this at a formal, logic-based level that is not appropriate for the average service consumer in an e-commerce setting. As a result, SWS applications typically provide request templates for common service needs. Those templates are then adjusted to fit to a consumer's requirements in a certain purchasing situation.

Though the resulting service requests might be a good estimate of a consumer's service needs, they cannot exactly met his true requirements. As a consequence, service discovery mechanisms that are purely based on the comparison of semantic request and offer descriptions might produce inaccurate results and thus lead to suboptimal service selection decisions.

To mitigate those problems, alternative retrieval techniques such as collaborative filtering [9] have been developed. Those techniques do not rely on explicit models of consumer requirements and product properties. They evaluate product ratings of neighboring users, i.e. those that have a similar taste, to recommend products or services that might be of interest to a potential consumer. Though collaborative filtering approaches are very effective in many domains, they lack the powerful knowledge representation and matchmaking capabilities provided by SWS and thus perform not well in situations where feedback is scarce [9]. As a solution, we propose to combine both techniques. More specifically, we suggest to perform retrieval based on semantic service descriptions and then use a collaborative feedback mechanism to verify and refine those results. We think, that such a hybrid approach can benefit from the best of both worlds and thus has the potential to significantly improve the retrieval quality. Combining semantic retrieval with collaborative feedback mechanisms is not new (see for example [8, 11]). However, we argue that simply re-using existing techniques, as done in other approaches, will not tap the full potential of this type of approach. This is due to the fact, that the multi-faceted nature and the peculiarities of SWS impose special requirements on the underlying feedback mechanism and in particular on the properties of the consumer feedback that is required. In this paper, we will analyze those requirements (Sect. 2) and will show that they are only partially met by existing collaborative filtering solutions (Sect. 3). The focus of this paper is on how to elicit consumer feedback that can be effectively used in the context of SWS retrieval and how to support users in that process (Sects. 4 and 5). Our main contribution is an algorithm that suggests which service aspects should be judged by a consumer (Sect. 6). The approach accounts for a user's ability and willingness to provide judgments and ensures that the provided feedback is meaningful and appropriate in the context of a certain service interaction. Our evaluation results show that the proposed procedure effectively adjusts to a consumer's personal judgment preferences and thus provides helpful support for the process of feedback elicitation (Sect. 7). A detailed discussion on how to effectively use consumer feedback to enhance SWS retrieval is published in [6].

2. REQUIREMENTS

Various collaborative filtering mechanisms that allow to retrieve products or services that are of interest to a consumer [9] have been proposed. Those mechanisms are very effective in many domains and seem to be very promising in the context of our work. However, we argue that the multi-faceted nature of SWS imposes special requirements on the underlying feedback mechanism, that are only partially met by existing CF solutions. In the following, we will specify those requirements.

Consumer feedback is *subjective*, since it reflects a service's suitability as perceived through a certain consumer's eyes. Hence, feedback is biased by personal expectations and preferences about the invoked service. Moreover, feedback may refer to different services and to different request contexts. For example, a ticket booking service might have been used to buy group tickets for a school class or to buy a single ticket. However, the suitability of a service might differ depending on the request context and hence the resulting feedback also does. Feedback mechanisms should account for those facts. To enable effective usage, feedback has to be *meaningful*, i.e., the expectations and the context underlying a judgment should be clear. In addition, it should be evident whether and how feedback made under one circumstance can be used to infer about a service's suitability in another situation.

We would also like to emphasize the necessity of feedback to be as *detailed* as possible, i.e. comprising of judgments referring to various aspects of a service interaction. This is for several reasons. Firstly, feedback, judging the quality of a provided service as a whole, is of limited significance, since as an aggregated judgment it provides not more than a rough estimate of a service's performance. Secondly, aggregated feedback tends to be inaccurate. This is due to the fact, that humans are bad at integrating information about different aspects, as they appear in a multi-faceted service interaction, in particular if those aspects are diverse and incomparable [2, 10]. Finally, it has been shown in [4] that using detailed consumer feedback allows to estimate user taste's more accurately and thus can significantly improve prediction accuracy. In the context of detailed, i.e. multi-criteria, consumer feedback, meaningful also means that the relationship between different service aspects that might have been judged is clear and that all relevant aspects characterizing a certain service interaction have been judged. The latter is due to the fact, that inferred judgments based on incomplete information might be incorrect.

Another problem we encounter is feedback *scarcity*. Given certain service requirements, a certain context and a particular service, feedback for exactly this set-up is rare and typically not available at all. Hence, scarce feedback has to be exploited effectively. In particular, service experiences related to different, but similar contexts and those related to other, but similar services have to be leveraged. However, unfolding the full potential of consumer feedback, in particularly when using multi-aspect feedback, requires that users provide useful responses. To ensure this, the feedback elicitation process should be assisted. In particular, it should be taken care that elicited feedback is comprehensive and *appropriate* in the context of a certain service interaction. In addition, a consumer's *willingness* to provide feedback as well as his *expertise* in the service domain should be accounted for. This is important, since asking a consumer for a number of judgments he is not able and/or not willing to provide will result in no or bad quality feedback. Finally, it should also be ensured that all relevant information that are necessary for effectively exploring consumer feedback are recorded. This should happen transparently for the user.

Since the type of service interactions to be judged and the kind of users that provide feedback are diverse and not known in advance, even for a specific area of application, a hard-wired solution with predefined service aspects to judge is inappropriate. In fact, the process of feedback elicitation should be *customizable* and should be *automatically configurable* at runtime.

3. RELATED APPROACHES

Aspects such as feedback scarcity and subjectivity of consumer feedback are typically addressed in existing collaborative filtering solutions [9]. Also, dealing with the context-dependent nature of judgments has been an issue (see e.g. [1]). However, existing solutions only partially address the question of how to effectively use judgments made in one context to infer about a service's suitability in another context. Multi-criteria feedback has been an issue in both academic [4] and commercial recommender systems. Typically, the set of aspects that might be judged by a consumer is either the same for all product types or specific per product category. However, in the first case, this set of aspects is either very generic, i.e. not product-specific, or not appropriate for all products. In the second case, this set has to be specified manually for each new product. Moreover, typically the single aspect ratings are supplementary in the sense that they do not have any influence on a product's overall rating. Alternatively, some reviewing engines such as those provided by Epinions [1]or Powerreviews[2], offer more flexible reviewing facilities based on tagging. Those systems allow consumers to create tags describing the pros and contras of a given product. These tags can then be reused by other users. Tagging provides a very intuitive and flexible mechanism that allows for product-specific judgments. However, the high flexibility of the approach is at the cost of the judgments' meaningfulness. This is due to the fact that tags do not have a clear semantics. In particular, the relationship between different tags is unknown and thus makes them incomparable. Moreover, those systems do not ensure that all relevant aspects of a product or a service interaction are judged. To summarize our findings, more flexible and adaptive mechanisms to elicit and describe multi-criteria feedback are required. In particular, the question of how to describe this type of feedback meaningfully has been hardly considered. To the best of our knowledge, the issue of assisting consumers in providing comprehensive, appropriate and meaningful feedback has not been addressed at all. Also, aspects such as a consumer's ability and willingness to provide judgments for specific aspects have been hardly considered in existing solutions.

4. SEMANTIC WEB SERVICE RETRIEVAL

As a basis for further discussion, we introduce the semantic service description language DSD (DIANE Service Description) [7] and its mechanisms for automatic semantic service matchmaking that underlie our approach. Similarly to other service description approaches, DSD is ontology-based and describes the functionality a service provides as well as the functionality required by a service consumer by means of the precondition(s) and the set of possible effect(s) of a service execution. In the service request depicted in Fig. 1, the desired effect is that a product is owned after service execution. A single effect corresponds to a particular service instance that can be executed. While service offer descriptions describe the individual service instances that are offered by a service provider, e.g. the set of mobile phones offered by a phone seller, service request descriptions declaratively characterize the set of service instances that is acceptable for a consumer. In the service request in Fig. 1, acceptable instances are mobile phones that are cheaper than 50$, are either silver or black, are of bar or slider style and are from either Nokia or Sony Ericsson. As

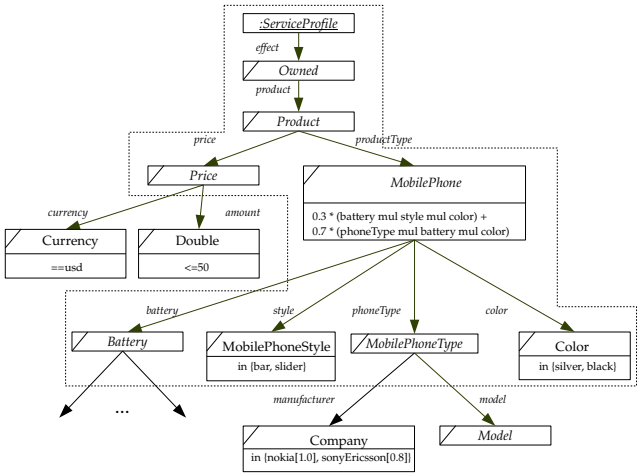


Figure 1: DSD service request

can be seen in the example, DSD utilizes a specific mechanism to declaratively and hierarchically characterize (acceptable) sets of service effects: Service effects are described by means of their attributes, such as price or color. Each attribute may be constrained by direct conditions on its values and by conditions on its subattributes. For instance, the attribute phoneType is constrained by a direct condition on its subattribute manufacturer, which indicates that only mobile phones from Nokia or Sony Ericsson are acceptable. The direct condition $<= 50$ on the price amount in Fig. 1 indicates that only prices lower than 50$ are acceptable. Attribute conditions induce a tree-like and more and more fine-grained characterization of acceptable service effects. A DSD request does not only specify which service effects are acceptable, but also indicates to which degree they are acceptable. In this context, a preference value from $[0, 1]$ is specified for each attribute value. The default is 1.0 (totally acceptable), but alternative values might be specified in the direct conditions of each attribute. For example, the preference value for the attribute manufacturer is 1.0 for Nokia phones and 0.8 for mobile phones from Sony Ericsson.

As demonstrated in [7], DSD service and request descriptions can be efficiently compared. Given a service request, the semantic matchmaker outputs an aggregated overall preference value $\in [0, 1]$ for each available service offer description. This value is called *matching value* and indicates how ell a considered service offer fits to a consumer's requirements encoded in the service request. Based on the matching values, the best fitting service offer is determined and invoked.

5. FEEDBACK ELICITATION

In the following, we will analyze what is required to make detailed consumer feedback meaningful, comprehensive and appropriate to characterize a certain service interaction. We will demonstrate how semantic service descriptions can be used to elicit feedback that fulfills those requirements. A detailed discussion on how to effectively use the elicited consumer feedback to enhance SWS retrieval is out of the scope

[1]http://www.epinions.com
[2]http://www.powerreviews.com

of this paper and is published in [6].

What is required to make consumer feedback appropriate, comprehensive and meaningful.

We assume, that a service request at least covers all service aspects that are important to the consumer. Potentially, all service aspects in a request description might be rated by a consumer. In order to be able to exploit these ratings, we need to make sure that they are meaningful (i.e., contain the rating context, e.g., which product a rating refers to) and comprehensive (i.e., contain all relevant aspects, a quality rating without information whether the price was ok is not helpful). In addition, we need to know how different service aspects relate to each other (e.g., how can a rating about quality be gained from ratings on subaspects such as usability and battery capacity?). The challenging question is how to fulfill the identified requirements while still being flexible in the choice of the aspects to rate.

Creating appropriate, comprehensive and meaningful consumer feedback.

We propose the concept of a *feedback structure* to deal with that issue. A feedback structure is a subtree of the request tree, whose leaves correspond to the aspects that may be rated by the user. Consider the example request depicted in Fig. 1. The dotted part of the tree indicates a possible feedback structure for that request, where the aspects price, battery, style, color and phoneType have to be rated by the consumer. Note that this structure contains all information that are necessary to effectively utilize the provided ratings. In particular, it encodes the context of a rating in terms of the path from the request root to the rated aspect, the other aspects that were judged and the hierarchical relationship between the considered aspects.

To assure that the provided feedback is comprehensive, the request subtrees rooted at the feedback structure's leaves should cover all leaves of the tree. This guarantees that all service aspects considered in the request description are either directly or indirectly (by providing an aggregated rating) judged by the service consumer. The feedback structure depicted in Fig. 1 fulfills this requirement and thus is valid. Omitting, e.g., the aspect phoneType would result in an invalid structure. Note, that we are still flexible in the choice of the attributes to be rated, e.g. we could allow the consumer to provide a single rating for productType instead of asking him to judge battery, style, color and phoneType separately. The feedback structure together with the consumer provided ratings are propagated to other consumers and might be used to infer knowledge about a service's suitability for consumers with other service requirements (see [6] for details).

6. RECOMMENDING WHAT TO JUDGE

To ensure feedback quality, the feedback elicitation process should be assisted and should account for a consumer's judgment preferences such as his willingness to provide ratings as well as his expertise in the considered service domain. However, those judgment preferences might differ from request to request, e.g. I might be an expert in judging the quality of Personal Computers, but I do not know that much about servers. As a consequence, I like to/I'm able to judge the quality of a purchased computer, in case of a PC, but I'm not willing to do that when purchasing a new server for our working group. This aspect should be considered during feedback elicitation. To achieve this, we propose the following solution.

Assume, that given a certain service request, an appropriate service was selected and invoked and now its suitability has to be judged by the consumer. In a first step, we utilize the provided service request to determine possible feedback structures as defined in the previous section. Subsequently, the structure that is most suitable for the user, i.e. in the context of the given request, fits best to the consumer's personal abilities and judgment preferences, is selected and presented to the user. The required knowledge about the user's judgment requirements is learned from the his behavior in previous judgment sessions. The presented feedback structure represents a careful compromise between the consumer's competing judgment requirements and might be adjusted to his actual judgment needs. This can be done by expanding and/or hiding subtrees of the presented structure. For example, in the structure depicted in Fig. 1, we might expand the leaf phoneType to judge its subaspects manufacturer and model. Finally, the user judges all leaf attributes of the structure, e.g. by providing a rating. Once, the consumer submits his judgments, the system takes care of storing all relevant feedback information and session data for future recommendations. In particular, it is recorded which and how many service aspects were judged by the consumer and which service request lead to the judgment. The acquired information are used later on to identify suitable feedback structures in future judgment sessions.

6.1 Feedback structure suitability

Given a consumer's service request, typically many different feedback structures are possible. However, how to measure the suitability of each feedback structure to identify one that fits best to the user's personal abilities and willingness to provide judgments? We have to consider two aspects here. Firstly, comprise the feedback structure leaves of those attributes that the consumer's is able to judge and secondly, is the consumer willing to judge all those aspects?

As a measure of a consumer's willingness and ability to judge a certain service aspect, we us the frequency with which the user judged this aspect in the past. We also consider the request context in which an aspect was judged. More specifically, we consider how similar the request that lead to the past judgment is to our request. Let r be the service request that was posed by the consumer. Then the consumer's willingness and ability to judge service aspect a is determined by $w_a(r) = \sum_{r' \in R_a} sim(r', r)$, where R_a is the set of past service requests that lead to a judgment of a. The value $sim(r', r)$ indicates how similar the service requirements encoded in the past request r' are to those in current request r. A detailed discussion on how compute the semantic similarity of two requests is provided in Sect. 6.3.

The suitability $s_{attributes}(fs, r)$ of a given feedback structure *fs* is determined by the consumer's willingness and ability to judge its leaf aspects A_{fs}. We propose to compute it as the sum of its leaf attributes' w_i-values.

$$s_{attributes}(fs, r) = \frac{\sum_{i \in A_{fs}} w_i(r)}{\sum_{j \in A_r} w_j(r)} \quad (1)$$

The term is normalized by dividing it by the sum of the w_j-values of all attributes $j \in A_r$ that are contained in the

given request r. Hence, $s_{attributes}(fs, r) \in [0, 1]$.

To measure a consumer's willingness to judge $k = |A_{fs}|$ leaf aspects A_{fs}, we compare how similar the past requests that also led to a judgment of k aspects are to the service request r posed by the consumer. More specifically, the suitability $s_{number}(fs, r)$ of the feedback structure fs with respect to the number of service aspects that have to be judged is determined by

$$s_{number}(fs, r) = \overline{sim}(R_k, r), \tag{2}$$

where $\overline{sim}(S_k, r)$ is the mean request similarity of all past service requests that lead to a judgment of k aspects. In cases, where no previous requests lead to a number of k service aspects to be judged, $s_{number}(fs, r)$ is determined as the mean of $\overline{sim}(R_{k'}, r)$ and $\overline{sim}(R_{k''}, r)$, where k' is the largest $k' < k$ for which a past request with k' judgments exists and k'' is the smallest $k'' > k$ for which a past request with k'' judgments exists. In case, k'/k'' did not exist, $\overline{sim}(R'_k, r)/\overline{sim}(R''_k, r)$ was assumed to be 1.0/0.0, i.e. by default feedback structures with a low number of service aspects to be judged are preferred. Assuming that $\overline{sim}(x, y)$ is a value from $[0, 1]$, $s_{number}(fs, r)$ is also from $[0, 1]$. The overall suitability $s(fs, r) \in [0, 1]$ of a feedback structure fs in the context of the posed request r is

$$s(fs, r) = \alpha \cdot s_{attributes}(fs, r) + \beta \cdot s_{number}(fs, r). \tag{3}$$

The parameters α and β with $\alpha, \beta \in [0, 1]$ and $\alpha = 1 - \beta$ determine the influence of the terms $s_{attributes}(fs, r)$ and $s_{number}(fs, r)$, respectively. The values α and β might vary from user to user. In Sect. 6.4, we will demonstrate how those values can be learned from a consumer's past judgment behavior.

6.2 Determining possible feedback structures

For a given request, the number of possible feedback structures might be high, whereas the number of those that have the potential to be optimal (with respect to their suitability $s(fs, r)$ for the user) is low. Hence, we require a way to determine potentially optimal feedback structures effectively, i.e. without having to construct all possible structures. In the following, we propose an algorithm that performs this task. It constructs potentially optimal feedback structures recursively and drops non-optimal partial structures as soon as possible. Fig. 2 shows how the algorithm

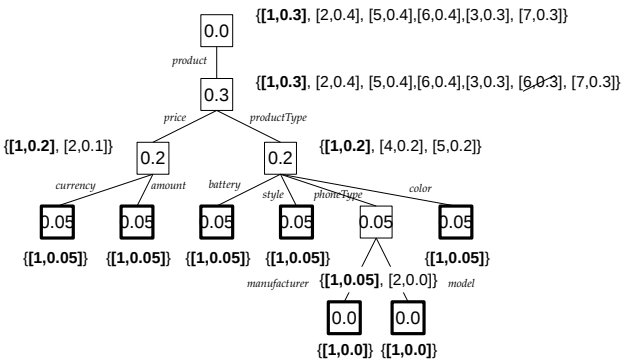


Figure 2: Determining possible feedback structures

works, exemplary for the service request depicted in Fig. 1. Each request node is associated with a list of entries, each corresponding to one of the feedback structures that are possible for the subtree rooted at that node. Let fs be one of those structures and let $[a, b]$ be its corresponding entry. Then a is the number of aspects that have to be judged in fs and b is $s_{attributes}(fs, r)$, where r is the request subtree rooted at the considered node. The algorithm works as follows. First, it initializes each request node's list with an entry $[1, s_{attributes}(fs, r)]$, where fs is the feedback structure comprising only of the node itself and r is the request subtree rooted at the considered node. For an example, consider Fig. 2. The initial entry in each list is highlighted. The number within each node indicates the value $s_{attributes}(fs, r)$, which, for the sake of this example, is arbitrarily chosen. Starting from the request leaves (highlighted request nodes), the algorithm recursively computes lists for all parent nodes. Computing a node's list is done in three steps. First, the cross product C of the child nodes' entry sets is computed. For example to determine possible feedback structures for the product-node (Fig. 2), we have to determine $C = [1, 0.2], [2, 0.1] \times [1, 0.2], [4, 0.2], [5, 0.2]$, i.e. the cross product of the price and productType node's entry lists. Each element c of C gives rise to an entry $[a, b]$ in the product-node list, i.e. to a possible feedback structure fs of this node's subtree. Since a is the number of attributes to judge in fs, it is computed as the sum of the a values in c. The suitability b of fs with respect to the selection of attributes that have to judged is computed as the sum of its leaf attributes' b values (Formula 1), i.e. the sum of the b values in c. In a final step, we prune the computed list. This is done by keeping only a single entry $[a, b]$ for each different value of a per node, where $b = \max\{x | [a, x] is in the list\}$. Note, that in doing so, we keep only those feedback structures that have the potential to be optimal and hence reduce the length of the node list to at most l, where l is the number of leaves of the subtree rooted at the considered node. Finally, we end up with a list for the request root comprising of entries for all possible feedback structures for the request, that have the potential to be optimal. Those structures are compared with respect to their suitability (Formula 3). The most suitable is selected and presented to the user.

6.3 Request similarity

As mentioned earlier, a consumer's judgment preferences depend on the request context, i.e. the kind of service interaction, that has to be judged. To allow for a comparison of the request contexts, in which judgments have been made in the past, with the current request, we require a measure for the semantic similarity of two requests, i.e. the similarity of the service requirements they encode. In this section, we will propose such a measure. It recursively computes the similarity $sim(r, r')$ of two request trees r and r' by computing the similarity of their root nodes' ontological type $(sim_{type}(root(r), root(r')))$ and direct conditions $(sim_{dc}(root(r), root(r')))$ and the aggregated similarity of their root nodes' child trees $(sim_{attr}(root(r), root(r')))$. More specifically, we define $sim(r, r')$ to be the mean of these three values. In the remainder of the section, we will explain the rationale between those three similarity values and particularize on how to determine them. Possible similarity values $sim(r, r')$ are from the interval $[0, 1]$, where a similarity value of 0.0 means "not similar at all" and a value of 1.0 means that the service requirements encoded by two requests are

identical.

Determining the type similarity.

The type similarity $sim_{type}(n, n') \in [0, 1]$ of two nodes n and n' indicates how similar those nodes are with respect to their ontological type. It is defined similar to Jaccard's index [5], that is often used to compare the sample sets,

$$sim_{type}(n, n') = \frac{|A_n \cap A_{n'}|}{|A_n \cup A_{n'}|} \quad (4)$$

where A_n is the set of attributes defined for the type of n and $A_{n'}$ is the set of attributes defined for the type of n'. The type similarity $sim_{type}(n, n')$ for the root nodes of the requests depicted in Fig. 3 is $\frac{|\{battery, phoneType, color\}|}{|\{battery, phoneType, color, style\}|} = 0.75$.

Determining the similarity of the direct conditions.

The similarity $sim_{dc}(n, n') \in [0, 1]$ of two nodes n and n' indicates how similar those nodes are with respect to their direct conditions. As mentioned in Sect. 4, direct conditions restrict acceptable values of a service attribute. For each kind of direct condition that might be specified for a certain attribute, we define a separate similarity measure. For example, for direct conditions of type $IN\{\ldots\}$, the similarity is determined as the quotient of the number of common values divided by the number of values that are allowed for n or n'. For direct conditions of type $<= x$ and $>= x$, the similarity is calculated as $\min\{x, y\}/\max\{x, y\}$, where x is the upper/lower bound for the values of n and y for those of n'. Accordingly, if only one of the nodes specified a certain type of direct condition, the similarity is defined to be 0.0 and if both nodes do not specify any direct conditions, the similarity is defined to be 1.0.

As an example, consider again the requests depicted in Fig. 3. The Color-nodes both specify a direct condition of type $IN\{\ldots\}$. The similarity $sim_{dc}(n_{color}, n'_{color})$ with respect to this direct condition is $1/2 = 0.5$. The Battery-nodes both do not specify any direct conditions, hence $sim_{dc}(n_{battery}, n'_{battery}) = 1.0$.

Determining and aggregating the similarity of the root nodes' child trees.

The similarity value $sim_{attr}(n, n') \in [0, 1]$ indicates how similar two nodes n and n' are with respect to their child trees. Let A be the set of attributes defined either for the type of n, the type of n' or for both types and let $\{sim(r_a, r'_a) | a \in A\}$ be the similarity values for corresponding attribute subtrees r_a and r'_a of n and n'. Again, inspired by Jaccard's index, the aggregated similarity of two nodes' child trees is defined as the sum of the similarity values $\{sim(r_a, r'_a) | a \in A\}$ divided by the sum of the maximal similarity values that can be achieved for each attribute, i.e. $|A|$.

$$sim_{attr}(n, n') = \frac{\sum_{a \in A} sim(r_a, r'_a)}{|A|} \quad (5)$$

Since attributes in A are not necessarily defined for both, the type of n and n', we set $sim(r_a, r'_a) = 0.0$, if the attribute a is not defined for one type. Attributes in A might also not be specified in one or both of the nodes. If an attribute a is not specified in both nodes, we set $sim(r_a, r'_a) = 1.0$, else, if a is specified in just one of the nodes, $sim(r_a, r'_a)$ is defined to be $sim(r_a, t')$ resp. $sim(t, r'_a)$, where t is a tree comprising of a single node, having the most generic type defined for a in the ontology. We illustrate the procedure for the root nodes of the two request fragments depicted in Fig. 3. The type of r's root node is MobilePhone and that of r''s root node is Phone. Assume, that the ontology defines the attributes battery, phoneType and color for the type Phone and an additional attribute style for the type MobilePhone, which is a subtype of Phone. The similarity $sim_{attr}(n, n')$ of the requests' root nodes n and n' is determined by the similarity of their corresponding child trees for the attributes $A = \{$battery, phoneType, color, style$\}$. The attributes battery and color are specified in both requests, hence the similarity values $sim(r_{battery}, r'_{battery})$ and $sim(r_{color}, r'_{color})$ can be computed by determining the request similarity for the request subtrees rooted at the Battery-nodes and the subtrees rooted at the Color-nodes. The attribute style is only defined for the type MobilePhone, hence $sim(r_{style}, r'_{style}) = 0.0$. The attribute phoneType is defined for both types, MobilePhone and Phone, but only specified in r. Hence, $r'_{phoneType}$ has to be replaced by a node t' having the most generic type defined for the attribute phoneType. Let PhoneType be this type. This means that the type of the node that describes the attribute phoneType has to be PhoneType or one of its subtypes. Presume that MobilePhoneType is a subtype of PhoneType. The similarity $sim(r_{phoneType}, r'_{phoneType})$ is determined by computing $sim(r_{phoneType}, t')$, where $r_{phoneType}$ is the subtree of r rooted at the MobilePhoneType-node.

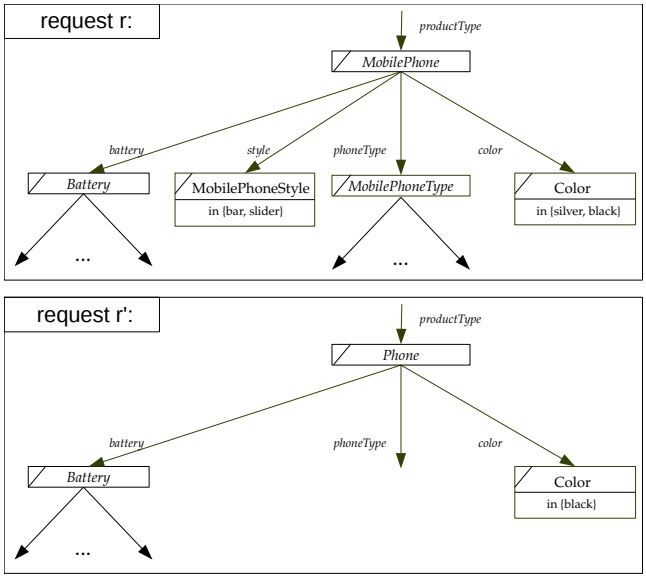


Figure 3: Determining the similarity of two requests r and r'

6.4 Dynamically adjusting α and β

As discussed earlier, the parameters α and β that weight the influence of the terms $s_{attributes}(fs, r)$ and $s_{number}(fs, r)$, might vary from user to user. In this section, we will demonstrate how those values can be learned from a consumer's past judgment behavior. Initially, i.e. without having information about a user's previous judgment behavior, we do not know anything about those parameters' values, so α could be any value from the interval $[0, 1]$ and $\beta = 1 -$

α. Hence, for the purpose of computing the suitability $s(fs, r)$ of possible feedback structures, we set α to the midpoint of this interval, i.e. $\alpha = 0.5 = \beta$. Once having determined the most suitable feedback structure *fs*, we present it to the consumer, who has the opportunity to change it by expanding/collapsing nodes. Finally, the consumer provides judgments for the resulting structure's leaf nodes. Obviously, the resulting feedback structure *fs'* was more suitable to the user than the structure *fs* that was recommended. Hence, we conclude that $s(fs', r)$ should be larger than $s(fs, r)$. Using Formula 3, we get that $s(fs, r) - s_{number}(fs', r)/s_{attributes}(fs', r) - s_{number}(fs', r) < \alpha$ for $s_{attributes}(fs', r) > s_{number}(fs', r)$ and $> \alpha$ for $s_{attributes}(fs', r) < s_{number}(fs', r)$. Using those information, we can adjust, i.e. shrink the range of α correspondingly. For example, if we get $\alpha < 0.8$, we adjust the interval to $[0, 0.8)$. In case, the consumer's judgment behavior is inconsistent, e.g. having $\alpha \in (0.5, 0.7)$, we get $\alpha < 0.8$, we simply ignore those information. To ensure, that the most recent information have the most influence, we process session data in the order of increasing age.

7. EVALUATION

In the evaluation of our approach, we wanted to find out how fast the recommendation algorithm proposed in Sect. 6 adjusts to different judgment preferences.

Test setting.

For that purpose, we created a set of DSD service requests covering typical requirements of consumers looking for computer items from different categories, such as desktop PCs, PDAs, servers, notebooks or organizers. For our tests, we created 48 service requests, 6 per category. Requests within each category varied in the selection of attributes that were specified and in the range of attribute values that were acceptable for the user. All request types shared common attribute types, e.g. for all kinds of requests an attribute color and an attribute price could be specified.

Using this requests we performed several tests with a single test user. The basic procedure for each test was as follows. Starting with no information about previous judgment behavior, several judgment sessions were performed. During each session, one of the 48 requests was selected. After that, the system proposed a feedback structure using the algorithm proposed in Sect. 6 with knowledge about the user's judgment behavior in the previous judgment sessions. After being provided with the recommended feedback structure, the user had the opportunity to change this structure. For that purpose, the consumer was allowed to expand/collapse feedback structure nodes. By clicking on a particular node, all its direct children were expanded/collapsed. The quality of the proposed feedback structure was measured as the edit distance between the proposed feedback structure and the actual feedback structure that was used. More formally, we counted the number of expand/collapse operations the user had to perform to get the structure whose leaves he finally judged. The rationale behind this measure is, that the edit distance is a direct measure of the users effort to get to the desired structure and thus, in our opinion, is a good measure for the quality of the recommended structure. For each of the test, we looked at whether and how fast the edit distance decreased with the number of judgment sessions.

Test runs and results.

We performed test runs with different judgment preferences and different sets of requests that were posed during a sequence of sessions. In a first series of tests, the requests within each sequence of sessions were different, but chosen from a single (computer) category, e.g. just notebook requests. This test setting served as a baseline and was chosen to evaluate the performance of our approach in the absence of any context effects. We performed three kinds of tests differing in the judgment preferences of the judging user. In test $A1$, the consumer always judged a certain number of aspects. However, the types of aspects that were judged differed. In test $A2$, the user judged a different number of attributes during each session, but required that the set of attributes to judge contained a certain set of attributes. For example, a user might require to always judge the price of a product, but is also willing to rate other service aspects. Finally, we performed a test $A3$, were the consumer had specific requirements on both, the number and kind of aspects to judge. The tests $A1$-$A3$ were performed with request sets from different categories. The plot depicted in Fig.4 (A2) is representative for all test runs and all types of tests in this series. It shows the results for test $A2$ performed with requests from the category digital watches. As can be seen, the adaptation of the recommendation algorithm to the consumers judgment preferences is very fast. The initial edit distance decreases to 0 after just one session. This is due to the fact, that request similarity does not play a role in those tests and hence the values of α and β can be arbitrarily chosen. The depicted behavior was observed for all three kinds of tests ($A1$-$A3$).

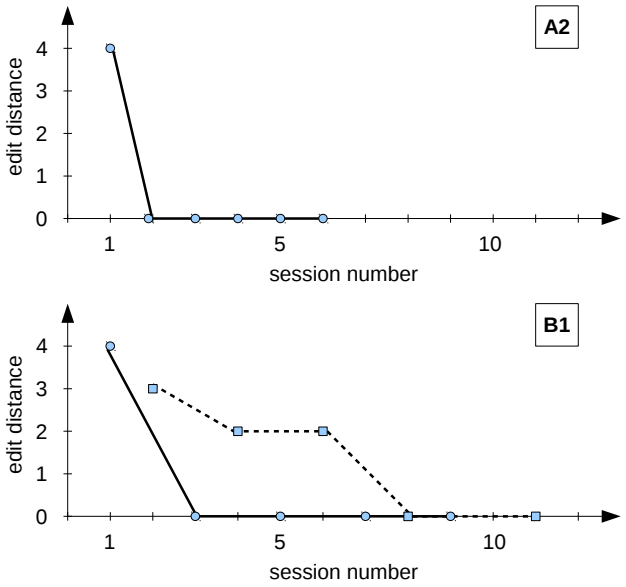


Figure 4: Results of the tests $A2$ and $B1$

In a second series of tests, we evaluated how fast the proposed recommendation algorithm adjusts to a consumers judgment preferences, if those depend on the request context. For that purpose, we performed judgment sessions, were the user posed requests from different categories and exhibited a different judgment behavior for each category. We run three types of tests. In test $B1$, similarly to test

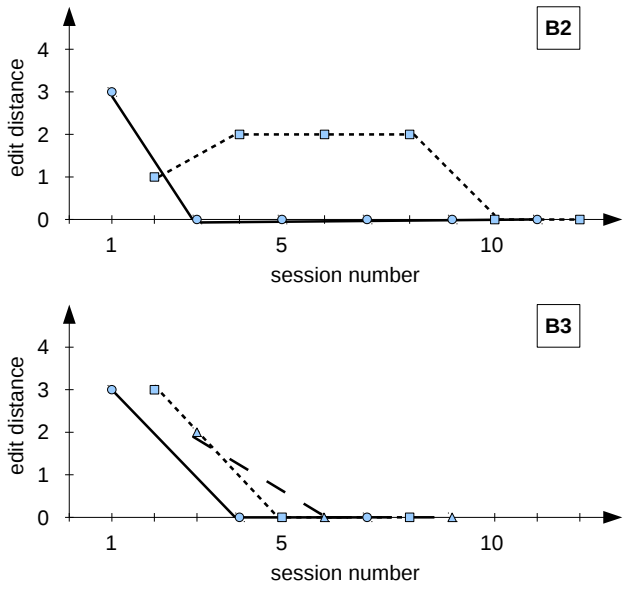


Figure 5: Results of the tests $B2$ and $B3$

$A1$, the user always judged a particular number of aspects. However, this number differed for each request category. For example, a user might always judge 3 aspects when asking for desktop PCs, but is willing to judge 5 service aspects, when asking for notebooks. Analogously to test $A2$, the user in test $B2$ required the set of aspects to be judged to contain a particular set of aspects. However, this set varied for different request categories. Finally, we performed a test $B3$, were the consumer had specific requirements on both, the number and kind of aspects to judge. Those requirements were different for each request category. Fig. 4 (B1) exemplary shows the results for tests of type $B1$. In the depicted test, we alternated sessions based on a request for a desktop PC (continuous line), where the user judged 11 service aspects, with those based on a request for a PDA (dotted line), where the consumer judged only one aspect. As can be seen, the adjustment to the consumer's judgment preferences for PDAs takes 3 sessions. This is due to the fact, that at the beginning both terms $s_{attributes}$ and s_{number} are equally weighted. Since for desktop PCs many aspects are judged and since most of those aspects are also shared by PDA requests, term $s_{attributes}$ dominates the suitability value and thus favors improper feedback structures. This changes when α and β adjust over time. Fig. 5 (B2) exemplary shows the results for tests of type $B2$. Again, we alternated desktop PC requests with those for a PDA. While when judging desktop PCs, we had a set of two aspects that had to be judged in any case, it was only one specific aspect when judging PDAs. Again, it required 4 sessions to adjust α and β appropriately. Finally, Fig. 5 (B3) exemplary shows the results for tests of type $B3$. In this test, we alternated three types of requests (desktop PC, PDA and digital watch requests). As can be seen, the algorithm propose appropriate feedback structures after just 1 session of each type. This is due to the fact, that for the three request types, the consumer's judgment behavior differed much in terms of the number and types of aspects to be judged. Hence, though α and β are not yet adjusted, the correct feedback structure can be identified.

8. CONCLUSION

In this paper, we demonstrated how detailed consumer feedback, that is meaningful and appropriate in the context of a service interaction, can be elicited and how users can be supported in that process. Our main contribution is an algorithm that suggests service aspects that might be judged by a consumer. Our evaluation results show, that the proposed procedure effectively adjusts to a user's ability and willingness to provide judgments.

9. REFERENCES

[1] G. Adomavicius. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Transactions on Information Systems*, 23(1):103, 2005.

[2] R. M. Dawes. The robust beauty of improper linear models in decision making. *American Psychologist*, 34(7):571–582, 1979.

[3] D. Fensel, H. Lausen, A. Polleres, J. de Bruijn, M. Stollberg, D. Roman, and J. Domingue. *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Springer, 2007.

[4] Y. K. Gediminas Adomavicius. New recommendation techniques for multi-criteria rating systems. *IEEE Intelligent Systems*, 22(3), 2007.

[5] P. Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

[6] F. Klan and B. König-Ries. Enabling trust-aware semantic web service selection - a flexible and personalized approach. Jenaer Schriften zur Mathematik und Informatik, Math/Inf/02/10, Friedrich-Schiller-University Jena, August 2010.

[7] U. Küster, B. König-Ries, M. Klein, and M. Stern. Diane - a matchmaking-centered framework for automated service discovery, composition, binding and invocation. In *Proceedings of the 16th International World Wide Web Conference (WWW2007)*, Banff, Alberta, Canada, May 2007.

[8] U. S. Manikrao and T. V. Prabhakar. Dynamic selection of web services with recommendation system. In *Intl. Conf. on Next Generation Web Services Practices*, pages 117–121, Washington, DC, 2005. IEEE Computer Society.

[9] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web: Methods and Strategies of Web Personalization*, volume 4321 of *Lecture Notes in Computer Science*, pages 291–324. Springer, Berlin, Heidelberg, 2007.

[10] P. Slovic. *Limitations of the Mind of Man: Implications for decision making in the nuclear age*. Los Alamos Scientific Laboratory, 1972.

[11] H. C. Wang, C. S. Lee, and T. H. Ho. Combining subjective and objective qos factors for personalized web service selection. *Expert Syst. Appl.*, 32(2):571–584, 2007.

Groups Identification and Individual Recommendations in Group Recommendation Algorithms [*] [†]

Ludovico Boratto
Dipartimento di Matematica e Informatica, Università di Cagliari
Via Ospedale 72
09124 Cagliari, Italy
ludovico.boratto@unica.it

Salvatore Carta
Dipartimento di Matematica e Informatica, Università di Cagliari
Via Ospedale 72
09124 Cagliari, Italy
salvatore@unica.it

Michele Satta
Dipartimento di Matematica e Informatica, Università di Cagliari
Via Ospedale 72
09124 Cagliari, Italy
michele_satta@hotmail.com

ABSTRACT

Recommender systems usually deal with preferences previously expressed by users, in order to predict new ratings and recommend items. To support recommendation in social activities, group recommender systems were developed. Group recommender systems usually consider predefined/a priori known groups and just a few existing approaches are able to automatically identify groups.

When groups are not already formed, another key aspect of group recommendation is related to groups identification. In this paper a novel algorithm able to identify groups of users and produce recommendations for each group is presented. The algorithm uses individual recommendations and a classic clustering algorithm to identify and model groups. Experimental results show how this approach substantially improves the quality of group recommendations with respect to the state-of-the-art.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering; H.4 [**Information Systems Applications**]: Miscellaneous; M.4 [**Knowledge Modeling**]: Miscellaneous

General Terms

Algorithms, Experimentation, Performance

Keywords

Group Recommendation, Collaborative Filtering, Clustering

[*]This work is partially funded by Regione Sardegna under project CGM (Coarse Grain Recommendation) through Pacchetto Integrato di Agevolazione (PIA) 2008 "Industria Artigianato e Servizi".

1. INTRODUCTION

With the development of Web 2.0, the use of the web has become increasingly widespread and users have had the chance to express opinions about shared content updated daily. This generates an incredible amount of data that can't be handled directly by the users. So finding relevant information over the Internet nowadays is becoming more and more difficult [18].

Recommender systems have been developed to deal with information overload and produce personalized content for the users by exploiting context-awareness in a domain. This is done by computing a set of previously expressed preferences, in order to recommend items that are likely of interest to a user. Collaborative Filtering (CF) [11, 15, 19] is by far the most successful recommendation technique. The main idea of CF systems is to use the opinions of a community, in order to provide item recommendations.

There are context and domains where classic recommendation cannot be used, because the recommendation process involves more than a person and preferences have to be combined in order to produce a single recommendation that satisfies everyone (e.g., people traveling together or going to a restaurant/museum together). Therefore, in order to support recommendations in social activities, algorithms able to provide group recommendations were developed. Group recommendations are provided according to the way a group is modeled. Group modeling is the combination of the preferences expressed by single users into a common group preference.

A special type of group recommendation is needed when technological constraints limit the bandwidth available for the recommendation. This is for example the case of Satellite Systems, in which the number of channels is limited and a personalized TV schedule cannot be produced.

Another useful application scenario in which limitations are imposed in the recommendation process is the printing of recommendation flyers that contain suggested items. Even if a company has all the data to produce a flyer with individual recommendations for each customer, the process of printing

a different flyer for everyone would be technically too hard to achieve and costs would be too high. A possible solution would be to print n different flyers that can be affordable in terms of costs and that can satisfy users by recommending interesting items to the recipients of the same flyer.

In both the scenarios described the first result that the algorithm has to compute is a proper identification of groups, in order to produce a recommendation that maximizes users satisfaction. This preliminary phase of the group recommendation process is not performed by the great part of algorithms in literature, because they consider only how to model already existing groups.

In this paper a novel approach for group recommendation with automatic identification of groups is proposed.

To enhance the readability of the paper and the properties of the proposed approach, a baseline version of the algorithm is preliminarily presented (*BaseGRA*, Baseline Group Recommendation Algorithm). *BaseGRA* uses a classic clustering algorithm to identify groups, by exploiting past preferences expressed by each user of the system. To model the group, *BaseGRA* combines the preferences of each user with the ratings predicted using a CF algorithm for the unrated items.

Since the number of items evaluated by a user in a system is usually much lower than the number of the items that can be evaluated, we considered the fact that the clustering step may be affected by the well-known problem of *sparsity* of the available data.

The algorithm presented in this paper, named *Improved-GRA* (Improved Group Recommendation Algorithm), has been developed to overcome this potential problem and improve the quality of clustering. This is done by using the predictions of the missing ratings to complete the matrix of the preferences already expressed by users. The algorithm predicts individual recommendations, combines them with the preferences explicitly expressed by users, and uses both of them as input for a classic clustering algorithm. As highlighted by the experiments, this leads to an identification of groups of users with similar preferences with a high quality of the predicted results. Individual recommendations and explicitly provided preferences are also used to model the groups.

The proposed approach is the first that combines clustering of the users with an aggregation of individual recommendations. In fact none of the existing recommender systems that automatically identify groups merges individual recommendations and the approaches that merge individual recommendations deal with groups that have a predefined structure.

Another scientific contribution of the approach relies in the algorithm used to automatically identify groups, which mixes recommendation and clustering algorithms, leading to a substantial improvement of the quality of the group recommendations with respect to the state-of-the-art.

Moreover the paper presents an analysis of two more fundamental aspects of this kind of group recommendation: homogeneity of group size and homogeneity of recommendations quality.

Considering the size of groups, it is evident that it should be sufficiently homogeneous. In simple words, if the recommendation process involves 70000 users and 10 available channels, it would not be acceptable to have a group with 61000 users and 9 groups with 1000 users. In fact it would be a waste of bandwidth to produce recommendations for small groups and, at the same time, it would be hard for a system to produce recommendations that gather the preferences of a large group.

Considering the quality of the predicted results, it should not vary too much between the groups. In other terms, the system should try to keep a sufficient quality of the predictions for every group. Providing inadequate recommendations to any group should always be avoided.

The rest of the paper is organized in the following way: section 2 presents related work, considering both group recommender systems able to automatically identify groups and group recommender systems that build individual recommendations; section 3 contains a detailed description of the baseline group recommendation algorithm, BaseGRA; section 4 will do the same for the improved algorithm Improved-GRA; section 5 describes the experiments we conducted to evaluate the proposed algorithm and outlines main results; section 6 contains comments, conclusions and future developments.

2. RELATED WORK

As mentioned in the Introduction, group recommender systems were developed to support the recommendation process in activities that involve more than a person.

In [13] and [5] the state-of-the-art in group recommendation is presented. The existing systems were developed for different domains like web/news pages, tourist attractions, music tracks, television programs and movies. A classification of those approaches can be made from two perspectives:

- the type of group considered;
- the way group recommendations are built.

Considering the first classification of the existing systems, which is based on the type of groups considered, we can identify four different types of groups, described below.

- **Established group**: a number of persons who explicitly choose to be part of a group, because of shared, long-term *interests*;
- **Occasional group**: a number of persons who do something occasionally together, like visiting a museum. Its members have a common *aim* in a particular moment;
- **Random group**: a number of persons who share an environment in a particular moment, without explicit interests that link them;
- **Automatically identified group**: groups that are automatically detected considering the preferences of the users and/or the resources available.

The second classification of the existing approaches can be done considering the way group recommendations are built. There are two ways to build group recommendations, described in the list below.

- Merge of individual recommendations into a group recommendation.
- Merge of the individual preferences to build a group profile and predict specific recommendations for the group.

The approach described in this paper automatically identifies groups and merges individual recommendations. The existing approaches for those two categories of group recommender systems will now be described and differences with our approach will be highlighted.

As a general consideration, please note that none of the approaches that automatically identify groups merges individual recommendations.

2.1 Approaches that automatically identify groups

The approach proposed in [8] aims to automatically discover Communities of Interest (CoI) (i.e., a group of individuals who share and exchange ideas about a given interest) and produce recommendations for them.

CoI are identified considering the preferences expressed by users in personal ontology-based profiles. Each profile measures the interest of a user in concepts of the ontology. Users interest is exploited in order to cluster the concepts.

User profiles are then split into subsets of interests, to link the preferences of each user with a specific cluster of concepts. Hence it is possible to define relations among users at different levels, obtaining a multi-layered interest network that allows to find multiple CoI. Recommendations are built using a content-based CF approach.

The difference with our approach is that preferences of users are not expressed through an ontology. Moreover, our recommendation technique is based on a CF user-based approach.

The system proposed in [6] generates group recommendations and automatically detects intrinsic communities of users whose preferences are similar. Communities of users with similar preferences are identified using a Modularity-based Community Detection algorithm [4] and group recommendations are predicted for each community. See 5.2 for a more detailed description of the approach.

This approach, although it achieves exactly the same purposes, differs from the one presented in this paper both in the way group predictions are built and in the way groups are identified. The approach was chosen for comparison with the algorithm presented in this paper because of the mentioned similarities in several aspects.

2.2 Approaches that merge individual recommendations

PolyLens [17] is a system built to produce recommendations for groups of users who want to see a movie.

To produce recommendations for each user of the group a CF algorithm is used. In order to model the group, a "least misery" (LM) strategy is used: the rating used to recommended a movie to a group is the lowest predicted rating for that movie, to ensure that every member is satisfied.

In contrast with the LM strategy used by PolyLens, in our approach group preferences are built combining individual recommendations in a single value that averages the preferences of the single users.

We considered the use of a group modeling technique based on the average of users ratings instead of using a LM strategy because it seems more suited for an approach where large groups are considered. A LM strategy is useful for small groups and in fact Polylens handles groups with two or three users. Even if groups are composed by people with homogeneous preferences, using a LM strategy a low rating expressed by a user for a movie would be enough to have a low rating for that movie for the whole group. With large groups such an approach would probably lead to extremely low ratings for almost all the movies.

INTRIGUE (INteractive TouRist Information GUidE) [2, 3] is a system that recommends sightseeing destinations using the preferences of the group members. The approach merges individual recommendations and, in order to build group recommendations, some subgroups are considered more influential (e.g., disabled people).

In our approach we don't consider a specific domain of application and every individual recommendation is weighted equally, so that group recommendations reflect all the users preferences.

The approach presented in [1] computes group recommendations by combining individual recommendations built for every user and considering a *consensus function*, which combines *relevance* of the items for a user and *disagreement* between members.

Since our approach automatically builds groups of users with similar preferences, we don't expect disagreement to be a characterizing feature when computing group recommendations. Therefore this aspect was not considered in our approach.

The system proposed in [9, 10] presents a group recommendation approach based on Bayesian Networks (BN). To represent users and their preferences a BN is built. The authors assume that the composition of the groups is a priori known and model the group as a new node in the network that has the group members as parents. A collaborative recommender system is used to predict the votes of the group members. A posteriori probabilities are calculated to combine the predicted votes and build the group recommendation.

The main difference with our approach is that, in order to combine preferences and build group recommendations, we

don't rely on a Bayesian Network and a posteriori probabilities.

3. BASELINE GROUP RECOMMENDATION ALGORITHM (BASEGRA)

The baseline version of our algorithm identifies groups of similar users considering the preferences expressed by each user and models each group using individual recommendations built for each user of a group.

3.1 Overview of BaseGRA

The algorithm works in two steps:

1. Using a *Ratings Matrix* that contains the preferences of each user, groups of similar users are detected through the k-means clustering algorithm [14].
2. Once the groups have been detected, a group preference is produced by aggregating the preferences of the individual users.

3.2 Groups Identification

The input of the algorithm is a *Ratings Matrix* M that associates a set of *users* to a set of *items* through a *rating*. A rating indicates the level of satisfaction of a user for a considered item. So each value m_{ui} of the Ratings Matrix is:

$$m_{ui} = \begin{cases} r_{ui} & \text{if user } u \text{ expressed a preference for item } i \\ \emptyset & \text{if user } u \text{ didn't express a preference for item } i \end{cases}$$

A rating r_{ui} is always such that $r_{min} \leq r_{ui} \leq r_{max}$ and $r_{ui} > 0$. In other words, a rating value is always inside a fixed range and its value is always positive.

The Ratings Matrix is used as input for the k-means clustering algorithm [14]. Since the algorithm's input are the preferences expressed by each user, the output is a partition in groups of users with similar preferences.

3.3 Groups Modeling

The objective of group modeling is to calculate, for each item, a group rating which will be evaluated in order to decide which items should be recommended to the group. In order to model a group, the preferences of each user that belongs to the group have to be combined.

An average is a single value that is meant to typify a list of values. The most common method to calculate such a value is the arithmetic mean, which also seems an effective way to put together the preferences of each user in a group, in order to reach our objective.

Combining just the preferences expressed by the users would lead to a poor modeling of the group, since each user usually gives an explicit preference to a small set of item. This is especially true when modeling small groups. In fact group preferences have to be extracted considering a small set of preferences expressed by a small set of users.

In order to improve the efficiency of group modeling, our algorithm completes the Ratings Matrix, adding individual recommendations predicted for each user. The result is a *Predicted Ratings Matrix* PR that associates each user u with an item i either through an explicitly expressed rating r_{ui} or through a predicted rating p_{ui}.

A predicted rating p_{ui} is calculated using a classic User-Based Nearest Neighbor CF Algorithm, proposed in [20]. The algorithm predicts a rating p_{ui} for each item i that was not evaluated by a user u, considering the rating r_{ni} of each similar user n for the item i. A user n similar to u is called a *neighbor* of u. Equation 1 gives the formula used to predict the ratings:

$$p_{ui} = \overline{r}_u + \frac{\sum_{n \subset neighbors(u)} sim(u,n) \cdot (r_{ni} - \overline{r}_n)}{\sum_{n \subset neighbors(u)} sim(u,n)} \quad (1)$$

Values $\overline{r}_u$ and $\overline{r}_n$ represent, respectively, the mean of the ratings expressed by user u and user n. Similarity $sim()$ between two users is calculated using the Pearson correlation, a coefficient that compares the ratings of all the items rated by both the target user and the neighbor (*corated* items). Pearson correlation between a user u and a neighbor n is given in Equation 2. $CR_{u,n}$ is the set of corated items between u and n.

$$sim(u,n) = \frac{\sum_{i \subset CR_{u,n}} (r_{ui} - \overline{r}_u)(r_{ni} - \overline{r}_n)}{\sqrt{\sum_{i \subset CR_{u,n}} (r_{ui} - \overline{r}_u)^2} \sqrt{\sum_{i \subset CR_{u,n}} (r_{ni} - \overline{r}_n)^2}} \quad (2)$$

4. IMPROVED GROUP RECOMMENDATION ALGORITHM (IMPROVEDGRA)

BaseGRA identifies groups of similar users using a Ratings Matrix, i.e., a matrix that contains all the preferences expressed by users for the evaluated items.

However, the number of items rated by users is much lower than the number of available items. This leads to the sparsity problem that is common in clustering.

ImprovedGRA was conceived to improve the quality of the clustering step of BaseGRA. ImprovedGRA identifies groups giving as input to the k-means algorithm not the original Ratings Matrix M, that contains the ratings already expressed by users, but the complete *Predicted Ratings Matrix* PR previously presented, where the predicted values of the unrated items for each user are added.

In order to do so, the individual recommendations are predicted by ImprovedGRA at the beginning of the computation. Using more values as input for the clustering, the algorithm should be able to identify better groups, i.e., groups composed by users having more correlated preferences. This should lead to a higher overall quality of the group recommendations.

In conclusion, ImprovedGRA performs the same steps performed by BaseGRA but computes individual recommen-

dations before clustering the users. This allows to cluster the users using more preferences and identify better groups. The preferences expressed by users and the individual recommendations are also used to model the group.

5. EXPERIMENTS

In this section we first describe the strategy and aims which drove our experiments.

Then a state-of-the-art group recommender system that automatically identifies groups, chosen for comparison with the proposed approach, is described.

Experiments setup and metrics used are then described and, at the end of the section, results are shown and commented.

5.1 Experimental Methodology

In order to evaluate the quality of the system, three aspects were considered: quality of the predicted ratings, distribution of the quality between the groups and homogeneity of the groups size. The details of each experiment will be described next.

5.1.1 Quality of the predicted ratings evaluation

The main objective of a recommender system is to produce high quality predictions. The algorithm presented in this paper produces group recommendations adapting to the bandwidth available for the recommendation process.

In order to evaluate the quality of the predicted ratings for different bandwidths, i.e., for different numbers of channels that can be dedicated to the recommendation, we built three different partitions of the users in groups. A partition is a set of n groups in which users are subdivided. Of course, if groups are homogeneous, the larger is n, the smaller are the groups and the system can predict better ratings, because the preferences of a small amount of users have to be combined.

In order to properly evaluate the performances of the proposed algorithms, we compared them with the results obtained considering a single group with all the users (predictions are calculated considering all the preferences expressed for an item), and the results obtained using no partition of the users (i.e., quality of the individual recommendations is calculated).

To measure the quality of the predicted ratings, we used the Root Mean Squared Error (RMSE). This metric was chosen because it is the most common in literature.

In order to analyze the quality of the predictions produced by each algorithm for different partitions, we produced a plot that shows the trend of RMSE for each partition in n groups.

5.1.2 Distribution of quality between the groups evaluation

A second important aspect that has to be evaluated is how the quality of the predicted results is distributed between the groups of a partition.

In fact a group recommender system should be able to distribute the quality of the predicted results in a sufficiently equal way, in order to satisfy the recommendation demand for all the users of the system.

To analyze how RMSE is distributed between the groups produced by ImprovedGRA, a table that contains the mean value of RMSE for each partition and how many groups have a RMSE value close/far to the mean is presented.

To compare the different algorithms, we measured the standard deviation of the RMSE values obtained for every group of a partition.

5.1.3 Distribution of size between the groups evaluation

The last aspect we evaluated is how homogeneous are the groups in terms of size. Indeed, it is not acceptable to have too large or too small groups. At the same time the clustering step cannot create an homogeneity which is not intrinsically existent in users. To evaluate this trade-off we measured the standard deviation of the size of the groups present in a partition.

5.2 Benchmark algorithm: ModularityBasedGRA

The technique selected for comparison with ImprovedGRA, is the one proposed in [6]. From now on, the algorithm will be called ModularityBasedGRA, because of the approach used to identify groups (based on the Modularity function).

ModularityBasedGRA is an algorithm that generates group recommendations and automatically detects intrinsic communities of users whose preferences are similar. The input is a Ratings Matrix that associates a set of *users* to a set of *items* through a *rating*. Based on the ratings expressed by each user, the algorithm evaluates the level of similarity between users and generates a network that contains the similarities.

A modularity-based Community Detection algorithm proposed in [4] is run on the network in order to find partitions of users in communities. For each community, ratings for all the items are predicted using an item-based CF algorithm.

Since the Community Detection algorithm is able to produce a dendrogram, i.e. a tree that contains hierarchical partitions of the users in communities of increasing granularity, the quality of the recommendations can be evaluated for the different partitions.

To achieve the objectives previously outlined, i.e., detect the communities and produce group recommendations for them, ModularityBasedGRA computes four steps, described below.

Users similarity evaluation In order to create communities of users, the algorithm takes as input a *Ratings Matrix* and evaluates through a standard metric (cosine similarity) how similar the preferences of two users are. The result is a weighted network where nodes rep-

resent users and each weighted edge represents the similarity value of the users it connects. A post-processing technique is then introduced to remove noise from the network and reduce its complexity.

Communities detection In order to identify intrinsic communities of users, a Community Detection algorithm proposed by [4] is applied to the users similarity network and partitions of different granularities are generated.

Ratings prediction for the items rated by the group A group's ratings are evaluated by calculating, for each item, the mean of the ratings expressed by the users of the group. In order to predict meaningful ratings, the algorithm calculates a rating only if an item was evaluated by a minimum percentage of users in the group. With this step it is not possible to predict a rating for each item, so another step was created to predict the remaining ratings.

Ratings prediction for the remaining items For some of the items, ratings could not be calculated by the previous step. In order to estimate such ratings, similarity between items is evaluated, and the rating of an item is predicted with a CF item-based algorithm that considers the items most similar to it.

The choice to compare ImprovedGRA with this approach is motivated by the fact that both approaches produce group recommendations and automatically identify groups of users. Moreover, both can be evaluated for different partitions of users in groups. This allows a direct comparison between the two approaches.

Let us also note that even if the aim of the two algorithms is the same, the two techniques work in completely different ways: ImprovedGRA clusters users with a classic algorithm (k-means) after building individual recommendations and then models the groups preferences, while ModularityBasedGRA clusters users with a Community Detection algorithm and then builds group recommendations.

5.3 Experiments Setup

The experimentation was made using the MovieLens-1M dataset, which is composed of 1 million ratings, expressed by 6040 users for 3900 movies. In order to evaluate the quality of the ratings predicted by each of the algorithms, around 20% of the ratings was extracted as a test set and the rest of the dataset was used as a training set for the algorithm.

Each group recommendation algorithm was run with the training set and, for each partition of the users in groups, ratings were predicted.

The obtained values were used to conduct the experiments previously described.

5.4 Evaluation metrics

This section will introduce the two metrics used to evaluate different characteristics of our algorithm, the Root Mean Squared Error (RMSE) and the Standard deviation. Both metrics compare the obtained results with a comparison value, in order to evaluate the quality of the system.

5.4.1 *Root Mean Squared Error (RMSE)*

The quality of the predicted ratings was measured through the Root Mean Squared Error (RMSE). The metric compares the test set with the predicted ratings: each rating r_{ui} expressed by a user u for an item i is compared with the rating p_{gi} predicted for the item i for the group in which user u is. The formula is shown below:

$$RMSE = \sqrt{\frac{\sum_{i=0}^{n}(r_{ui} - p_{gi})^2}{n}}$$

where n is the number of ratings available in the test set.

5.4.2 *Standard deviation*

The homogeneity of the groups size and the distribution of RMSE between the groups was measured with the standard deviation (considering respectively the size of the groups and the RMSE values of the groups).

The metric evaluates how much variation there is from the "average" value. A low standard deviation indicates that the size of the groups/the RMSE obtained for the groups tend to be close to the mean, while high values of standard deviation indicate that the obtained values are scattered over a large range of values.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

5.5 Experimental results

The first experiment, presented in 5.1.1, aims to evaluate the quality of the predicted values for a partition of the users in groups. Figure 1 shows the trend of the RMSE values for the different partitions of the users in groups.

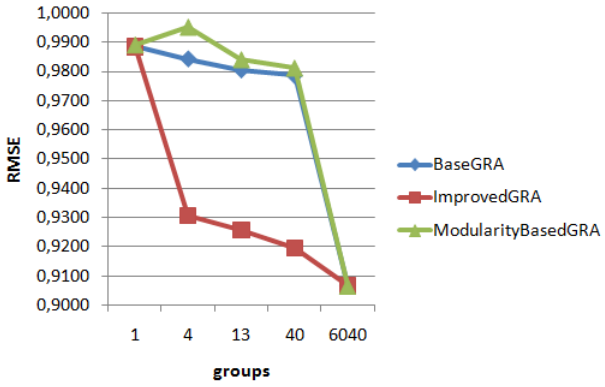


Figure 1: RMSE values for each partition

For all the algorithms, we can notice that as the number of groups grows, the quality of the recommendations improves, since groups get smaller and the algorithms can predict more precise ratings. We can see that the values of RMSE notably decrease when the algorithms start grouping the users (i.e., there is a big difference of RMSE between 1 and 4 groups). The RMSE values continue to decrease for the other partitions, but the improvement in quality is lower.

Comparing the algorithms, we can see that BaseGRA and ImprovedGRA outperform the benchmark algorithm ModularityBasedGRA. Moreover, the performances of ImprovedGRA are much better than the performances of BaseGRA: this proves that enhancing the Ratings Matrix with individual recommendations leads to great improvements in the quality of the predicted results.

The second experiment, presented in 5.1.2, was conducted to evaluate how the quality of the predicted values is distributed between the groups. To do so we measured the standard deviation of RMSE of the groups in each partition.

Partition	Number of groups with RMSE r			
4 groups	$r = 0,85$	$r = 0,89$	$r = 0,95$	$r = 1,04$
$\bar{x} = 0,93$	1	1	1	1
13 groups	$r < 0,87$	$0,87 \leq r \leq 1,00$		$r > 1,00$
$\bar{x} = 0,93$	3	7		3
40 groups	$r < 0,90$	$0,90 \leq r \leq 1,00$		$r > 1,00$
$\bar{x} = 0,96$	15	15		10

Table 1: Distribution of RMSE between the groups

Table 1 shows, for each partition, the mean of the RMSE obtained for every group with ImprovedGRA and how the RMSE is distributed between the groups. As we can see, the majority of the groups in each partition has a RMSE value sufficiently close to the mean. This means that RMSE is distributed quite equally between the groups and our approach is able to satisfy the recommendation demand for all the users.

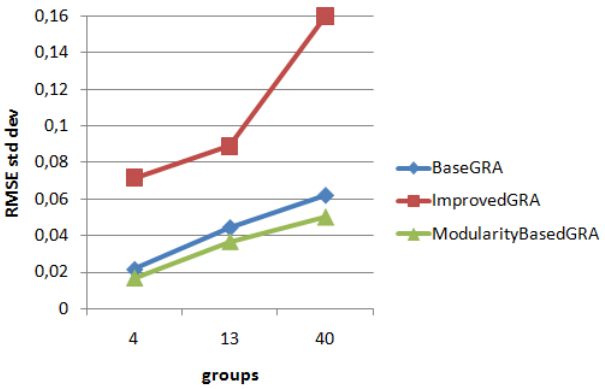


Figure 2: Standard deviation of RMSE of the groups

Figure 2 compares the standard deviation of RMSE of the groups for the different approaches. ImprovedGRA values are slightly higher if compared to the other approaches. However, it is important to remember that in this case there is a trade-off between an equal distribution in terms of RMSE and the similarity between the users in a group. In fact the groups have to be intrinsic in order to improve the quality of the predicted results. So it seems reasonable to loose a bit of homogeneity in distribution of the quality in order to improve the overall quality of the results predicted by the system. This is the case of ImprovedGRA in which the RMSE is distributed less equally between the groups but the quality of the predictions compared with the other approaches is much higher.

The third experiment, presented in 5.1.3 was conducted to evaluate how the size of the groups is distributed in each partition (i.e., how homogeneous are the groups in terms of size). To do so we measured the standard deviation of the size of all the groups in each partition.

Partition	Number of groups with size s			
4 groups	$s = 633$	$s = 1334$	$s = 1807$	$s = 2266$
$\bar{x} = 1510$	1	1	1	1
13 groups	$s < 300$	$300 \leq s \leq 540$		$s > 540$
$\bar{x} = 464,62$	3	7		3
40 groups	$s < 80$	$80 \leq s \leq 250$		$s > 250$
$\bar{x} = 151$	9	26		5

Table 2: Distribution of size of the groups

Table 2 shows, for each partition, the mean of the size obtained for every group with ImprovedGRA and how the size is distributed between the groups. As the table shows, most of the groups have size values close to the mean. This means that the size is distributed in a sufficiently equal way between the groups and our algorithm is able to produce recommendations properly, i.e., without handling the preferences of too small/large groups.

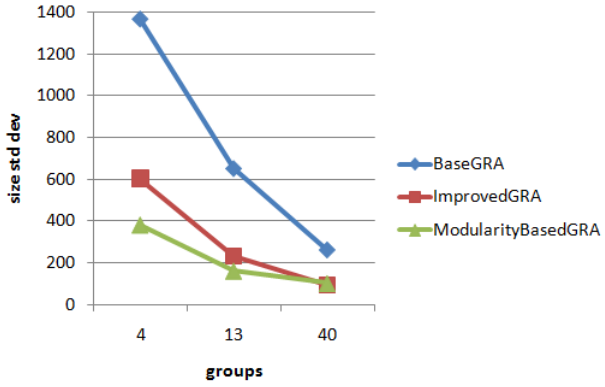


Figure 3: Standard deviation of size of the groups

Figure 3 compares the standard deviation of the size of the groups for the different approaches. It is important to notice how the enhancement of the Ratings Matrix made for ImprovedGRA leads to more homogeneous partitions in groups compared with BaseGRA.

The values obtained by ImprovedGRA are slightly higher than ModularityBasedGRA but also in this case there is a trade of between homogeneity of the groups size and similarity between the users. In fact it is important to find partitions of intrinsic groups with similar preferences that can lead to a high quality of the predicted results. So, a little loss in homogeneity of the size leads to great improvements in the quality of the results.

6. CONCLUSIONS AND FUTURE WORK

In this paper we presented an algorithm that combines user clustering with individual recommendations in order to identify and model groups of users with similar preferences and improve the quality of group recommendations in systems that automatically identify groups. In fact, BaseGRA and ImprovedGRA outperform the benchmark algorithm ModularityBasedGRA.

Moreover, we can notice that ImprovedGRA, using an enhanced Ratings Matrix to identify and model the groups, is able to produce sufficiently homogeneous groups in terms of size and distribution of RMSE. Therefore, all the three important objectives that should be achieved by a group recommender systems are reached by the proposed algorithm ImproveGRA.

Future developments of the algorithm have been planned for different steps performed by the algorithm. In [16] several strategies for group modeling were presented. We are currently studying how different strategies affect the quality of group recommendation with groups that are automatically identified.

Recently [7, 12] highlighted how different metrics to evaluate the quality of recommendation lead to completely different results. As a future work we plan to evaluate our systems with such metrics, in order to catch different aspects of our system.

7. REFERENCES

[1] S. Amer-Yahia, S. B. Roy, A. Chawla, G. Das, and C. Yu. Group recommendation: Semantics and efficiency. *PVLDB*, 2(1):754–765, 2009.

[2] L. Ardissono, A. Goy, G. Petrone, and M. Segnan. A multi-agent infrastructure for developing personalized web-based systems. *ACM Trans. Internet Technol.*, 5(1):47–69, 2005.

[3] L. Ardissono, A. Goy, G. Petrone, M. Segnan, and P. Torasso. Intrigue: Personalized recommendation of tourist attractions for desktop and handset devices. *Applied Artificial Intelligence*, 17(8):687–714, 2003.

[4] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *J. Stat. Mech.*, 2008(10):P10008+, October 2008.

[5] L. Boratto and S. Carta. State-of-the-art in group recommendation and new approaches for automatic identification of groups. In G. A. Alessandro Soro, Eloisa Vargiu and G. Paddeu, editors, *Information Retrieval and Mining in Distributed Environments*. Springer Verlag. In press, 2010.

[6] L. Boratto, S. Carta, A. Chessa, M. Agelli, and M. L. Clemente. Group recommendation with automatic identification of users communities. In *Web Intelligence/IAT Workshops*, pages 547–550. IEEE, 2009.

[7] E. Campochiaro, R. Casatta, P. Cremonesi, and R. Turrin. Do metrics make recommender algorithms? *International Conference on Advanced Information Networking and Applications Workshops*, 0:648–653, 2009.

[8] I. Cantador, P. Castells, and E. P. Superior. Extracting multilayered semantic communities of interest from ontology-based user profiles: Application to group modelling and hybrid recommendations. In *Computers in Human Behavior, special issue on Advances of Knowledge Management and the Semantic*. Elsevier. In press, 2010.

[9] L. M. de Campos, J. M. Fernández-Luna, J. F. Huete, and M. A. Rueda-Morales. Group recommending: A methodological approach based on bayesian networks. In *ICDE Workshops*, pages 835–844. IEEE Computer Society, 2007.

[10] L. M. de Campos, J. M. Fernández-Luna, J. F. Huete, and M. A. Rueda-Morales. Managing uncertainty in group recommending processes. *User Model. User-Adapt. Interact.*, 19(3):207–242, 2009.

[11] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communication of the ACM*, 35(12):61–70, 1992.

[12] A. Gunawardana and G. Shani. A survey of accuracy evaluation metrics of recommendation tasks. *J. Mach. Learn. Res.*, 10:2935–2962, 2009.

[13] A. Jameson and B. Smyth. Recommendation to groups. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web: Methods and Strategies of Web Personalization*. Springer, 2007.

[14] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[15] T. W. Malone, K. R. Grant, F. A. Turbak, S. A. Brobst, and M. D. Cohen. Intelligent information-sharing systems. *Communication of the ACM*, 30(5):390–402, 1987.

[16] 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.

[17] M. O'Connor, D. Cosley, J. A. Konstan, and J. Riedl. Polylens: a recommender system for groups of users. In *ECSCW'01: Proceedings of the seventh conference on European Conference on Computer Supported Cooperative Work*, pages 199–218, Norwell, MA, USA, 2001. Kluwer Academic Publishers.

[18] S. Ram. Intelligent agents and the world wide web: Fact or fiction? *Journal of Database Management*, 12(1):46–49, 2001.

[19] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, 1994. ACM.

[20] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. In *The Adaptive Web: Methods and Strategies of Web Personalization*, volume 4321 of *Lecture Notes in Computer Science*, chapter 9, pages 291–324. Springer, 2007.

A Generalized Probabilistic Framework and its Variants for Training Top-k Recommender Systems

Harald Steck
Bell Labs, Alcatel-Lucent
Murray Hill, NJ
Harald.Steck@alcatel-lucent.com

Yu Xin*
CSAIL MIT
Cambridge, MA
YuXin@mit.edu

ABSTRACT

Accounting for missing ratings in available training data was recently shown [3, 17] to lead to large improvements in the top-k hit rate of recommender systems, compared to state-of-the-art approaches optimizing the popular root-mean-square-error (RMSE) on the observed ratings. In this paper, we take a Bayesian approach, which lends itself naturally to incorporating background knowledge concerning the missing-data mechanism. The resulting log posterior distribution is very similar to the objective function in [17]. We conduct elaborate experiments with real-world data, testing several variants of our approach under different hypothetical scenarios concerning the missing ratings. In the second part of this paper, we provide a generalized probabilistic framework for dealing with possibly *multiple observed rating values* for a user-item pair. Several practical applications are subsumed by this generalization, including aggregate recommendations (e.g., recommending artists based on ratings concerning their songs) as well as collaborative filtering of sequential data (e.g., recommendations based on TV consumption over time). We present promising preliminary experimental results on IP-TV data.

Categories and Subject Descriptors

H.2.8 **[Database Management]:** Database Applications–*Data Mining*

General Terms

Algorithms

Keywords

Recommender Systems

*This work was done while an intern at Bell Labs, Alcatel-Lucent.

1. INTRODUCTION

The idea of recommender systems is to automatically suggest items to each user that s/he may find appealing. The quality of recommender systems can be assessed with respect to various criteria, including accuracy, diversity, surprise / serendipity, and explainability of recommendations.

This paper is concerned with accuracy. The *root mean squared error (RMSE)* has become the most popular accuracy measure in the literature of recommender systems–for training and testing. Its computational efficiency is one of its main advantages. Impressive progress has been made in predicting rating values with small RMSE, and it is impossible to name all approaches, e.g., [4, 6, 7, 11, 13]). There is, however, also some work on optimizing the ranking of items, e.g., measured in terms of normalized Discounted Cumulative Gain (nDCG) [18]. Despite their differences, they have in common that they were trained and tested on *observed ratings only*. Obviously, these measures cannot immediately be evaluated if some items have unobserved ratings.

In this paper, we consider the *top-k hit rate*–based on *all* (unrated) items–as the natural accuracy measure for recommender systems, as only a few out of *all* unrated items can be recommended to a user in practice (see Section 3 for exact definition of top-k hit rate). While this measure is computationally tractable for *testing* the predictions of recommender systems, unfortunately it is computationally very costly for *training* recommender systems. For training, we thus resort to appropriate surrogate objective functions that are computationally efficient.

In recent work [3, 17], it was shown that the top-k hit rate can be significantly improved on large real-world data by accounting for the fact that the observed ratings provide a skewed picture of the (unknown) distribution concerning *all* (unrated) items and users.

Motivated by the results of [17], as the first contribution of this paper, in Section 2 we present a probabilistic approach that allows us to naturally include background knowledge concerning the (unknown) distribution of all items and users into our training objective function, the posterior probability of the model.

In our second contribution, we conduct elaborate experiments on the Netflix Prize data [1] and test several models under different hypothetical scenarios concerning the missing ratings in Section 4. These different scenarios serve as a sensitivity analysis, as the ground truth of the missing data-mechanism is unknown due to lack of data. These experiments are based on our *popularity-stratified* recall measure, which we define in Section 3.

As the third contribution of this paper, we generalize this probabilistic approach as to account for possibly *multiple observed rating values* for a user-item pair in Section 5. This general framework subsumes several applications in addition to the one outlined in Section 2. Two of which are outlined in Section 5: while the training objective function for a recommender system concerning TV programs seems motivated in an ad-hoc manner in [5], we show that it can be understood and improved naturally in a Bayesian framework; apart from that, we also provide a Bayesian approach for making aggregate recommendations, e.g., recommending an artist or concert to a user based on the ratings given to individual songs.

2. MODEL TRAINING

In this section, we outline a probabilistic framework that allows us to incorporate background knowledge when training recommender systems on available data. The use of background knowledge in addition to the available training data can significantly improve the accuracy of recommender systems on performance measures like top-k hit rate, recall, precision, area under the ROC curve, etc. This was demonstrated for implicit feedback data in [5, 10], and for explicit feedback data in [3, 17]. Like in [17], we use background knowledge that missing rating values tend to reflect negative feedback, as experimentally observed in [9, 8]; i.e., negative feedback tends to be missing from the available data with a larger probability than positive feedback does.

The Bayesian approach lends itself naturally to this task. We consider the rating matrix R as a *matrix of random variables*: each element $R_{i,u}$ concerning item $i = 1, ..., i_0$ and user $u = 1, ..., u_0$ is a random variable with normal distribution, where i_0 denotes the number of items and u_0 is the number of users.

2.1 Model

We take a collaborative filtering approach, and use a low-rank matrix-factorization model, which has proven successful in many publications. Like the rating matrix, we consider our model as a *matrix of random variables*, M. Each random variable $M_{i,u}$ corresponds to the rating of item i assigned by user u. In matrix notation, it reads

$$M = r^{\text{offset}} + PQ^\top \tag{1}$$

where $r^{\text{offset}} \in \mathbb{R}$ is an offset value, and P, Q are low-rank matrices of random variables with dimensions $i_0 \times d_0$ and $u_0 \times d_0$, respectively, where rank $d_0 \ll i_0, u_0$. We use upper case symbols to denote random variables (with a Gaussian distribution), and lower case symbols to denote values.

2.2 Prior over Matrix Elements

In our Bayesian approach, we first define the usual prior over model parameters, concerning each entry of the low rank matrices P and Q (see also [12]):

$$\begin{aligned} p(M|\sigma_P^2, \sigma_Q^2) &= \left\{ \prod_i \prod_d \mathcal{N}(P_{i,d}|0, \sigma_{P,i}^2) \right\} \\ &\quad \cdot \left\{ \prod_u \prod_d \mathcal{N}(Q_{u,d}|0, \sigma_{Q,u}^2) \right\} \end{aligned} \tag{2}$$

The vectors of variances $\sigma_Q^2 = (\sigma_{Q,u}^2)_{u=1,...,u_0}$ and $\sigma_P^2 = (\sigma_{P,i}^2)_{i=1,...,i_0}$ for all users $u = 1, ..., u_0$ and items $i = 1, ..., i_0$ are free parameters of the zero-mean normal prior distribution, denoted by $\mathcal{N}$. There are several ways of defining the standard deviations in Eq. 2, eventually resulting in different kinds of regularization. The obvious choice is to assume that $\sigma_{P,i} = \sigma_{Q,u} = 1/\sqrt{2\lambda'} \quad \forall\, i, u$, with $\lambda' \in \mathbb{R}$. This results in the regularization term

$$\log p(M|\sigma_P^2, \sigma_Q^2) = -\lambda' \left(||P||_2^2 + ||Q||_2^2 \right) + c_1,$$

where $||\cdot||_2$ denotes the Frobenius norm of a matrix, and c_1 is an irrelevant constant when training our model.

When optimizing root mean square error on observed data (like in the Netflix Prize competition [1]), however, numerous experimental works reported significant improvements by using a different regularization. This is obtained by choosing the standard deviations σ_P and σ_Q as follows: $\sigma_{P,i} = 1/\sqrt{2\lambda' \cdot u_0(i)}$, $\sigma_{Q,u} = 1/\sqrt{2\lambda' \cdot i_0(u)}$, where $i_0(u)$ denotes the number of items rated by user u, and $u_0(i)$ is the number of users who rated item i. This results in the popular regularization term

$$\begin{aligned} &\log p(M|\sigma_P^2, \sigma_Q^2) \\ &= -\lambda' \left(\sum_i u_0(i) \sum_d P_{i,d}^2 + \sum_u i_0(u) \sum_d Q_{u,d}^2 \right) + c_2 \\ &= -\lambda' \left(\sum_{\text{observed } (i,u)} \left(\sum_d P_{i,d}^2 + Q_{u,d}^2 \right) \right) + c_2, \end{aligned} \tag{3}$$

where c_2 denotes again an irrelevant constant when training our model. Note that this choice increasingly regularizes the model parameters related to the items and users with a larger number of observed ratings. This may seem counter-intuitive at first glance. A theoretical explanation for this empirical finding was recently provided in [14].

2.3 Informative Background Knowledge

We now incorporate the following background knowledge into our sequential Bayesian approach: absent rating values tend to be lower than the observed ratings on average (see [17]). We insert this knowledge into our approach by means of a virtual data point for *each* pair (i, u): a virtual rating value $r_{i,u}^{\text{prior}}$ with small confidence (i.e., large variance $\sigma_{\text{prior},i,u}^2$). Then the likelihood of our model in light of these virtual data points reads (assuming i.i.d. data):

$$p(r^{\text{prior}}|M, \sigma_{\text{prior}}^2) = \prod_{\text{all } i} \prod_{\text{all } u} p(R_{i,u} = r_{i,u}^{\text{prior}}|M_{i,u}, \sigma_{\text{prior},i,u}^2), \tag{4}$$

where r^{prior} denotes the matrix of virtual data points $r_{i,u}^{\text{prior}}$, and σ_{prior}^2 the matrix with elements $\sigma_{\text{prior},i,u}^2$. We assume that the probabilities in this likelihood are determined by normal distributions with mean $M_{i,u}$ and variance $\sigma_{\text{prior},i,u}^2$. The log likelihood then reads

$$\log p(r^{\text{prior}}|M, \sigma_{\text{prior}}^2) = -\sum_{\text{all } i} \sum_{\text{all } u} w_{i,u}^{\text{prior}} \left(r_{i,u}^{\text{prior}} - M_{i,u} \right)^2 + c_3 \tag{5}$$

where we defined the weights of the virtual data points as $w_{i,u}^{\text{prior}} = 1/(2\sigma_{\text{prior},i,u}^2)$; c_3 is again an irrelevant constant when training our model.

With Bayes rule, we obtain the posterior distribution of

the model in light of these virtual data points:

$$p(M|r^{\text{prior}}, w^{\text{prior}}) = \frac{p(r^{\text{prior}}, w^{\text{prior}}|M)p(M)}{p(r^{\text{prior}}, w^{\text{prior}})}. \quad (6)$$

This equation combines our prior concerning the elements in the matrices P and Q (for regularization) with our background knowledge on the expected rating values. This serves as our prior when observing the actual rating values in the training data.

2.4 Training Data

Now we use the rating values actually observed in the training data. The likelihood of the model in light of observed rating values $r^{\text{obs}}_{i,u}$ reads (assuming i.i.d. data):

$$p(r^{\text{obs}}|M, \sigma^2_{\text{obs}}) = \prod_{\text{observed } (i,u)} p(R_{i,u} = r^{\text{obs}}_{i,u}|M_{i,u}, \sigma^2_{\text{obs},i,u})$$

Again assuming a normal distribution, the log likelihood reads:

$$\log p(r^{\text{obs}}|M, \sigma^2_{\text{obs}}) = -\sum_{\text{observed } (i,u)} w^{\text{obs}}_{i,u}(r^{\text{obs}}_{i,u} - M_{i,u})^2 + c_4 \quad (7)$$

where we defined the weights of the observed rating values as $w^{\text{obs}}_{i,u} = 1/(2\sigma^2_{\text{obs},i,u})$; c_4 is again an irrelevant constant when training our model.

2.5 Posterior

The posterior after seeing the observed ratings is again obtained by Bayes rule (we omit the weights $w^{\text{obs}}, w^{\text{prior}}$ for brevity of notation here):

$$\begin{aligned} p(M|r^{\text{obs}}, r^{\text{prior}}) &= \frac{p(r^{\text{obs}}|M, r^{\text{prior}})p(M|r^{\text{prior}})}{p(r^{\text{obs}})} \\ &\propto p(r^{\text{obs}}|M, r^{\text{prior}})p(r^{\text{prior}}|M)p(M) \end{aligned} \quad (8)$$

where we assumed in the denominator that the observed ratings are independent of the chosen prior ratings, i.e., $p(r^{\text{obs}}|r^{\text{prior}}) = p(r^{\text{obs}})$. Substituting Eqs. 3, 5, 6 and 7 into Eq. 8, we obtain the following log posterior:

$$\begin{aligned} &\log p(M|r^{\text{obs}}, r^{\text{prior}}, w^{\text{obs}}, w^{\text{prior}}, \lambda) = \\ &-\sum_{\text{obs.}(i,u)} w^{\text{obs}}_{i,u} \left\{ (r^{\text{obs}}_{i,u} - M_{i,u})^2 + \lambda \sum_d \left[P^2_{i,d} + Q^2_{u,d}\right] \right\} \\ &-\sum_{\text{all}(i,u)} w^{\text{prior}}_{i,u} \left\{ (r^{\text{prior}}_{i,u} - M_{i,u})^2 + \lambda \sum_d \left[P^2_{i,d} + Q^2_{u,d}\right] \right\} \\ &+c_5 \end{aligned} \quad (9)$$

We found that using a prior that involves also the regularization term of P and Q in the third line in Eq. 9 leads to a slight improvements in our experimental results. The weights $w^{\text{obs}}_{i,u}$ and $w^{\text{prior}}_{i,u}$ as well as λ' are absorbed in λ; this is a slight but straight-forward generalization of the prior in Section 2.2; c_5 is again an irrelevant constant for training.

Eq. 9 serves as our training objective function. For simplicity, we choose the same value for all virtual rating values r^{prior}. For computational efficiency, we choose our model offset to equal the prior rating values: $r^{\text{offset}} = r^{\text{prior}}$. Its main effect is that this retains the sparsity of the observed rating matrix. Apart from that, it also leads to the simplification: $(r^{\text{prior}}_{i,u} - M_{i,u})^2 = ((PQ^\top)_{i,u})^2$. For simplicity, we set $w^{\text{obs}}_{i,u} = 1$ for all observed pairs (i, u), and also choose all prior weights to be identical: $w^{\text{prior}} = w^{\text{prior}}_{i,u}$ for all (i, u). In summary, the three tuning parameters in Eq. 9 are w^{prior}, r^{prior} and λ, which can be chosen as to optimize the performance measure on cross-validation data.

2.6 MAP Estimate of Model

For computational efficiency, our training aims to find the *maximum-a-posteriori (MAP)* parameter estimate of our model, i.e., the MAP estimates $\hat{P}$ and $\hat{Q}$ of the matrices P and Q. We use the *alternating least squares* approach. The idea is that one matrix can be optimized exactly while the other one is assumed fixed. A local maximum of the log posterior can be found by alternating between the matrices $\hat{P}$ and $\hat{Q}$. While local optima exist [16], we did not find this to cause major computational problems in our experiments. The update equation for each row i of $\hat{P}$ is (for fixed $\hat{Q}$):

$$\begin{aligned} \hat{P}_{i,\cdot} = (\bar{r}_{i,\cdot} - r^{\text{prior}})(\tilde{W}^{(i)} + w^{\text{prior}} I)\hat{Q} \cdot \\ \left[\hat{Q}^\top(\tilde{W}^{(i)} + w^{\text{prior}} I)\hat{Q} + \lambda(\text{tr}(\tilde{W}^{(i)}) + w^{\text{prior}} u_0)I\right]^{-1} \end{aligned} \quad (10)$$

where $\bar{r}_{i,u} = (r^{\text{obs}}_{i,u} w^{\text{obs}}_{i,u} + r^{\text{prior}} w^{\text{prior}}_{i,u})/(w^{\text{obs}}_{i,u} + w^{\text{prior}}_{i,u})$ denotes the average rating; we defined $w^{\text{obs}}_{i,u} = 0$ if rating at (i, u) is missing; note that $\bar{r}_{i,\cdot} - r^{\text{prior}} = 0$ if rating is missing for (i, u); $\tilde{W}^{(i)}$ is a diagonal matrix containing the i^{th} column of the weight matrix w^{obs}; the trace is $\text{tr}(\tilde{W}^{(i)}) = \sum_{u \in S_i} w^{\text{obs}}_{i,u}$, where S_i is the set of users who rated item i; I denotes the identity matrix; and u_0 is the number of users. This equation can be re-written for efficient computations, e.g., see [17]. The update equation for $\hat{Q}$ is analogous.

3. MODEL TESTING

A key challenge in testing recommender systems is that the observed ratings in the available data typically provide a skewed picture of the (unknown) true distribution concerning *all* ratings [9, 8]. This may be caused by the fact that users are free to choose what items to rate, and they tend to not rate items that would otherwise receive a low rating. If the ratings are missing not at random (MNAR), it is not guaranteed that correct or meaningful results are obtained from testing a recommender system on the *observed ratings only*. The latter is, however, common practice in the literature or recommender systems, using measures like root mean square error or nDCG [4, 6, 7, 11, 13, 18].

The *top-k hit-rate / recall* of *relevant* items is a particularly useful performance measure for assessing the accuracy of recommender systems [17]. An item i is *relevant* to user u if s/he finds this item interesting or appealing [17]. For instance, in the Netflix data [1] we consider items i with a 5-star rating, $r^{\text{obs}}_{i,u} = 5$, as relevant to user u.

Recall can be calculated for a user by ranking the items according to their scores predicted by the recommender system, and determining the fraction of relevant items that are among the top-k items, i.e., the k items with the highest scores. The value of $k \in \mathbb{N}$ has to be chosen, e.g., as the number of items that can be recommended to a user. Only small values of k are important in practice. The goal is to maximize recall for the chosen value of k.

Recall has two interesting properties in this context [17]: it is proportional to precision on fixed data and fixed k when comparing different recommender systems with each other. In other words, the recommender system with the larger

recall also has the larger precision. More interestingly, however, recall can be calculated from the available MNAR data and provides an *unbiased* estimate for the recall concerning the (unknown) complete data (which comprises all rating values of all users) under the following assumption: the *relevant* ratings are *missing at random*, while an arbitrary missing-data mechanism may apply to all other rating values (as long as they are missing with a larger probability than the relevant ones) [17]. This assumption is much milder than the one underlying the popular approach of ignoring missing ratings, i.e., assuming that *all* ratings are missing at random.

The expected performance on the (unknown) *complete data* is important because it is directly related to *user experience*: the recommender system has to pick a few items from among all items the user has not rated yet (and which may hence be new to the user); one can expect the distribution on all unrated items to be well-approximated by the distribution on all (rated and unrated) items under the (mild) assumption that only a small fraction of the relevant ratings has been observed.

Given that ground truth (i.e., the complete data) is typically not available (at low cost), the validity of the assumption in [17] cannot be verified in practice. For this reason, we carry out a sensitivity analysis in the following. We relax this assumption even further and determine its effect on the recall test-results for different models. Note that the assumption in [17] allows for an arbitrary missing-data mechanism concerning all ratings, except for the relevant ratings; only the latter are assumed to be missing at random. For this reason, the following is concerned with the *relevant ratings only.*

We consider the case that the probability of observing a relevant rating depends on the popularity of items. We define the popularity of an item by the number $N^+_{\text{complete},i}$ of *relevant* ratings it obtained in the (unknown) complete data. Let $N^+_{\text{obs},i}$ be the number of relevant ratings observed in the available data; then the probability of observing a relevant rating regarding item i is

$$p_{\text{obs}}(i) = \frac{N^+_{\text{obs},i}}{N^+_{\text{complete},i}}. \tag{11}$$

Assuming that there are no additional (possibly hidden) factors underlying the missing data mechanism concerning the relevant ratings, we obviously obtain an unbiased estimate for recall on the (unknown) complete data by calculating the *popularity-stratified recall* (for user u),

$$\text{recall}_u(k) = \frac{\sum_{i \in S^{+,k}_u} s_i}{\sum_{i \in S^+_u} s_i} \tag{12}$$

on the available MNAR data; S^+_u denotes the set of *relevant* items of user u; $S^{+,k}_u$ is the subset of relevant items ranked in the top k items based on the predictions of the recommender system; the popularity-stratification weight for each item i is

$$s_i = \frac{1}{p_{\text{obs}}(i)}.$$

If we choose the stratification weights $s_i = 1$ for all items i, we obtain the usual recall measure. Given that complete data is unavailable (at low cost), p_{obs} in Eq. 11 cannot be calculated in practice. For this reason, we now examine different choices for p_{obs} and their effects on recall in Eq. 12.

In the first scenario, p_{obs} may take only two values: it is small for unpopular items, and large for popular items. If the ratio of these two values approaches infinity, this results in $s_i \to 0$ for popular items. Removing the relevant ratings of the most popular items from the test set is indeed common practice, e.g., in [3]. In the second scenario, we consider the case where p_{obs} is a smooth function of the items' popularities. We assume the polynomial relationship

$$p_{\text{obs}}(i) \propto \left(N^+_{\text{complete},i}\right)^\gamma \tag{13}$$

with $\gamma \in \mathbb{R}$. This is consistent with the power-law behavior of the observed relevant ratings (see Figure 1 a) in the sense that also the (unknown) complete data then follows a power-low distribution concerning the relevant ratings. This results in the stratification weights

$$s_i \propto 1/(N^+_{\text{obs},i})^{\gamma/(\gamma+1)}.$$

Note that the unknown proportionality factor cancels out in Eq. 12, so that it provides an unbiased estimate of the recall concerning the complete data for the correct polynomial degree γ (and assuming that there are no additional factors underlying the missing data mechanism). If $\gamma = 0$, the relevant ratings are missing at random; if $\gamma = 1$, the probability of observing relevant ratings increases *linearly* with item popularity ($N^+_{\text{complete},i}$). The extreme case when $\gamma \to \infty$ has several interesting properties: first, one observes in the available data only relevant ratings of the item with the largest popularity in the complete data, which does not agree with empirical evidence. Second, as $\gamma/(\gamma+1) \to 1$, the stratification weights s_i are inversely proportional to the number of observed relevant ratings $N^+_{\text{obs},i}$; this means that every item has the same weight in the recall-measure in Eq. 12, independent of its number of observed relevant ratings. This means that, once an item obtains its first relevant rating, it is weighted the same as all other items that may have obtained thousands of relevant ratings. This obviously entails low robustness against statistical noise as well as against manipulation and attacks. As this is the limiting case of γ, we nevertheless provide experimental results for this extreme scenario in Section 4. The (unknown) realistic case can be expected to be less extreme.

If the test set is a random subset of the observed data, then $N^+_{\text{obs},i}$ can be determined from the test set. Given that the training set is typically much larger than the test set, it might be more robust to determine $N^+_{\text{obs},i}$ based on all the available data, i.e., the training and test set combined. We use the latter in our experiments reported in the next section, but the former choice of $N^+_{\text{obs},i}$ leads to very similar results.

Finally, we define $\text{recall}(k) = \sum_u w^u \text{recall}_u(k)$ as the average recall over all users, with normalized weights, $\sum_u w^u = 1$, like in [17]. In our experiments, we choose $w^u \propto \sum_{i \in S^+_u} s_i$, as a generalization of the definition in [7, 17].

Obviously, stratification like in Eq. 12 carries over analogously to other measures, like ATOP [17] or the area under the ROC curve.

4. EXPERIMENTS

This section summarizes our results on the Netflix Prize data [1]. These data contain 17,770 movies and almost half a million users. About 100 million ratings are available. Ratings are observed for about 1% of all possible movie-

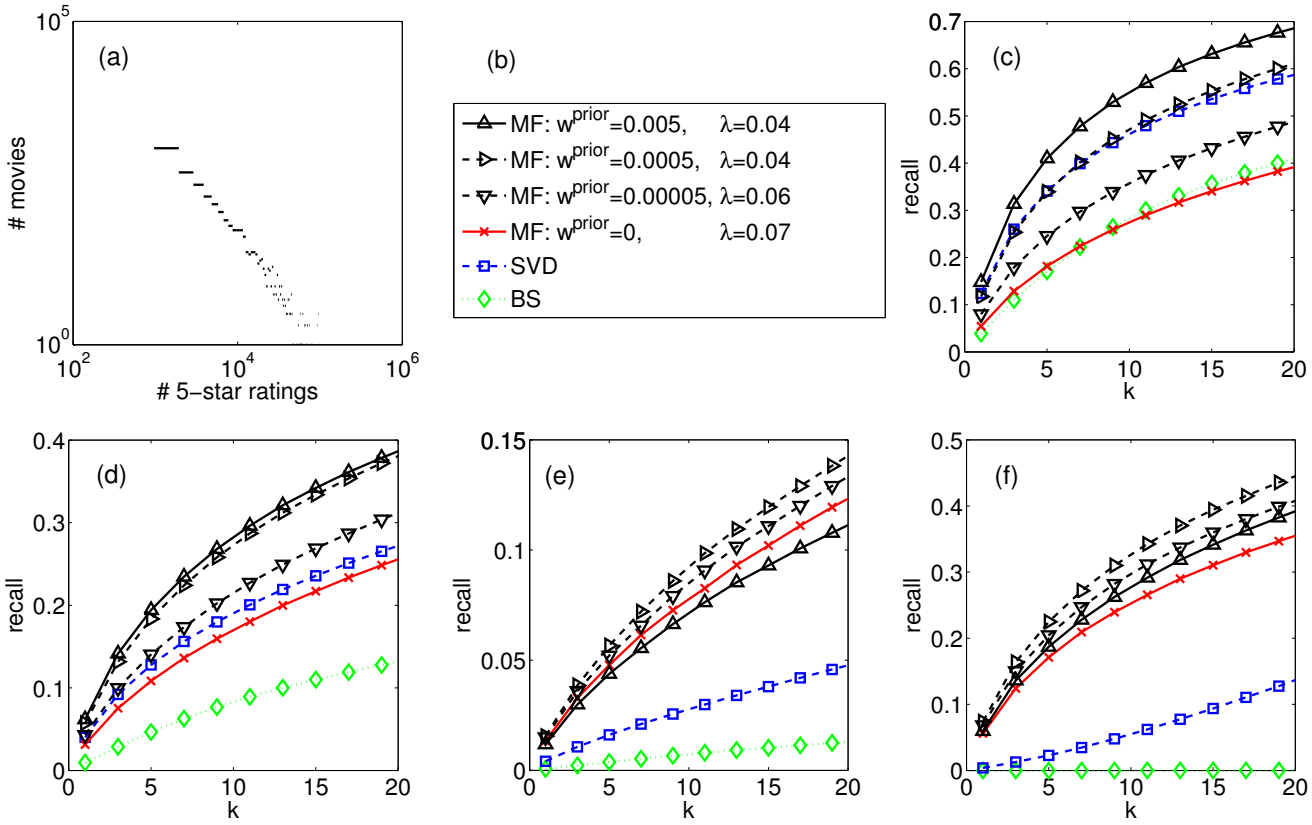


Figure 1: Netflix data [1]: (a) the number of relevant (i.e., 5-star) ratings per movie in the training data shows a close to power-law distribution (i.e., straight line in the log-log plot); (b) legend of models (see text for details); (c)-(f) show recall on probe set for different hypothetical missing-data mechanisms concerning the relvant (i.e., 5-star) ratings (while an arbitrary missing-data mechanism is allowed for the other ratings): (c) relevant ratings are missing at random ($\gamma = 0$ in Eq. 13); (d) relevant ratings observed with probability increasing linearly with item popularity ($\gamma = 1$ in Eq. 13); (e) unrealistic extreme case where $\gamma \to \infty$ in Eq. 13; (f) relevant ratings of the 10% most popular items removed. As a result, for all these missing-data mechanisms, recall test-results are improved by using an appropriate *small* prior weight $w^{\text{prior}} > 0$ during training, compared to the popular approach of ignoring the missing-data mechanism ($w^{\text{prior}} = 0$) during training.

user-pairs. The ratings take integer values from 1 (worst) to 5 (best). The provided data are already split into a training and a probe set. We removed the probe set from the provided data as to obtain our training set.

We consider 5-star ratings as *relevant* to a user (as defined above), and use the popularity-stratified recall, as outlined in Section 3, as our performance measures on the Netflix probe set. For all experiments, we chose rank $d = 50$ of our low-rank matrix factorization (MF) model (Eq. 1). In the following, we compared our MF model, trained with different prior weights $w^{\text{prior}} > 0$, against the popular MF approach that ignores the missing-data mechanism (i.e., $w^{\text{prior}} = 0$ in our notation). The latter achieved a root mean square error on the observed ratings in the probe set of 0.922. Additionally, we compared to singular value decomposition (SVD), where we used the *svds* function of Matlab, which implicitly imputes a zero value (with unit weight) for all missing ratings; and to the bestseller list (BS), which ranks the items according to the number of ratings in the training set. The values of tuning parameters in our training objective function (log posterior) in Eq. 9 are summarized in Figure 1 (b). Like in [17], we chose the prior rating value $r^{\text{prior}} = 2$ in Eq. 9.

Figures 1 (c)-(f) shows the performance of these models under different test scenarios, concerning our popularity-stratified recall measure for the practically important range of small k values. For computational efficiency, we computed recall by randomly sampling, for a user, 1,000 unrated items for each relevant rating in the test set, like in [3]. The only difference to the test procedure used in [7, 17] is that we sample from unrated items only, rather than from all items. This is more realistic. It also results in slightly higher recall values compared to the procedure used in [7, 17].

When comparing the different graphs, it is obvious that the performance of all the models depends on the (unknown) missing-data mechanism concerning the relevant ratings. In particular, when p_{obs} of relevant ratings is assumed to increase more rapidly with growing popularity ($N^+_{\text{complete},i}$),

the expected recall on the (unknown) complete data *decreases* for all models, cf. Figures 1 (c)→(d)→(e), and (c)→(f).

As p_{obs} of relevant ratings increases more rapidly with item popularity (compare Figures 1 (c)→(d)→(e)), the difference in recall among the various MF models decreases. Training with smaller but positive weights $w^{\text{prior}} > 0$ results in the best recall on the test set, even in the unrealistic extreme limit in Figure 1 (e). This suggests that, compared to the popular approach of ignoring the missing-data mechanism when training MF models, recall can be improved by using a small prior weight $w^{\text{prior}} > 0$; its value is upper bounded by the value that optimizes the (usual) recall measure on the test set, i.e., under the assumption that the relevant ratings are missing at random, like in [17].

The bestseller list (BS) and SVD perform surprisingly well if relevant ratings are missing at random, see Figure 1 (c), while the popular MF model with $w^{\text{prior}} = 0$ has low recall in comparison. This was also found in [17, 3]. BS and SVD perform rather poorly, however, if p_{obs} increases rapidly with item popularity, as shown in the extreme scenarios in Figure 1 (e) and (f). This suggests that not only BS, but also SVD tend to recommend items that are popular in the available training data. Their recommendations may hence result in a relatively low degree of serendipity or surprise, relative to our MF models trained with a small positive prior weight.

5. GENERALIZED APPROACH

This section outlines a generalization of the Bayesian approach given above. In our probabilistic approach, we consider the rating matrix R as a *matrix of random variables*. As each entry $R_{i,u}$ is a random variable (rather than a value), this naturally allows for possibly *multiple values* concerning each pair (i, u) in the data. This has several advantages over a *matrix of values*, which has typically been considered in the literature of recommender systems. After developing our generalized probabilistic framework, we outline three special cases / applications in Section 5.2.

Let the given data set be $D = \{r_{i,u,j}\}_{i,u,j}$, where $i = 1, ..., i_0$ is the index concerning items, $u = 1, ..., u_0$ is the index regarding users, and $j = 1, ...$ is the index over possibly multiple observed ratings for the same pair (i, u). The likelihood of the model in light of i.i.d. data reads

$$p(D|M) = \prod_{i,u,j} p(R_{iu} = r_{i,u,j}|M). \tag{14}$$

Assuming again a normal distribution of the ratings (with standard deviations $\sigma_{i,u,j}$, or equivalently weights $w_{i,u,j} = 1/(2\sigma^2_{i,u,j})$), the log likelihood of the model is

$$\begin{aligned} \log p(D|M) &= -\sum_{i,u}\sum_{j} w_{i,u,j}(r_{i,u,j} - M_{i,u})^2 + c_5 \\ &= -\sum_{i,u}\sum_{v} w_{i,u,v}(v - M_{i,u})^2 + c_5 \end{aligned} \tag{15}$$

where the second line is obtained by switching–for each pair (i, u)–from index j (over multiple ratings in the data) to the actual rating values v; the cumulative weight is $w_{i,u,v} = \sum_j w_{i,u,j} I_{r_{i,u,j}=v}$, where indicator function $I_{r_{i,u,j}=v} = 1$ if $r_{i,u,j} = v$ and 0 otherwise.

Combining this likelihood with the same kind of prior over the model parameters as in Eq. 2, we obtain the log posterior of our model:

$$\begin{aligned} \log p(M|D) = &-\sum_{i,u}\sum_{v} w_{i,u,v}(v - M_{i,u})^2 \\ &-\sum_{i}\frac{1}{2\sigma^2_{P,i}}\sum_{d} P^2_{i,d} - \sum_{u}\frac{1}{2\sigma^2_{Q,i}}\sum_{d} Q^2_{u,d} + c_6 \end{aligned} \tag{16}$$

The standard deviations $\sigma_{Q,i}$ and $\sigma_{P,i}$ may be chosen as to achieve the desired variant of regularization, as discussed in Section 2.2.

5.1 MAP Estimate of Model

For computational efficiency, we focus on optimizing the log posterior in Eq. 16. The *maximum-a-posteriori (MAP)* parameter estimate of our model, i.e., the MAP estimates $\hat{P}$ and $\hat{Q}$ of the matrices P and Q, can be determined by *alternating least squares*, which alternately optimizes one matrix while the other one is assumed fixed. Using the usual necessary condition for the optimum of Eq. 16, we equate its partial derivative to zero, and obtain the following update equation for each row i of $\hat{P}$ (for fixed $\hat{Q}$):

$$\hat{P}_{i,\cdot} = (\bar{v}_{i,\cdot} - r^{\text{offset}})\tilde{W}^{(i)}\hat{Q} \cdot \left[\hat{Q}^\top \tilde{W}^{(i)}\hat{Q} + \frac{1}{2\sigma^2_{P,i}} I\right]^{-1}, \tag{17}$$

where I denotes the identity matrix, and $\tilde{W}^{(i)}$ is a diagonal matrix containing the i^{th} row of the aggregate weight matrix with elements

$$w_{i,u} = \sum_{v} w_{i,u,v},$$

and $\bar{v}_{i,u}$ is the average rating value

$$\bar{v}_{i,u} = \left(\sum_{v} v w_{i,u,v}\right) / w_{i,u}.$$

Analogously, the update equation for each row u of $\hat{Q}$ is:

$$\hat{Q}_{u,\cdot} = (\bar{v}_{\cdot,u} - r^{\text{offset}})\tilde{W}^{(u)}\hat{P} \left[\hat{P}^\top \tilde{W}^{(u)}\hat{P} + \frac{1}{2\sigma^2_{Q,u}} I\right]^{-1}, \tag{18}$$

where $\tilde{W_u}$ is the diagonal matrix containing the u^{th} column of the aggregate weight matrix.

This derivation shows that optimizing Eq. 16 is equivalent to optimizing

$$\begin{aligned} \log p(M|D) = &-\sum_{i,u} w_{i,u}(\bar{v}_{i,u} - M_{i,u})^2 \\ &-\sum_{i}\frac{1}{2\sigma^2_{P,i}}\sum_{d} P^2_{i,d} - \sum_{u}\frac{1}{2\sigma^2_{Q,i}}\sum_{d} Q^2_{u,d} + c_6 \end{aligned} \tag{19}$$

where multiple rating values for an item-user pair are replaced by their mean value $\bar{v}_{i,u}$ and their aggregate weight $w_{i,u}$.

5.2 Applications

This generalized probabilistic approach subsumes several applications as special cases. In addition to the use of virtual ratings in the prior, as outlined in Section 2, we present two additional applications in the following.

5.2.1 Aggregate Recommendations

The universe of all items available for recommendation may have a structure that goes beyond a flat list. Items can

often be arranged in a hierarchical manner. For instance, songs may be grouped by artist, album, genre, etc. Possibly, there are several layers of hierarchy. Now let us consider the problem that data are available where users have rated individual songs, but the task is to recommend an artist to a user. This problem arises in several situations. For instance, the recommender system may want to suggest a concert to the user, based on the data on individual songs. Another scenario is the release of a new song by an artist: this cold start problem may be overcome by recommending the new song to users who like the artist.

The ratings matrix concerning songs and users can be used to construct a rating matrix regarding artists and users by aggregating the songs of each artist. As a user may have rated several songs of an artist, we now have possibly multiple rating values for an entry in the artist-user matrix. This is exactly the problem solved by our general approach in Section 5. Our framework also shows that, for each user, the rating of each artist can be determined as the weighted average of the ratings of his/her songs, and the weight is the sum of the weights of the songs, as one may have intuitively expected.

5.2.2 Recommendation of TV Shows

IP-TV is much more interactive than traditional TV. Concerning recommender systems, it allows one to collect information on the users' TV consumption. This can be used to learn preferences of users to TV programs, as to make accurate recommendations of TV shows. Unlike the previous applications described in this paper, we now use implicit feedback data (time spent watching a TV show) in place of the ratings. Our approach carries over immediately. This problem can be cast in our general probabilistic framework as follows: we divide the length of each show into $n_{\max}$ time intervals (of equal length), where $n_{\max}$ is the same large integer for all shows. We consider each time interval associated with a random experiment; a show hence comprises $n_{\max}$ repetition of the experiment. The random variable $R_{i,u}$ takes the value 1 if user u watched show i for a time-interval, and 0 otherwise. So, if user u watches $n_{i,u} \in \{1, ..., n_{\max}\}$ out of $n_{\max}$ intervals of show i, then we have $n_{i,u}$ observations of value 1 for the random variable $R_{i,u}$, and $n_{\max} - n_{i,u}$ observations of value 0; as shown in Section 5, these multiple observations can be substituted equivalently in our least-squares objective function in Eq. 16 by the aggregate weight $n_{\max}$ and the averaged value of the observations: $\bar{r}_{i,u} = n_{i,u}/n_{\max}$, i.e., the fraction of the show watched. In addition, if a show comprises several (e.g., weekly) episodes, then we assume that the above applies to each episode; the total weight of the show is then $t_{i,u} n_{\max}$, where $t_{i,u} \in \mathbb{N}$ is the number of episodes of show i watched by user u. Then the averaged observed value is $\bar{r}_{i,u} = \sum_{j=1}^{t_{i,u}} \bar{r}_{i,u,j}/t_{i,u}$, where $\bar{r}_{i,u,j}$ is the fraction of episode j watched (analogous to above). This results in the log likelihood (with an irrelevant constant c_7 in our optimization problem):

$$\log p(D|M) \quad = \quad -n_{\max} \sum_{(i,u)\in S} t_{i,u}(\bar{r}_{i,u} - M_{i,u})^2 + c_7,$$

where the set S contains all pairs (i, u) with shows i that have been *watched at least partially* by user u. Concerning all shows, we additionally incorporate background knowledge that users tend to not like shows with some small confidence / weight. This weight is small, as it can also be interpreted as the variance of our prior, which is large as there are many reasons for not watching a show. Analogously to Section 2, we use virtual observations of value 0, with weight w^{prior}. This results in the log likelihood in light of the virtual data points D^{prior}:

$$\log p(D^{\text{prior}}|M) \quad = \quad -n_{\max} w^{\text{prior}} \sum_{\text{all } (i,u)} (0 - M_{i,u})^2 + c_8$$

Combining these two likelihoods, together with the prior over the model parameters in Eq. 2 (first variant), we obtain the log posterior

$$\begin{aligned} &\log p(M|D, D^{\text{prior}}) \propto \\ &- \sum_{(i,u)\in S} t_{i,u}(\bar{r}_{i,u} - M_{i,u})^2 - w^{\text{prior}} \sum_{\text{all } (i,u)} (0 - M_{i,u})^2 \\ &-\lambda \sum_d \left[\sum_i P_{i,d}^2 + \sum_u Q_{u,d}^2 \right] + c_9, \end{aligned} \tag{20}$$

where we replaced the standard deviations in the prior by $\lambda \in \mathbb{R}$; the proportionality is due to omitting $n_{\max}$, which is an irrelevant constant when optimizing the posterior; c_9 is an irrelevant additive constant in our optimization problem.

In [5], the following objective function was experimentally found to result in the best recommendation accuracy from among several variants (re-written, but equivalent to Eq. 3 in [5]):

$$\begin{aligned} &\sum_{(i,u)\in S} (1 + \alpha n_{i,u}^{\text{t}})(1 - M_{i,u})^2 + \sum_{(i,u)\notin S} (0 - M_{i,u})^2 \\ &+\lambda \sum_d \left[\sum_i P_{i,d}^2 + \sum_u Q_{u,d}^2 \right], \end{aligned} \tag{21}$$

where $n_{i,u}^{\text{t}} = \sum_j n_{i,u,j}$ is the total time spent by user u watching show i (including all episodes j); S denotes again the set of pairs (i, u) where show i is at least partially watched by user u; α takes essentially the role of w^{prior}.

Interestingly, their objective function is similar to ours. The main difference, however, is in the least squares term, where we fit our model to the fraction $\bar{r}_{i,u}$ of the show watched, while the indicator value 1 is used in [5]. We attribute to this difference the fact that our model performs slightly better in our preliminary experiments on our (proprietary) IP-TV data [2], see below. Besides the experimental improvement, our theoretical framework also provides a clear understanding of the assumptions underlying our approach, while the approach in [5] appears to be found experimentally in a somewhat ad-hoc manner.

Preliminary Experiments: We used a (proprietary) IP-TV data set [2] concerning TV consumption of N =25,777 different shows by 14,731 users (living in 6,423 households) in the UK over a 6 month period in 2008/2009 (see also [15] for a more detailed description). In our collaborative filtering approach, we used only implicit feedback data concerning TV consumption (user ID, program ID, the length of the program and the time the user spent on this program), and ignored the available explicit user profiles and content information for simplicity. We randomly split these data into a training and a disjoint test set, with 10 shows per user assigned to the test set. In the test set, we considered shows interesting or relevant to users if they watched at least

model	$k' = 1\%$	$k' = 2\%$	$k' = 3\%$	$k' = 4\%$	$k' = 5\%$
ours	0.671	0.763	0.819	0.856	0.882
[5]	0.624	0.723	0.785	0.828	0.857
Nbr	0.547	0.642	0.704	0.760	0.804

Table 1: Recall(k') test results on IP-TV data.

80 % of them. We used these shows to evaluate the recommender systems w.r.t. the performance measures *recall* (as defined in Section 3 with $\gamma = 0$). Table 1 summarizes our preliminary results in terms of recall(k'), where $k' = k/N$ is normalized regarding the number N of available shows. Again, we used rank $d = 50$ for the matrix factorization models. We find that our new approach (with $\lambda = 0.005$) and the approach in [5] (with $\lambda = 0.02$) give similar results, compared to the neighborhood model (Nbr), $s_{ij} = \frac{r_i r'_j}{\|r_i\| \|r_j\|}$ and $\hat{r}_{iu} = \sum s_{ij} r_{uj}$, which was also used in [5] for comparison. Concerning recall, small values of k' are particularly important in practice, as only a small number of shows can be recommended to a user; in this regime, our new approach seems to perform better than the approach of [5]. We are currently running more refined experiments to confirm this finding.

6. CONCLUSIONS

This paper provides three contributions. First, we outlined a Bayesian framework that naturally allowed us to insert background knowledge concerning the missing-data mechanism underlying the observed rating data. The obtained log posterior probability of the model is very similar to the training objective function outlined in [17].

In our second contribution, we conducted experiments where we considered several hypothetical missing-data mechanisms underlying the observed real-world data. Given that the true missing-data mechanism is unknown in the absence of ground truth data, this sensitivity analysis provided valuable information. Our key insight based on these experiments is that the top-k hit-rate or recall can be improved considerably by training recommender systems with an appropriately chosen *small* positive prior weight concerning background knowledge on the missing-data mechanism. This is in contrast to the popular approach in the literature, which only considers observed ratings.

As third contribution, we provided a generalized probabilistic framework for factorizing a user-item-matrix that possibly has *multiple observed rating values* associated with each user-item pair. We discussed three important special cases / applications: besides training a top-k recommender system by using virtual data points, we outlined how ratings can be aggregated when items are grouped in a hierarchical manner rather than in a flat list, and how recommendations can be made using this hierarchical structure. Additionally, this framework enabled us to derive the training objective function for a recommender system on sequential data concerning TV consumption. This derivation not only provides a clear understanding of the assumptions underlying the training objective function, but also led to improvements in the top-k hit rate over state-of-the-art approaches in our preliminary experiments.

Acknowledgements

We are greatly indebted to Tin Ho for her encouragement and support of this work. We are also very grateful to the anonymous reviewers for their valuable feedback.

7. REFERENCES

[1] J. Bennet and S. Lanning. The Netflix Prize. In *Workshop at SIGKDD-07, ACM Conference on Knowledge Discovery and Data Mining*, 2007.

[2] BARB: Broadcaster Audience Research Board. http://www.barb.co.uk.

[3] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-N recommendation tasks. In *ACM Conference on Recommender Systems*, 2010.

[4] S. Funk. Netflix update: Try this at home, 2006. `http://sifter.org/ simon/journal/20061211.html`.

[5] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, 2008.

[6] R. Keshavan, A. Montanari, and S. Oh. Matrix completion from noisy entries, 2009. arXiv:0906.2027.

[7] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, 2008.

[8] B. Marlin and R. Zemel. Collaborative prediction and ranking with non-random missing data. In *ACM Conference on Recommender Systems (RecSys)*, 2009.

[9] B. Marlin, R. Zemel, S. Roweis, and M. Slaney. Collaborative filtering and the missing at random assumption. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

[10] R. Pan, Y. Zhou, B. Cao, N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *ICDM*, 2008.

[11] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *KDDCup*, 2007.

[12] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *NIPS*, 2008.

[13] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann machines for collaborative filtering. In *ICML*, 2007.

[14] R. Salakhutdinov and N. Srebro. Collaborative filtering in a non-uniform world: Learning with the weighted trace norm, 2010. arXiv:1002.2780.

[15] C. Senot, D. Kostadinov, M. Bouzid, J. Picault, A. Aghasaryan, and C. Bernier. Analysis of strategies for building group profiles. In *Conference on User Modeling, Adaption and Personalization (UMAP)*, 2010.

[16] N. Srebro and T. Jaakkola. Weighted low-rank approximations. In *ICML*, pages 720–7, 2003.

[17] H. Steck. Training and testing of recommender systems on data missing not at random. In *KDD*, 2010.

[18] M. Weimer, A. Karatzoglou, Q. Le, and A. Smola. Cofi rank - maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

From Recordings to Recommendations: Suggesting Live Events in the DVR Context

Alessandro Basso, Marco Milanesio, André Panisson, Giancarlo Ruffo
Dipartimento di Informatica
Università degli Studi di Torino
Torino, Italy
{basso,milane,panisson,ruffo}@di.unito.it

ABSTRACT

Providing valuable recommendations in the DVR domain is quite straightforward when enough information about users and/or contents is known. In this work, we discuss the possibility of recommending future live events without knowing anything else but past user programmed recording schedules.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering*; J.4 [**Computer Applications**]: Social And Behavioral Sciences

General Terms

Algorithms, Experimentation, Human Factors.

Keywords

Digital Video Recorders, TV Broadcasts, Recommendation Systems, Collaborative Algorithms, Implicit Data

1. INTRODUCTION

A Digital Video Recorder (DVR) is a device aimed at recording digital streams to a storage. DVRs can be either hardware devices, such as set-top-boxes and portable media players, or software devices, such as web/PC-based Personal Video Recorders (PVRs), managing all user interactions and personalizations. By using DVRs, users are no longer bound to the broadcaster's schedule, but are free to define their personal lists of programs at any time.

In order to provide a better user experience by means of focused advices (e.g., recommendation of new contents), the arisen issues can be summarized in two main categories.
First, some logging activity must be done to find common usage patterns on which identify potential users' interests. Users are not willing to offer an *explicit profile* when using a DVR, thus we do have, possibly, only a set of observations on their activity. This is an important challenge for many known recommendation algorithms, that exploit user profiles for increasing accuracy and take into account privacy issues as well.
Second, differently from the Video on Demand domain, the usage of an Electronic Program Guide (EPG) is not always assured. This fact brings two consequences: (a) there is no knowledge on the content the user is recording and/or watching, and (b) there is no well defined one-to-one correspondence between a *recording* and a *broadcast event*. This leads to the impossibility of directly recommending recordings to users.

Taking into account these considerations brings us our research question: in such a domain, is it possible to give valuable live event recommendations to users, only considering their recording activity on the DVR? Users have to be brought to contents of interest, but, differently from other approaches, we are not using anything but collaborative filtering technique on users' activity. Thus, the main contribution of our approach is the demonstration that this can be achieved without any knowledge on what is being broadcast, neither EPGs nor content classifications.

2. RELATED WORK

The task of recommending live events, such as TV shows, has been already investigated in the past years. Proposed methods can exploit different ways to collect the required information for user profiling, as well as can make use of various recommendation algorithms. In particular, some approaches, such as [6], explicitly ask the users about their interests and build suggestions on top of the resulting user profiles. A different idea, which is adopted in several works [2, 9, 10], makes use of *implicit feedbacks*, i.e., information derived from the analysis of the user behavior while using the DVR. Other solutions, as [4, 12, 15], propose recommender systems which make use of user's view history as well as both *explicit* and *implicit feedbacks*. According to authors, such a mixed technique allows to obtain the best performance.

Another feature to tell apart existing methods for live events recommendation is the recommender algorithm used. A common approach relies on the content of the programs broadcast and it is therefore called *content-based*. Examples in this category can be found in [10, 12]. Some authors devised recommenders that make use of multiple content-based techniques, as in [3, 4].
A solution able to increase novelty of recommendations is

collaborative filtering, like the works in [2, 5]. Another interesting method is proposed in [9] and exploits the latent factor model.

In this work, we focus on implicit feedbacks only and we use a collaborative filtering approach to compute recommendations. Our aim is to minimize the information required as input of the recommender system, without sacrificing the novelty. The real challenge is to be able to recommend programs to users without actually knowing anything about what is broadcast on TV, since no EPG is used (differently from existing methods).

3. DATASET

Faucet is a PVR integrated in a podcasting service[1], which allows the recording and further downloading of Italian TV and Radio broadcasts [1]. The activity of the users is incrementally collected (hourly) into a log file containing the scheduled recordings set in the past hour as well as the occurred downloads. The resulting dataset is populated by real users expressing their preferences through the recorded programs. The dataset is publicly available at `http://secnet.di.unito.it/vcast`.

Each registered user can fix the desired settings for the recording of interest. At the end of the process, her recording is scheduled for the given time and will be further available for downloading purposes. Each recording r_i, thus, is defined as a tuple $< u_i, c_i, t_i, b_i, e_i, p_i >$ with the following notation: user u_i sets up a recording on channel c_i, starting from time b_i and ending at time e_i, with a title t_i and a periodicity p_i (e.g., once, every Tuesday, mon-fri). In Faucet, channels and periodicity values are fixed (users can choose their c_i and p_i from a combobox), while all other fields are completely up to the user.

After the end time expires, the recording is made available to the user for downloading. In case of periodic events, the recording step can occur an undefined number of times. After each recording step, the respective download is made available.

4. METHODOLOGY

In this section, we want to outline what our approach is. Given no knowledge on the broadcasts, we collect the users activity to compute what we call *discrete events*, to be used for recommendation purposes and top chart list building.

4.1 From Recordings to Events

The extraction of meaningful information from the unstructured amount of data contained in the dataset is essential to define a set of events which map the broadcast programs. Through the *event discovery* phase, we can discretize the continuous domain of timings defined by the recordings, creating the basis for the application of a recommender algorithm. The basic procedure used in the discretization was first introduced in [1] and covers a number of subsequent steps:

Clustering. Recordings are clustered together by considering the *channel*, the *periodicity* and the difference between *starting* and *ending* times. All recordings belonging to the same cluster are thus equal as channel and periodicity, whilst similar on timings. Specific values are used to define the maximum clustering distance for the start and the end times. The output of this activity is a set of clusters, each identifying a single event. The centroid of the cluster, i.e., the recording that minimizes the intra-cluster timing distances, is considered the representative of the event.

Aggregation. As the clustering occurs periodically, this operation aims to identify newly created events characterized by the same channel and periodicity of the formerly created ones, but comparable timings. Such elements refer to the same programs and are therefore merged into unique events, whose properties are updated by taking into account the values of all the similar ones.

Collapsing. A further refinement phase is required to grant the consistency of the generated events. In fact, due to the high variability of timings, especially when a new transmission appears, events which are initially considered as non referring to the same transmission tend to slowly and independently converge to more stable timeframes. This implies the need of merging them into single events.

As a result of the processing phase, given a set of recording clustered together, each one with the same channel c_i and the same periodicity p_i, we compute a *discrete event* e_i in the form of: $< \{u_i\}, \bar{t}, c_i, \bar{b}, \bar{e}, p_i >$, where $\{u_i\}$ is the set of users whose recordings were clustered together; $\bar{t}$ is the user generated title most frequent among users in $\{u_i\}$; $\bar{b}$ and $\bar{e}$ are, respectively, the starting and ending time computed as the median value of all the clustered recordings.

4.2 From Events to Recommendations

When future events are computed from scheduled recordings, we are thus able to propose them to users by means of two different charts: (1) a global chart returning those events computed starting from the largest groups of recordings, i.e., those chosen by the largest sets of users; and (2) a user-based recommendation list, returning a set of new events of possible interest to each user requesting it, computed through a similarity function over the whole population. We call them *Most Popular* and *Rec*^{2} *(Recordings times Recommendations)*, respectively. Both charts are computed by means of the memory based *collaborative filtering* approach named k-Nearest Neighbors (kNN) [14]. We apply both variants of the kNN algorithm: the user-based one [8], by identifying users interested in similar contents; and the item-based approach [7], by focusing on items shared by two or more users.

In kNN, the *weight* (i.e., a measure of interest) of an element e_i for an user u_k can be defined as:

$$\mathrm{w}(u_k, e_i) = \sum_{u_a \in N(u_k)} \mathrm{r}(u_a, e_i) \cdot \mathrm{c}(u_k, u_a), \qquad (1)$$

where $N(u_k)$ are the neighbors of user u_k and $\mathrm{r}(u_a, e_i)$ is equal to 1 if user u_a is associated to the event e_i, and 0 otherwise. The coefficient $\mathrm{c}(u_k, u_a)$ represents the neighbor's information weight for user u_k. In most of the kNN-based algorithms [8], the coefficient used is the similarity between u_k and u_a.

[1] `http://www.vcast.it/`

Most Popular. The *MostPopular* algorithm can be defined by means of eq. (1), assuming the number of neighbors unbounded, which implies $N(u_k) = U$, $\forall u_k \in U$; and $\mathrm{c}(u_a, u_b) = 1$, $\forall u_a, u_b \in U$, with U as the set of all users. Thus, the weight is modified as $\mathrm{w}(u_k, e_i) = \sum_{u_a} \mathrm{r}(u_a, e_i)$.

After calculating the weight of all elements, they are sorted in descendant order. In the *MostPopular* algorithm, as the set of neighbors is independent of the user, all users receive the same recommended elements, i.e., the most popular ones.

***Rec*^{2}.** In order to provide personal suggestions, we have to define a similarity function for grouping similar users (items) from which choosing the appropriate elements to recommend. Our definition of similarity is based only on implicit feedbacks, resulting from observing the behavior of users: if she records something, then we assume that she is interested in it; otherwise, we can not infer anything about the interest of the user for that element. We are therefore considering binary feedbacks.

Given two users u and v and the associated discrete events E_u and E_v, we can choose the similarity metric, $S(u, v)$, considering several well known measures (e.g., Dice, Cosine and Matching) [11]. After choosing a metric, $\forall u$ we can compute the subset $N_u \subseteq U$ of *neighbors* of user u. A user v such that $E_v \cap E_u \neq \emptyset$ is thus defined as a neighbor of u. Starting from the neighborhood of u, the similarity with u is computed for each pair $< u, v >$ such that $v \in N_u$. Finally, if $S(u, v) > 0$, we consider u similar to v. The value $S(u, v)$ is used to weight such a relation, therefore determining a similarity order among the neighborhood of u, from which choosing new events to recommend to u.

Similarly, this approach can be adopted for the item-based similarity: two events are considered similar if the *share* at least a single user that is associated to both of them.

5. EVALUATION

In the following, we evaluate the obtained results in the event extraction process and in the recommendation of new events to users, both in Most Popular and in Rec^2.

5.1 Event Extraction

As a remainder, we are dealing with several independent, user generated recording schedules, that we cluster together and from which we compute the discrete events. In Figure 1, a view of the distribution of the recordings is given: for each detected event, the number of recordings clustered together changes according to users' activity. As it turns out, most recordings (and, thus, most users) tend to be clustered and aggregated on very few events, while there are lots of events with very few recordings. The Most Popular algorithm exploit these inner features of the resulting discrete events to compute the top chart.

5.2 Computing Recommendation

We measure how accurate is the recommendation in predicting the elements that users would program in terms of recall. These values are computed as the average of all users' recall values using the top n recommended elements [13].

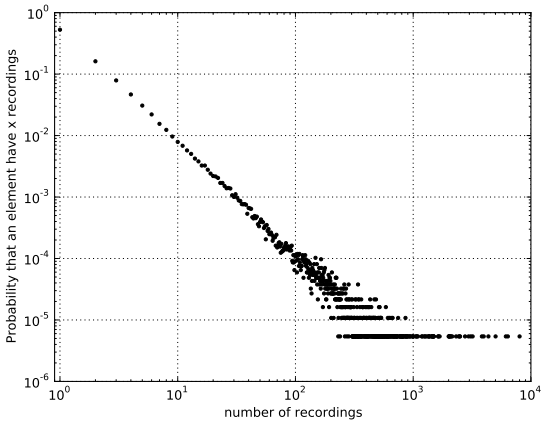


Figure 1: Number of recordings per event (Probability density function)

We are giving particular emphasis on the recall measure; in fact, since we do not have explicit feedbacks regarding the user's interest in those items which have not been considered (i.e., not programmed, nor downloaded), precision is not very meaningful [9].

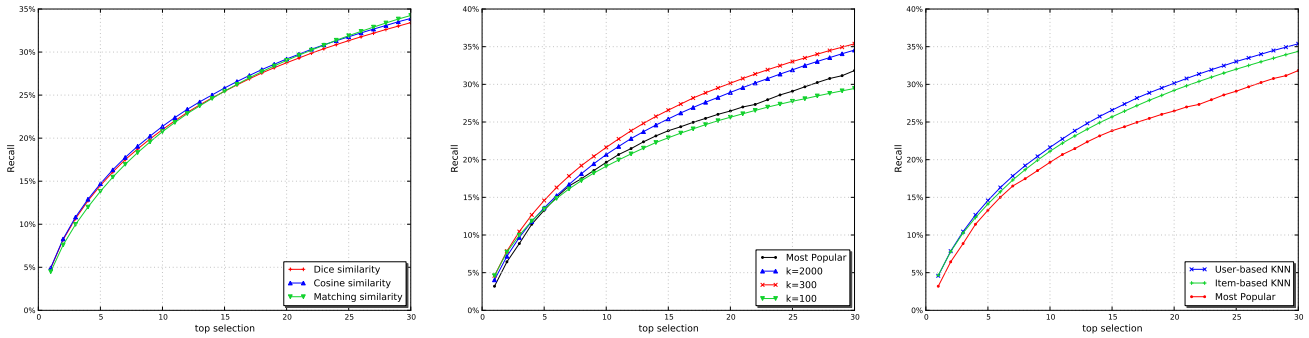
First, we choose different similarity functions to understand whether similarity influences the results of the user-based kNN algorithm. From Figure 2(a) it is clear that, in this case, all chosen similarity metrics show nearly the same performance.

The second step is to find the optimal value for k. Figure 2(b) shows the results with $k \in \{100, 300, 500, 700, 2000\}$ in user-based kNN (Dice similarity), and the *MostPopular* recommender. We omit the values of $k = \{500, 700\}$ since the results are almost equal to $k = 300$. Compared to the *MostPopular* algorithm (i.e., unbounded neighbors), a value $k = 100$ is not enough to outperform it, whilst for $k = 2000$, kNN starts to converge to it. Considering the top 10 recommended elements, we can achieve the best results for $k = 300$, whilst $k = 500$ is more suitable when taking further elements. As in most cases 10 elements are sufficient for a recommendation, $k = 300$ offers a good trade-off between valuable recommendations and resource consumption for building the neighborhood.

A comparison among user/item-based kNN and *MostPopular* is depicted in Figure 2(c). We can observe that the latter is clearly outperformed by the other two algorithms, especially when more than 7 recommended items are considered. The user-based algorithm performs slightly better than the item-based one (more noticeable with more than 15 recommended items). In general, item-based algorithms tend to perform better because usually the number of items is considerably lower than the users [14], but this property does not hold in our domain.

6. CONCLUSION

In this paper we show how to recommend live events to users without any knowledge about the broadcast content



(a) Comparison between similarity functions in user-based kNN

(b) Recall for user-based kNN

(c) Recall for kNN ($k = 300$) wrt Most-Popular

Figure 2: Comparison between recommenders and recall for kNN and Most Popular.

and user's likes. Recommendations can be given both globally and personally. It is important to underline that the most popular events are easier to predict since users tend to naturally focus on them, even without any specific suggestion. On the contrary, granting a high novelty in personal recommendations is a more challenging goal due to the reduced amount of explicit information. Nevertheless, we can obtain interesting results even exploiting a simple approach as the kNN. We are currently attempting other approaches to recommendation (e.g., latent factor model) with implicit feedbacks, with the aim of improving the prediction accuracy.

7. REFERENCES

[1] A. Basso, M. Milanesio, and G. Ruffo. Events discovery for personal video recorders. In *EuroITV '09: Proceedings of the seventh european conference on European interactive television conference*, pages 171–174, New York, NY, USA, 2009. ACM.

[2] P. Baudisch and L. Brueckner. Tv scout: Guiding users from printed guides to personalized tv program. In *In Proceedings of the 2nd Workshop on Personalization in Future TV (May 28, Malaga, Spain), Universidad de Malaga*, pages 151–160, 2002.

[3] Y. Blanco-Fernández, J. J. Pazos-Arias, A. Gil-Solla, M. Ramos-Cabrer, B. Barragáns-Martínez, M. López-Nores, J. García-Duque, A. Fernández-Vilas, and R. P. Díaz-Redondo. A multi-agent open architecture for a tv recommender system: A case study using a bayesian strategy. *Multimedia Software Engineering, International Symposium on*, pages 178–185, 2004.

[4] A. L. Buczak, J. Zimmerman, and K. Kurapati. Personalization: Improving ease-of-use, trust and accuracy of a tv show recommender. In *in Proceedings of the TV'02 workshop on Personalization in TV, Malaga*, pages 3–12, 2002.

[5] P. Cremonesi and R. Turrin. Analysis of cold-start recommendations in iptv systems. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, pages 233–236, New York, NY, USA, 2009. ACM.

[6] D. Das and H. ter Horst. Recommender systems for tv. In *In Proceedings of AAAI*, 1998.

[7] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, 2004.

[8] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237, New York, NY, USA, 1999. ACM.

[9] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[10] S. G. Kaushal, S. Gutta, K. Kurapati, K. Lee, J. Martino, J. Milanski, J. D. Schaffer, and J. Zimmerman. Tv content recommender system. In *In Proceedings of the 17th National Conference on Artificial Intelligence*, pages 1121–1122. AAAI Press / The MIT Press, 2000.

[11] B. Markines, C. Cattuto, F. Menczer, D. Benz, A. Hotho, and G. Stumme. Evaluating similarity measures for emergent semantics of social tagging. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 641–650, New York, NY, USA, 2009. ACM.

[12] M. Rovira, J. Gonzàlez, A. López, J. Mas, A. Puig, J. Fabregat, and G. Fernandez. Indextv: a mpeg-7 based personalized recommendation system for digital tv. In *ICME*, pages 823–826. IEEE, 2004.

[13] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl. Application of dimensionality reduction in recommender system - a case study. In *In ACM WebKDD Workshop*, 2000.

[14] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 285–295, New York, NY, USA, 2001. ACM.

[15] K. K. Srinivas, S. Gutta, D. Schaffer, J. Martino, and J. Zimmerman. A multi-agent tv recommender. In *In Proceedings of the UM 2001 workshop "Personalization in Future TV"*, 2001.

Behavior Based Adaptive Navigation Support

Michal Holub
Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology
Ilkovičova 3, 842 16 Bratislava, Slovakia
holub@fiit.stuba.sk

Mária Bieliková
Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology
Ilkovičova 3, 842 16 Bratislava, Slovakia
bielik@fiit.stuba.sk

ABSTRACT
Web portals contain large amount of information. Users could really benefit from it if personalized presentation is used. For this to be accomplished the website needs to "know" its users. When surfing the Web users leave digital footprints in the form of navigational paths and actions taken. We present a method for adaptive navigation support and link recommendation. The method is based on an analysis of the user's navigational patterns and behavior on the web pages while browsing through a web portal. We extract interesting information from a web portal and then recommend it. Finally, we provide our experience with a recommender system deployed on our faculty's website, which recommends events by means of personalized calendar.

Categories and Subject Descriptors
H.5.4 [**Information Interfaces and Presentation (e.g., HCI)**]: Hypertext/Hypermedia—*Navigation*. H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Relevance Feedback*.

General Terms
Algorithms, Design, Experimentation, Verification.

Keywords
Adaptive navigation support, automatic interest estimation, behavior, link recommendation, navigational patterns.

1. INTRODUCTION
Web portals are being visited by various users pursuing different goals. However, most websites offer all groups of visitors the same content. Therefore the visitors are often presented information in which they have no interest [4].

While browsing through a web portal some users can discover interesting pages that are hidden deeper in the hierarchy of the portal. If the users with similar goals knew about these pages they could find them interesting, too. Personalized navigation and recommendation based on monitoring user activities and social principles is a viable approach for such cases. In this paper we introduce a method of navigation adaptation and social recommendation of links among users with similar behavior.

2. RELATED WORK
People use different means of accessing information on a web portal. The most common way is to follow hyperlinks, which accounts for more than half of all possibilities [7]. This introduces a problem with improper navigation containing large number of links. The user has often a problem of deciding which link to use. Therefore a recommendation of links on the web page could bring significant improvement to user's browsing experience. Other dominant mean of navigation is using the browser's back button [12]. Accessing websites through the history, bookmarks and other means is insignificant.

User's habits can be derived from the navigational patterns found in the sequences of links he uses in a particular web portal. Four basic navigational patterns (path, loop, ring and spike) were described in [6]. From the prevailing patterns in browsing sessions different browsing strategies can be identified.

User's interests are often determined based on the content of documents the user has read [5]. The user model can be expressed by concepts or keywords extracted from these documents [2]. If we know what topics (usually expressed by the keywords) the user prefers, we can recommend him documents (web pages) with similar content. The disadvantage is that documents should be written in language which we can process (a translation can help).

In [16] authors use rather different approach based on user behavior tracking to estimate his interest. For this we need to get a feedback from the user. There are several ways how to implicitly determine user's interest. When links are well annotated (like on news portals where links to articles contain a short introduction) the event of clicking on the link is considered as positive interest [8, 14]. However, in general scenarios we cannot always consider clicking on a link as truly positive interest in the web page.

To determine user's interest we can also use actions he makes on a web page [11]. Printing the page or adding it to bookmarks show positive interest. Spending very short time reading it or even closing the browser prematurely shows negative interest [16].

With user's interest determined navigation personalization as well as link recommendation can be done [9, 13]. In [10] authors propose a method of interesting link recommendation by highlighting the links. They extract keywords from the pages a user visits and recommend links that lead to other pages which

contain the same keywords. Adaptive system *Web Watcher*, which implements this approach, can also show similar pages to the page that is currently being viewed based on this principle. The system uses a proxy server to incorporate its toolbar into every web page.

Other method is based on monitoring the context in which the links are being used [1]. This method consists of creating a knowledge base from the links each user has clicked on. Then the clusters of links, which are often used together, are built from the knowledge base. Links from a cluster with the largest overlap with the current session of the user are recommended to him.

All methods mentioned share the same feature which is user interest estimation based on his actions. They prefer behavior of the users over content of the documents which they were shown.

3. ADAPTIVE LINK RECOMMENDATION

We propose a method for adaptive recommendation of interesting links in a particular web portal (which we may or may not own). For a specific user we select links that similar users found interesting. We also recommend links to this user based on his previous surfing sessions. Our recommender system extracts further information from the web pages, which is also shown to the user. The recommended links are shown in special sections added to each web page of the portal.

When deciding which link to recommend we do not consider the content of web pages. We based our recommendations solely on the analysis of user's behavior. Our method thus does not depend on the language of the website. We are able to analyze interest and patterns on different language versions of the same portal. Our method of adaptive link recommendation works in two steps:

1. Mining web usage history for navigational patterns.
2. Recommendation of links based on user's behavior.

In the first step we analyze the sequences of followed links from each user's session. In these sequences we look for navigational patterns. We use the prevailing pattern to categorize users, as it determines user's surfing habits on a particular web portal. As an output we get groups of users with similar navigational patterns.

In the second step we monitor behavior of users on each visited web page of the portal. From their actions we automatically estimate their interest in that page. We then recommend links to interesting pages among users of each group from the step one.

3.1 Discovering navigational patterns

We find similar users based on comparison of navigational patterns they follow in a closed web portal. We believe that users who follow analogous paths should be recommended similar links. There are four basic navigational patterns according to [6]:

- *Path* – a sequence in which nodes do not repeat.
- *Ring* – a sequence that starts and ends in the same node.
- *Loop* – a sequence that goes through already visited node.
- *Spike* – a sequence that goes back through the same trail.

In each session a user visits several pages of the web portal. This session is described by a vector whose elements are links to the web pages arranged in order they were visited. We consider a continuous sequence of links to be a session. For this purpose we use the *referrer* field of HTTP request message. If the URL of previously visited page equals referrer value of currently visited page, we consider the pages to be in the same session.

The process of dividing users into groups is presented in Alg. 1. Similarity of users is expressed as Pearson correlation coefficient [15] commonly used in collaborative filtering.

Algorithm 1 Group users according to their similarity.

```
1: for each user u do
2:     find patterns in clickstreams of u
3:     put u to group according to prevailing pattern
4: for each group g do
5:     for each user u in group g do
6:         sort users in group g according to their
           similarity to u
```

Every user ends up in exactly one group according to the most dominant pattern found in his surfing history. There is one more group for users with no dominant navigational pattern. The order of similar users from one group is unique for each user.

Alg. 2 presents the process of recommending links among users.

Algorithm 2 Recommend links for user *u* by similar users.

```
1: similar = select top K similar users
2: for each user v in similar do
3:     for each page p visited by v and not visited by u do
4:         predict interest of user in page (u, p)
5: recommend top M pages with highest predicted interest
```

Navigational patterns of users have to be of a certain minimal length (so that each sequence of two following pages does not represent a *path* pattern). After finding similar users to user *u* we select top *K* of them to form a recommendation group. The groups change according to new browsing sessions in which the users can behave differently. This reflects the evolution of user's behavior in time. However, at each time the user belongs to exactly one group according to the most dominant pattern in browsing history.

3.2 Determining interest of users

In order to recommend links to a particular user we need to evaluate the interests of the users in his recommendation group. We can recommend pages which other users liked. To determine user's interest in a particular web page we observe actions he makes on this page. These include *time spent on a web page*, *number of scrolling events* that occur and *number of times he copies text into the clipboard*. We chose these interest indicators because their tracking is platform independent.

Our method is based on the comparison of current user's behavior with the behavior of other users. We compare the values of time and scrolling with values from other people who visited the same page. If the value for the current user is more than X % higher than the average, we consider it as a sign of positive interest in the page. In contrast, when it is more than X % lower than the average we consider it as a sign of negative interest. When the value is around average (± X %) it is a sign of neutral interest. This way we can also consider other interest indicating actions. The exact value of X depends on the calibration for selected domain; in our experiments we used the value of 20 %.

When no behavioral data for a particular web page is available we cannot estimate the user's interest. This is a problem with newly added pages as well as with pages visited for the first time.

We estimate the actual value of user's interest in each page he visits according to Figure 1. We increase this value by 0.1 when the user also copied text into clipboard; otherwise we decrease it by 0.1. Resulting interest is in the interval <0,1> with 0 meaning no interest and 1 meaning total interest in the visited web page.

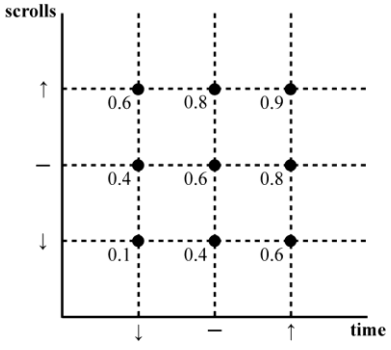


Figure 1. Estimation of user's interest from his actions.

3.3 Social recommendation of links

We recommend web pages by predicting user's interest in yet unseen pages using collaborative filtering method. We compute the predicted value of interest like this [15]:

$$p_{a,i} = r_a + \frac{\sum_{u=1}^{N} (r_{u,i} - r_u) \times S_{a,u}}{\sum_{u=1}^{N} S_{a,u}}$$

where $p_{a,i}$ means prediction of interest of user a in page i, r_a is the average interest of user a in all visited pages, $r_{u,i}$ is the interest of user u in visited page i, $S_{a,u}$ is the value of Pearson correlation coefficient [15] between users a and u determining the similarity of their interests, and N is the number of similar users.

4. EVALUATION AND EXPERIMENTS

To evaluate the proposed method for user's interest estimation we developed software tools which support adaptive navigation by recommending information extracted from potentially interesting web pages to guests of particular web portal. We experimented with the web portal of our faculty (www.fiit.stuba.sk).

We proposed client-server architecture with an adaptive proxy [3] in the middle as shown in Figure 2. Adaptive proxy can be extended to conduct various methods of web personalization on any web portal. We use proxy to put behavior tracking script into the web page. It sends logged behavioral data to the server when the user is active (i.e. when he uses the mouse). The user is aware of this when he agrees to use our proxy server. The data is anonymous – we only know a random ID associated with each user. The delay caused by the proxy server is imperceptible.

One component (*SpyImp*) creates the domain model by analyzing web pages of selected web portal. Another server component (*AdaptiveImp*) is responsible for grouping of users, estimating their interests and making predictions for unseen pages. Then it selects the links to be recommended. The user model consisting of the session vectors and his behavior is being periodically updated.

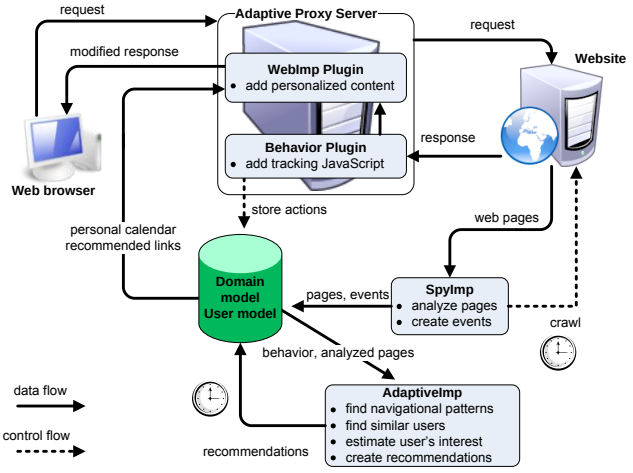


Figure 2. Architecture of proposed link recommender system.

Our plug-in to the adaptive proxy (*WebImp*) modifies the web page by adding special sections with recommended links. One of those sections is personalized calendar. Many web pages on the web portal of our faculty inform about an upcoming event. We automatically extract dates from these pages and create events. Using proposed method we determine user's interest in such page. Then, if the interest is positive, we add the event to user's calendar. This way we also recommend events among users.

We monitor the portal and capture every change in text of a web page (this could be for example a change in time and place of some event). Every changed page is marked as *news* and added to a special news section. We also recommend other potentially interesting links which are neither *events* nor *news*. Figure 3 shows part of a web page enhanced with personalized sections.

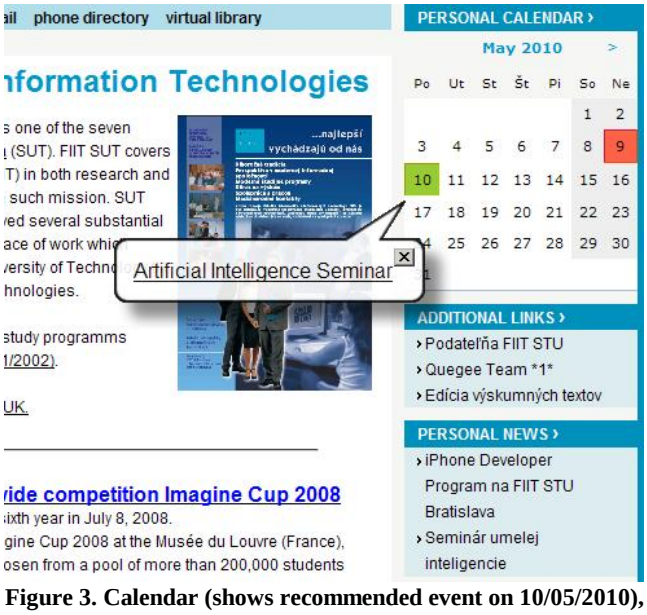


Figure 3. Calendar (shows recommended event on 10/05/2010), additional links and personal news sections.

We provided a series of experiments on our faculty website. Actions of 24 users on a modified website were monitored for 3 weeks. We compared our calculations with their explicit feedback.

Results indicate that time actively spent on a web page is the best interest indicator. Scrolling proved to indicate positive interest as well. However, when the user does not use scrolling, it does not always mean he is not interested in the page. The accuracy of our interest estimation method was 62 %.

The sections with recommended links – especially calendar – were attractive (according to answers from questionnaire) and the users found 55 % of recommended links and events interesting.

Some users were not satisfied with the recommendations. The problem was that they visited the website for the first time. Hence their user model was empty and we could not provide suitable recommendations. This is a common problem with recommender systems and new users. We tried to overcome it by recommending the most interesting events (links) according to behavior of all users. However, this is not always a suitable solution.

5. CONCLUSION AND FUTURE WORK

We have presented a method for adaptive recommendation of interesting links. Our approach is based on collaborative filtering, which has a potential to be used in unusual ways. We presented one of them when considering data about user's actions instead of content of pages. This way we are able to predict user's interest for unvisited pages. Our method achieves solid results and can be further improved in a recommender system which will combine content analysis with behavior, which is our plan for future work.

In this paper we presented a useful application of our method by creating personalized calendar of events on our faculty's website. Using this method we can also personalize other sections of a web page as well. In our opinion recommender systems should bring added value to users by doing further analysis of the domain which is being adapted. On the web they should recommend particular objects (e.g. events) instead of simply listing potentially interesting links to web pages. In order to accomplish this we need to use more text processing algorithms in our recommender systems so that they "understand" the meaning of text on the web.

We ran up against a problem with incorporating the sections with personalized content into a website. In order to do this we need to know the semantics of the website's structure. This is also useful during page analysis and content extraction. We consider the special tags in HTML5 (e.g. *nav*, *footer*) to be insufficient so we came up with a descriptive XML file which pairs HTML tags and their IDs with their predefined meaning (left menu, right menu, etc.). This way our recommender system understands the structure of a website and can alter some sections. We think that there is a need for further development of this format and we see an opportunity for its adoption by other recommender systems.

6. ACKNOWLEDGEMENTS

This work was supported by grants VG1/0508/09, KEGA 028-025STU-4/2010 and the Foundation of Tatrabanka. It is a partial result of the Research & Development Operational Program for the project SMART II, ITMS 25240120029, co-funded by ERDF. We wish to thank members of PeWe group, pewe.fiit.stuba.sk for valuable discussions and their help in experimental evaluation.

7. REFERENCES

[1] Baraglia, R., et al. 2006. A Privacy Preserving Web Recommender System. In *Proc. of the ACM Symposium on Applied Computing*, (Dijon, France). ACM Press, 559-563.

[2] Barla, M. and Bieliková, M. 2009. On Deriving Tagsonomies: Keyword Relations coming from the Crowd. In *LNAI 5796, Proc. of Int. Conf. on Computational Collective Intelligence*, ICCCI 2009, Springer, 309–320.

[3] Barla, M. and Bieliková, M. 2010. Ordinary Web Pages as a Source for Metadata Acquisition for Open Corpus User Modeling. In *IADIS Int. Conf. WWW/Internet.* To appear.

[4] Barla, M., Tvarožek, M. and Bieliková, M. 2009. Rule-based User Characteristics Acquisition from Logs with Semantics for Personalized Web-based Systems. *Computing and Informatics*, Vol. 28, No. 4, 399-427.

[5] Brusilovsky, P. 1996. Methods and Techniques of Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, Vol. 6, No. 2-3, Springer Netherlands, 87-129.

[6] Canter, D., Rivers, R. and Storrs, G. 1985. Characterizing User Navigation through Complex Data Structures. *Behavior and Information Technology*, Vol. 4, No. 2, 93-102.

[7] Cockburn, A. and McKenzie, B. 2001. What do Web Users Do? An Empirical Analysis of Web Use. *Int. Journal of Human-Computer Studies*, Vol. 54, No. 6, 903-922.

[8] Das, A.S., et al. 2007. Google News Personalization: Scalable Online Collaborative Filtering. In *Proc. of the 16th Int. Conf. on World Wide Web* (Banff, Canada), ACM Press, 271-280.

[9] Gurský, P., et al. 2009. User Preference Web Search – Experiments with a System Connecting Web and User. *Computing and Informatics.* Vol. 28, No. 4, 515-553.

[10] Joachims, T., Freitag, D. and Mitchell, T. 1997. WebWatcher: A Tour Guide for the World Wide Web. In *Proc. of the Int. Conf. on Artificial Intelligence* (Nagoya, Japan), Morgan Kaufmann, 770-777.

[11] Krištofič, A. and Bieliková, M. 2005. Improving Adaptation in Web-Based Educational Hypermedia by means of Knowledge Discovery. In *Proc. of 16th ACM Conf. on Hypertext and Hypermedia*, ACM Press, 184-192.

[12] Milic-Frayling, N., et al. 2004. Smartback: Supporting Users in Back Navigation. In *Proc. of the 13th Int. Conf. on World Wide Web* (NY, USA), ACM Press, 63-71.

[13] Návrat, P., Taraba, T., Bou Ezzeddine, A. and Chudá, D. 2008. Context search enhanced by readability index. *IFIP WCC Series 276*, Springer, 373-382.

[14] Suchal, J. and Návrat, P. 2010. Full Text Search Engine as Scalable k-nearest Neighbor Recommendation System. *IFIP WCC Series 331*, Springer, 165-173.

[15] Sugiyama, K., Hatano, K. and Yoshikawa, M. 2004. Adaptive Web Search based on User Profile Constructed without any Effort from Users. In *Proc. of the 13th Int. Conf. on World Wide Web* (NY, USA), ACM Press, 675-684.

[16] Velayathan, G. and Yamada, S. 2006. Behavior-based web page evaluation. In *Proc. of the 15th Int. Conf. on World Wide Web* (Edinburgh, Scotland), ACM Press, 841-842.

An Architecture for a General Purpose Multi-Algorithm Recommender System

Jose C. Cortizo, Francisco M. Carrero and Borja Monsalve
BrainSins
http://www.brainsins.com
Madrid, Spain
{josecarlos.cortizo, francisco.carrero, borja.monsalve}@brainsins.com

ABSTRACT

Although the actual state-of-the-art on Recommender Systems is good enough to allow recommendations and personalization along many application fields, developing a general purpose multi-algorithm recommender system is a tough task. In this paper we present the main challenges involved on developing such system and a system's architecture that allows us to face this challenges.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Design, Algorithms

Keywords

General purpose recommendations, System architecture, API, Multi-algorithm

1. INTRODUCTION

There is a lot of literature on Recommender Systems for specific online domains like social software items [4], music [1], queries (in search engines) [8], news [6], e-commerce [7] or even for non online domains such as [5]. Those recommender systems employ specific techniques for specific domains in order to produce the most accurate systems for each single domain.

We wanted to integrate a recommender system in Wipley[1], our social network for videogamers launched by the end of 2009. With this purpose, we worked on a recommender system for videogames using a collaborative filtering approach with multidimensional and contextual features to fit this particular domain. After that, we wanted to improve the recommender with a content-based recommender system for those games with none or few ratings and a social-based recommender system [3]. We also needed to adapt the resulting system to other domains, beginning with image recommendations to be integrated in FlickrBabel[2], our multilingual multimedia search engine. And, finally, we decided to use a "Software as a Service" (SaaS) model to separete rommendations from the rest of the platform.

Although there exist several commercial approaches to general SaaS recommender systems, like Strands[3] or DirectedEdge[4] there exist no literature focusing on the system's aspects of general recommender systems. In this paper we describe our experience developing our general purpose multi-algorithm recommender system, which is currently being used to personalize our products and services at BrainSins and will be used as experimental platform to compare and evaluate our further research on recommender systems.

In the next section we describe the main challenges we found in order to develop a general purpose multi-algorithm recommender system. In section 3 we describe the general architecture of the system focusing on the elements that allowed us to solve the main issues, and in section 4 we focus on our recommender system API, which enables all our products and services to interoperate with the recommender system. Section 5 describes the next research works we will face, and section 6 concludes the paper.

2. MAIN CHALLENGES FOR A GENERAL PURPOSE MULTI-ALGORITHM RECOMMENDER SYSTEM

When designing a general purpose multi-algorithm recommender system, we found several challenges that had to be addressed in order to develop a useful system.

- Interoperability: Recommender systems are usually created to access a specific database that uses a well-known data structure. However, since our system had to offer the possibility of being integrated into several different existing platforms, we had to deal with the problem of accessing sources with different data structures.

[1]http://www.wipley.es

[2]http://www.flickrbabel.com
[3]http://recommender.strands.com
[4]http://www.directededge.com

- Configurability and easy of use: As one of the main goals of this recommender system was to be able to manage several recommender algorithms to provide a particular recommendation, the system had to be highly configurable. In these cases usability may become a problem difficult to solve.

- Performance: Recommendations must be served in real time, but users do not tolerate an increase in page downloading time. However, when a system is designed with a high degree of configurability, there are always some issues that slow down performance. In our particular case, the new system had to be as effective as the recommender we already had on production.

- Disk usage: A highly configurable system also presents disk space problems, due to the fact that data representation cannot be optimized.

- Scalability: When applied to the web, the number of items to be recommended and the number of users to receive those recommendations often grow exponentially. Therefore, a recommender system needs to be scalable in order to grow at the same rate. However, being scalable also means that the system should somehow hide the way it scales, so there should be no need to code or rewrite configuration parameters in the products and services that access the recommender system.

After analyzing possible solutions for those challenges and start designing and developing the system, we thought that a key fact to achieve most of them was to conveniently separate recommendation process from the interface with the clients, so interoperability became our first focus. Our goal was to design a way to easily integrate the recommender system while keeping recommender complexity hidden from its clients, so we designed a REST API. REST APIs are easy to use and to integrate in any product and service on the web, and many developers are familiar to it. Furthermore, REST API helps to hide the complexity of the recommender system, and allows to transform the REST petitions into more complex data structures needed to maintain the performance and scalability of the recommender system.

To face configuration and performance issues we designed a back-end architecture that enables us to define recommendation algorithms as software modules that can be adapted to any domain required by clients. The REST API also allowed us to introduce configurability elements as optional parameters in the petition.

3. GENERAL ARCHITECTURE

The input to our system are API requests, which can be classified as online or batch:

- Online requests, which must be handled in real time. Their processing can't be delayed, because users are waiting for a response. They are also utilized to update the profile for new users and begin to provide them with recommendations.

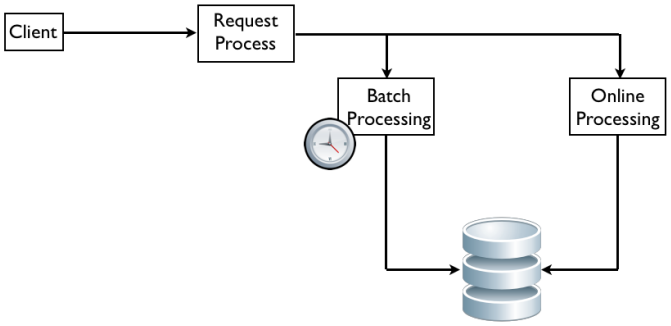


Figure 1: The request processor evaluates if a certain API request needs to be run online or it can be batched.

- Batch requests, which may be stored and processed only at given time periods. Now requests processing can be delayed and attended when the system is not at full capacity. These requests are used to upload the initial data from a client and also to update information concerning users with a wide user profile.

Figure 1 shows how API requests are being processed. Each API request generates an HTTP request to a certain endpoint where the Request Processor evaluates it and determines whether it must be processed at that moment or it can be delayed until more requests reach the system (for a more optimum processing) or until certain batch process is programmed to be run.

Requests can also be classified as update or retrieval:

- Retrieval requests just ask the system to return some kind of information, such as a recommendation.

- Update requests have the objective to update the profile of the source user.

When an update request begins to be processed there are two steps that must be taken to produce recommendations for the user. As we wanted to be able to process several types of recommendations (collaborative filtering, content based, social recommendations), the system had to be general enough to process data in several ways. So we defined those steps in a way that enabled the use of any possible recommender algorithm (see Fig. 2):

- Update user profile. This can be done by re-calculating simmilarity with other users, re-calculating trust or updating a content-based profile.

- Update user recommendations. This step uses the values obtained from the previous task as input for the recommendation algorithm and produces a new rank of recommendations for the user.

As a first approach we coded 3 different recommender algorithms:

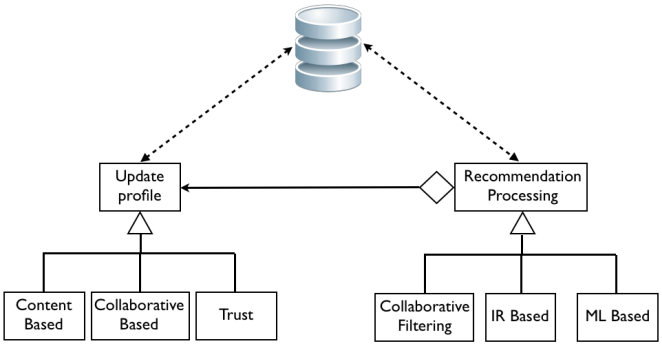


Figure 2: Main modules of the recommender system.

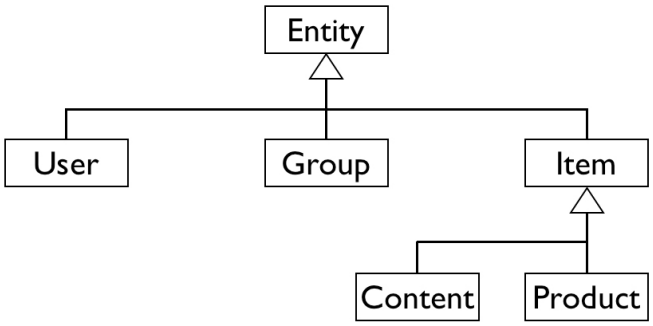


Figure 3: Entities describes every possible data used by the recommender algorithms.

- Collaborative filtering. A standard method that produces collaborative recommendations when using a collaborative similarity measure, or social recommendations when using trust.
- Information retrieval based recommender used for content based recommendations.
- Machine learning based recommender also used for content based recommendations.

From the data point of view, the system only stores entities and relationships among entities. Each possible data point (user, group, content or product), is represented by an entity (see Fig. 3). Relationships (see Fig. 4) among entities represent actions (ratings, comments, reviews) or behaviors (buy a product, pageview, joining a group). This data representation allows enough degree of data abstraction for the interoperability, and it also stands near enough to data representation used by the different recommender algorithms.

4. INTEROPERABILITY: A GENERAL API

A Representational State Transfer (REST) API [2] is a style of software architecture for distributed systems like the Web where clients initiate requests and the servers process requests and return appropriate responses. Requests and responses are built around the transfer of resources. REST is described in the context of HTTP, although it can be used in other contexts.

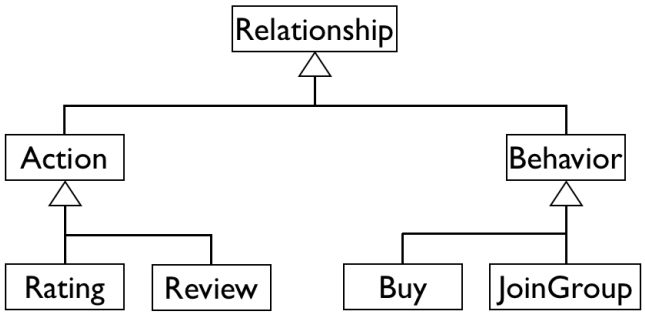


Figure 4: Relationships describes actions and behaviors.

Figure 5: Integration of the recommendations in Wipley, our social network for videogamers.

A client request is described by an HTTP request to a certain resource. For instance, for one of our domains, we may want videogames recommendations for a certain user (user id=2). With our REST API, the client must make an HTTP request to a certain URI[5]. The first part of the URI describes the endpoint where our API server is running[6]. Next part shows that a recommendation (and the recommendation's type) is requested ('/recomm') for a certain entity type ('/player') considering a relation of ownership with another entity type ('/has/videogame'). The user id is '2' and the response format is '.xml'. The server must return an XML file containing the recommendations.

Figure 5 shows our integration on Wipley, our social network for videogamers, where XML responses are processed with PHP in order to generate a more visual interface.

In Wipley we have configured our system to serve different types of recommendations for a user, such as products, users and groups. They can be requested just modifying the URI. In order to update the information that the recommender system manages internally, we have designed a XML based specification, which allows to update information about any entity (data points, see Fig. 6) or action (relationships among entities, see Fig. 7).

[5]http://webservices.brainsins.com/api/recomm/player/has/videogame/2.xml
[6]http://webservices.brainsins.com/api

```
<?xml version="1.0" encoding="UTF-8" ?>
<recsins version="0.1">
     <entity id="product01">
          <property name="entityClass">product</property>
          <property name="productName">Call of Duty:
Modern Warfare 2</property>
          <property
name="productClass">videogame</property>
          <property name="price">12.99</property>
     </entity>
</recsins>
```

Figure 6: Example XML containing a description of an entity (videogame).

```
<?xml version="1.0" encoding="UTF-8" ?>
<recsins version="0.1">
     <action id="action0000001">
          <property name="timestamp">2010-01-01 T
10:43:21 UTC</property>
          <property name="origin">user01</property>
          <property name="destination">item01</property>
          <property name="actionType">rating</property>
          <property name="actionValue">3</property>
     </action>
</recsins>
```

Figure 7: Example XML containing a description of an action (user rating a videogame).

5. FUTURE WORK

We have developed this general purpose multi-algorithm recommender system and we have integrated it into Wipley with great results, given that videogames recommendations are considered by our users as one of the top features. Our future work is defined in several lines:

- Test the performance of the general recommender system in terms of running time and disk usage. Our first impressions make us think that generalizing the recommender system does not introduces computational overhead, since processing the REST requests represents only a 3-4% of the total running time of a recommendation, but we need to run a larger set of experiments to evaluate the final performance.

- Scalability. As the REST API give us a lot of independence of the clients from the implementation, we are actually re-coding the recommender algorithms through a map-reduce perspective using Apache Mahout[7], which will allow the platform to be really scalable.

- Experimental platform. One of our main goals is to obtain an experimental platform to test the performance of our recommender systems implementations. We are starting to measure several metrics related to web analytics such as CTR. We expect to obtain real feedback about what recommender algorithms and settings works better for different domains. We are also developing a web based backend which will allow us to define the experiments and measure several aspects related to the effectivity of the recommendations.

[7]http://lucene.apache.org/mahout/

There are no references in the recommender systems literature describing a general purpose recommender system being used to test several recommender algorithms within several domains with the aim to produce an extensive experimental comparison of recommender algorithms. We have developed a general recommender system which encapsulates several recommender algorithms (collaborative filtering, content based recommender, and social recommender) with the main purpose of producing an extensive comparison of recommender algorithms in a real environment. We have also integrated this general recommender system in a real social network that represents an experimental setup with real users in real conditions.

6. ACKNOWLEDGMENTS

The research described in this paper has been partially supported by the Madrid autonomous region, IV PRICIT, S-0505/TIC/0267.

7. REFERENCES

[1] O. Celma and P. Lamere. If you like the beatles you might like...: a tutorial on music recommendation. In *MM '08: Proceeding of the 16th ACM international conference on Multimedia*, pages 1157–1158, New York, NY, USA, 2008. ACM.

[2] R. T. Fielding. *Architectural Styles and Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[3] J. Golbeck. Tutorial on using social trust for recommender systems. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, pages 425–426, New York, NY, USA, 2009. ACM.

[4] 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 *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, pages 53–60, New York, NY, USA, 2009. ACM.

[5] J. F. McCarthy. The challenges of recommending digital selves in physical spaces. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 185–186, New York, NY, USA, 2007. ACM.

[6] O. Phelan, K. McCarthy, and B. Smyth. Using twitter to recommend real-time topical news. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, pages 385–388, New York, NY, USA, 2009. ACM.

[7] J. B. Schafer, J. Konstan, and J. Riedi. Recommender systems in e-commerce. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166, New York, NY, USA, 1999. ACM.

[8] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In L. Carr, D. D. Roure, A. Iyengar, C. A. Goble, and M. Dahlin, editors, *WWW*, pages 1039–1040. ACM, 2006.

Open Source Recommendation Systems for Mobile Application

Renata Ghisloti De Souza
LISITE-ISEP
28 rue Notre Dame Des Champs
75006 Paris
renata.ghisloti@isep.fr

Raja Chiky
LISITE-ISEP
28 rue Notre Dame Des Champs
75006 Paris
raja.chiky@isep.fr

Zakia Kazi Aoul
LISITE-ISEP
28 rue Notre Dame Des Champs
75006 Paris
zakia.kazi@isep.fr

ABSTRACT

The aim of Recommender Systems is to suggest useful items to users. Three major techniques can be highlighted in these systems: Collaborative Filtering, Content-Based Filtering and Hybrid Filtering. The collaborative method proposes recommendations based on what a group of users have enjoyed and it is widely used in Open Source Recommender Systems. The work presented in this paper takes place in the context of SoliMobile Project that aims to design, build and implement a package of innovative services focused on the individual in unstable situation (unemployment, homeless, etc.). In this paper, we present a study of open source recommender systems and their usefulness for SoliMobile. The paper also presents how our recommender system is fed by extracting implicit ratings using the techniques of Web Usage Mining.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Performance evaluation (efficiency and effectiveness); H.2.8 [**Database applications**]: Data mining—*Web usage mining*

General Terms

Algorithms, Experimentation, Theory

Keywords

Open source recommender systems, collaborative filtering, Mahout, Web usage mining

1. INTRODUCTION

The amount of information in the web has greatly increased in the past decade. This phenomenon has promoted the advance of the recommender systems research area. These systems intend to help users by providing useful suggestions to them. They may suggest items in different manners, such as comparing the user taste with other users tastes or comparing the users preferences with other items definitions. These two methods are the so called collaborative filtering [1] and content-based filtering [10]. The collaborative method presents advantages over the content-based one. It is more efficient in practice and simpler to implement. Due to this fact, the majority of open source projects choose it. Current open source recommender system projects are usually built on the item-based approach, a type of collaborative filtering. Their features vary on the programming language, extent of documentation and magnitude of the project.

We give in this paper an overview on known recommendation techniques and we analyze open source projects in this field of research. Our interest of recommender systems is justified by the fact that we have to choose one of the studied systems and to integrate it in a complex platform that includes a Web platform, a personalization system and a mobile interface. This platform is developed through the SoliMobile project, funded by ProximaMobile [12].

The SoliMobile project in which we are involved, aims at providing a portal services helping and assisting people who are in different unstable situations. This project provides end users with information to facilitate the process to access to charities services from anywhere. The portal has to offer services adapted to each user profile, taking into account their preferences and navigation traces. Our work aims to provide the user with a recommendation of items (services) based on the profile. The recommendation's main function is to aggregate content from different sources and mobile Web portal and to customize the presentation of services for each user according to his profile. It allows classification or restriction of services into a selection that fits the user profile.

The remainder of this paper is organized as follows. Section 2 describes the context of the work presented in this paper. In Section 3 , we present the global architecture of the SoliMobile Project. Section 4 details the analysis of existing Open Source Recommender Systems. The recommender system used in the project is explained in section 5. Section 6 presents the utility of web usage mining in the recommendation. Finally, Section 7 concludes this paper and gives an outlook upon our ongoing and future research in this area.

2. WORK CONTEXT

The work presented in this paper fits in a collaborative project that aims to design, develop and implement a set of innovative services focused on persons in situations of instability or emerging from instability, in order to help them to find useful information such as jobs, offers of housing, welfare, or medical assistance. The charity association partner in the project observed that a large majority of people in unstable situation own a mobile phone that is considered as a link with family, friends or society. The project aims to facilitate, for the vulnerable people, processes to access charities services using their mobile phone from anywhere. However, different services are not suitable for all persons in a precarious situation. For example, a single mother needs child services such as pediatrics or nursery while an unemployed needs services to find a job or professional training.

Our role in this project is to enhance and customize services to users. In this context, we deal with the implementation of algorithms to adapt to user profile the platform services, to filter them and to show only items that may be of interest. Personnalization according to user profile is based both on data available on the platform (eg. databases), the features and traces of user navigation, and also the social environment of the user (collaborative filtering approach).

Typically, adaptation to the user profile will consolidate the resources (services) to target only the relevant users. Conversely, user profiles will also be ordered to form homogeneous groups in order to assign them to a given resource.

3. GLOBAL ARCHITECTURE

We present in this section the overall architecture of the application, illustrated in Figure 1, in order to show the role of the recommendation in the SoliMobile platform. In fact, end users create an account via the Web platform where many services are provided. The traces of Web browsing (also called logs) are collected from servers to feed the recommender system. These navigation traces will be used to create the user item ratings matrix. Services play the role of items. Information regarding the user profile such as age, address, occupationo or preferences as well as information concerning the description of services such as the category of services (health, employment, child care, etc..) and their addresses will be provided as input of recommender system. These inputs will be sent in XML format through Web services. Once the ratings matrix constructed, the recommendation is made to categorize and customize the layout of proposed services on the mobile phone. The recommendation system will provide as output an XML file that contains a subset of sorted services to be transmitted to the mobile. Traces of mobile browsing will also be used as input to the recommender system to improve results, they can also serve as a feedback to our system.

Our goal is structured along the following lines:

- Construct a generic model for the user profile and also for structural and semantic information of the application in order to integrate new data when needed;
- Select, automatically and dynamically, variables describing the user, the services and the log navigation that improve the quality of the recommendation;
- Ensure the proper functioning of the recommender system in case of registration of a new user whose profile is poor (or nonexistent) or in case of creating new services (items) that no one (in our data set) has yet rated or visited. This problem is well known in the field of information filtering and is referred as "Cold Start" problem. Almost solutions for the cold-start problem [Lam et al. 2008] are not suitable as they involve users to rate items.
- Develop a generic recommender system, i.e. that adapts to any application. The challenge is to design a real-time recommender system that filters resources dynamically depending on variation in user interests but also on variation in the environment. The idea is to associate with each resource a ranking based on the user profile and its context. We use for that incremental learning techniques [3] and mining data streams [4] that requires a limited number of passes on data and needs to process data on the fly. Using these methods improves computation time and memory space so we can ensure robustness and scalability of the system;
- Define satisfactory indicators in order to assess the quality of the recommendation;
- Conduct a software platform integrating all the tools developed during the project.

  Given the short duration of the project (18 months), we decided to study open source recommender systems. Thus, we present in the following section the related state of the art.

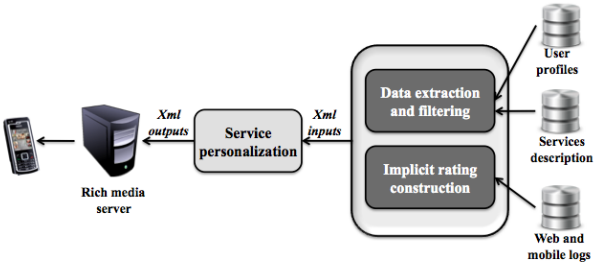


Figure 1: Global architecture of SoliMobile Project

4. OPEN-SOURCE RECOMMENDER SYSTEMS

The growth of Web content and the expansion of e-commerce has deeply increased the interest on recommender systems. This fact has led to the development of some open source projects in the area. Among the recommender systems algorithms available in the Web, we can distinguish the following: Duine [5], Apache Mahout [9], OpenSlopeOne [11], Cofi [2], SUGGEST [13] and Vogoo [14]. All of these projects offer collaborative-filtering implementations, in different programming languages.

The Duine Framework supplies also a hybrid implementation. It is a Java software that presents the content-based and collaborative filtering in a switching engine: it dynamically switches between each prediction given the current

state of the data. For example if there aren't many ratings available, it uses the content-based approach, and switches to the collaborative when the scenario changes. It also presents an Explanation API, which can be used to create user-friendly recommendations and a demo application, with a Java Client example.

Mahout constitutes a Java framework in the data mining area. It has incorporated the Taste recommender system, a collaborative engine for personalized recommendations. Vogoo is a PHP framework that implements a collaborative filtering recommender system. It also presents a Slope-One code.

A Java version of the Collaborative Filtering method is implemented in the Cofi library. It was developed by Daniel Lemire [6], the creator of the Slope-One algorithms. There is also a PHP version available in Lemire's webpage. OpenSlopeOne offers a Slope One implementation on PHP that cares about performance.

SUGGEST is a recommendation library made by George Karkys and distributed in a binary format.

Analyzing software in the recommendation area is not a simple task, since it is difficult to define measurement standards. In this work, we propose some criteria of evaluation: types of recommendation implemented by the project, programming language, level of documentation and magnitude of the project.

The documentation was evaluated based on its volume and clarity. It is possible to observe that the volume of documentation presented by Mahout and Duine is remarkably larger than the other systems. Both offer installation and utilization guides and come with a demonstration example. It must be taken into account that OpenSlopeOne and Cofi are smaller projects, and thus, their documentation tend to be smaller. In the Downloads column we have a representation of the magnitude of the project. It is presented the number of times the software, in any version, was downloaded from its source. Although Mahout does not present its number, its very populated mailing lists shows that it is a widely used software.

The two projects that stood out were Apache Mahout and Duine. We installed and tested them in order to verify which one was more applicable to our work. Both of them are based on the Java technology and present a demonstration example with the Movielens data set. The fact that Mahout is a greater project and has multiple machine-learning algorithms made it more interesting to our research. Also, its module structure encouraged us to choose it.

5. APACHE MAHOUT

The Apache Mahout is a solid project in the Data Mining area. It is a framework that features various scalable machine-learning algorithms. It is programmed using the Java language and runs with Maven project manager. In April 2008, it has incorporated the Taste Recommender System, a Java framework for providing personalized recommendations. Besides Taste, it also offers clustering algorithms and a Map Reduce implementation.

Taste is a very consistent and flexible collaborative filtering engine and supports the user-based, item-based and Slope-one recommender systems. It can be easily modified due to its well-structured modules abstractions. The package defines four interfaces: DataModel, UserSimilarity and ItemSimilarity, UserNeighborhood and Recommender.

With these interfaces, it is possible to adapt the framework to read different types of data, personalize the recommendation or even create new recommendation methods.

The User Similarity and Item Similarity abstractions are responsible for calculating the similarity between a pair of users or items. Their function usually returns a value from 0 to 1 indicating the level of resemblance, being 1 the most similar possible.

Trough the DataModel interface is made the access to the data set. It is possible to retrieve and store the data from databases or from filesystems (MySQLJDBCDataModel and FileDataModel respectively). The functions developed in this interface are used by the Similarity abstraction to help computing the similarity.

The main interface in Taste is Recommender. It is responsible for actually making the recommendations to the user by comparing items or by determining users with similar taste (item-based and user-based techniques). The Recommender access the similarity interface and uses its functions to compare a pair of users or items. It then collects the highest similarity values to offer as recommendations.

The UserNeighborhood is an assistant interface that helps to define the neighborhood in the user-based recommendation technique. It is know that, for greater data sets, the item-based technique provides better results. For that, many companies choose to use this approach, such as Amazon [7]. With the Mahout framework, it is not different, the item-based method generally runs faster and provides more accurate recommendation.

In our project, we choose to adapt the Slope One (a type of item-based algorithm) approach to our problem. Here follows a simple Java application example of how to initiate a recommendation with the Slope One technique:

```
1. DataModel model =
          new FileDataModel(new File("data.txt"));
2. Recommender recommender =
          new SlopeOneRecommender(model);
3. Recommender cachingRecommender =
          new CachingRecommender(recommender);
```

The challenge in adapting this approach to our project was the fact that our input data file was available in the XML format, a type not handled by Mahout. It then had to incorporate another file in the DataModel interface. We create a program that deals with the XML input files. To test this new data handler, we used the Movielens data set. A pack with one million ratings was converted to the XML type to be used as example. With this data set and the XMLfile, the running time of the Slope One algorithm takes less than one minute.

6. WEB USAGE MINING FOR RECOMMENDATION

One objective of the SoliMobile project is to develop a recommender system that has to be, as much as possible, the least intrusive. This implies that the system is based only on information that the user can be free to provide (explicit data) and must run properly with alternatives such as implicit data mining. To meet this need, we are studying how to append Web browsing analysis to the recommender system as done in [8]. Web browsing analysis becomes almost necessary for extracting and understanding user behaviors.

	Implementation	Language	Documentation	Downloads
Mahout	Item-based, User-based, Slope One	Java	High	Not available
Duine	User-based, Content Filtering	Java	High	1,113
Cofi	Item-based	Java	Low	Not available
OpenSlopeOne	Slope One	PHP	Low	653
Vogoo	Slope One, Item-based	PHP	Medium	2,128
SUGGEST	Item-based, user-based	C	Medium	Not available

Table 1: Utilisation ratio of each method.

In recent years, Web usage mining has become an important issue in the field of data mining. The term, Web usage mining focuses on predicting and learning the users preferences on the Internet. Generally, the data for Web usage mining are the user interactions on the web, usually residing on Web clients, Web servers, and proxy servers. The aim of Web usage mining is to analyze user behavior through analysis of its interaction with the Web platform. This analysis is particularly focused on all the users clicks where visiting the web application (also known as clickstream analysis). The interest of Web usage mining in our framework is to enrich the input of recommender system with user data extracted from the raw clickstream data, in order to refine the user profiles and behavioral patterns. The analysis of Web logs can also be used as implicit feedback of the user which will allow to assess the performance of models involved in the recommender system.

It is obvious that Web logs change over time for several reasons: an update of the Web application content or structure, a change in the user preferences, a change in the execution context, etc. This is why it is important to take into account the temporal dimension in the analysis of Web usage. To consider the temporal data in a dynamic way, we plan to use the techniques of data streams mining. By definition, data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. It is impossible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety. Therefore, all the treatment have to be applied in one pass. Several techniques for mining data streams have emerged as CluStream for clustering, StreamSamp for sampling, VFDT for incremental decision trees, etc. The reader may refer to [4] for more explanations on these different techniques.

7. CONCLUSIONS

In this paper, we presented the problem that we deal with in the SoliMobile project. Then, we presented the global architecture that is under development in this project. This architecture includes a recommender system to customize the services offered to users based on their profile and their browsing history. Given the limited duration of the project, we opted for an open source recommender system that is modular in order to easily integrate future developments, in particular the use of Web usage mining to address the problem of cold start.

In this paper, we also discussed several points concerning the issue of treatment of the temporal dimension in data analysis. The raised issues demonstrate the need for defining new methods or adapting existing methods for extracting knowledge and monitoring changing and evolutive data. Although there are many efficient methods for extracting knowledge, few studies have been devoted to the issue of temporary evolutive data.

8. REFERENCES

[1] John S. Breese, John S. Breese, David Heckerman, and Carl" Kadie. Empirical analysis of predictive algorithms for collaborative filtering. pages 43–52, 1998.

[2] Cofi. http://www.nongnu.org/cofi/.

[3] Antoine Cornuéjols. Getting order independence in incremental learning. In *ECML '93: Proceedings of the European Conference on Machine Learning*, pages 196–212, London, UK, 1993. Springer-Verlag.

[4] Baptiste Csernel, Fabrice Clerot, and Georges Hébrail. Streamsamp: Datastream clustering over tilted windows through sampling. *ECML PKDD 2006: the International Workshop on Knowledge Discovery from Data Streams (IWKDDS-2006)*, 2006.

[5] Duine. http://www.duineframework.org/.

[6] Daniel Lemire and Anna Maclachlan ". Slope one predictors for online rating-based collaborative filtering. 2005.

[7] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[8] Jiahui Liu, Peter Dolan, and Elin Rønby Pedersen. Personalized news recommendation based on click behavior. In *IUI '10: Proceeding of the 14th international conference on Intelligent user interfaces*, pages 31–40, New York, NY, USA, 2010. ACM.

[9] Mahout. http://mahout.apache.org/.

[10] Raymond J. Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. In *DL '00: Proceedings of the fifth ACM conference on Digital libraries*, pages 195–204, New York, NY, USA, 2000. ACM.

[11] OpenSlopeOne. http://code.google.com/p/openslopeone/.

[12] ProximaMobile. http://www.proximamobile.fr/.

[13] Suggest. http://glaros.dtc.umn.edu/gkhome/suggest/overview.

[14] Vogoo. http://www.vogoo-api.com/.