Context-Aware Recommendations in Decentralized, Item-Based Collaborative Filtering on Mobile Devices

Wolfgang Woerndl[1], Henrik Muehe[1], Stefan Rothlehner[2], and Korbinian Moegele[1]

[1] Technische Universitaet Muenchen, 85748 Garching, Germany
{woerndl,muehe,moegele}@in.tum.de
[2] EXO Exhibition Overview, c/o Lehrstuhl fuer Baurealisierung und Bauinformatik, 80333 Muenchen, Germany
rothlehn@in.tum.de

Abstract. The goal of the work presented in this paper is to design a context-aware recommender system for mobile devices. The approach is based on decentralized, item-based collaborative filtering on Personal Digital Assistants (PDAs). The already implemented system exchanges rating vectors among PDAs, computes local matrices of item similarity and utilizes them to generate recommendations. We then explain how to contextualize this recommender system according to the current time and position of the user. The idea is to use a weighted combination of the collaborative filtering score with a context score function. We are currently working on applying this approach in real world scenarios.

Keywords: collaborative filtering, context, mobile guides, item-based collaborative filtering.

1 Introduction

Recommender systems are an established research area and have also been successfully introduced in commercial applications such as online shops. So far, most recommenders are centralized systems where all the information about users and items is stored on a server. Decentralized management and interpretation of data without a central server offers some advantages though. Decentralized recommender systems can be used without a permanent connection to a server and thus improve the portability of the system [7]. Recommendations can be made locally by considering other users in the current vicinity. In addition, decentralized recommender systems are regarded as a way of reducing privacy concerns and improving user control over their personal data such as ratings [6]. This is especially true in a mobile or ubiquitous environment. In addition, recommender systems used on mobile devices can be much more dependent on context, such as the current location of the user, than recommenders mainly used in desktop computer settings.

S. Ystad et al. (Eds.): MobiCASE 2009, LNICST 35, pp. 383–392, 2010.

We have designed and implemented a system to recommend items like images on Personal Digital Assistants (PDAs) [9]. Our approach is an application of item-based collaborative filtering. The system exchanges rating vectors among PDAs, computes local matrices of item similarity and utilizes them to generate recommendations. Our innovation in comparison to existing systems includes improving the extensibility of the data model by introducing versioned rating vectors. The goal of the work presented in this paper is to extend this decentralized PDA recommender for context-aware recommendations in mobile scenarios. Context characterizes the situation of users [4]. In our case, context mostly means the physical location of a user when requesting a recommendation, or the current time. But the approach can be generalized to other context attributes such as temperature or the presence of other users in the current vicinity. The benefit of this model is that contextualized recommendations can be given in decentralized manner on a PDA.

The remainder of the paper is organized as follows. In Section 2 we provide some background on item-based collaborative filtering and an existing related approach for a decentralized recommender. Based on this existing approach, we explain our decentralized, item-based recommender system for PDAs in Section 3. Subsequently, we discuss the options for contextualization and then our applications scenarios. In Section 6 we briefly review related work and finally conclude with a short summary.

2 Background

2.1 Item-Based Collaborative Filtering

Recommender systems recommend items like products or services to an active user. Collaborative filtering (CF) utilizes the user-item matrix of ratings users have made to generate suggestions. The standard, user-based CF algorithm first determines a neighbourhood of similar users, i.e. users that have rated similar to the active user in the past. The algorithm then selects (new) items, which users in the neighborhood of the active user have preferred. The second alternative method of CF is item-based collaborative filtering.

Item-based CF does not consider the similarity of users, but of items [8]. Thus, the user-item matrix of ratings is not analyzed line by line, but column by column. The item similarities can be precomputed to build an item-item matrix. The item-item matrix is the model of the algorithm. Therefore, this type of recommendation algorithm is also called model-based collaborative filtering. One element of the item-item matrix expresses the similarity between two items, determined from the users' ratings. The rating vector of the active user is then used to recommend items that are similar to items that have been positively rated by the active user in the past.

Item-based CF has an advantage over user-based CF with regard to the complexity of the computation because the item-item matrix can be calculated as an intermediate result, independently from who the active user is. In addition,

at the time of the recommendation, the algorithm also needs the rating vector of the active user only, not all ratings. This is promising for improving the privacy of personal user data. Therefore, item-based CF appears to be well suited for an implementation on PDAs.

2.2 PocketLens

In this subsection, we are explaining the PocketLens system because it serves as a foundation for our approach. The basic motivation behind the development of PocketLens is that characteristics of item-based recommenders are well suited for decentralized adoption [7]. Though some problems arise from utilizing the item-item matrix.

In particular, it is required to recompute the matrix when a new rating is inserted. The algorithm has to evaluate all ratings associated with the concerning items in this process, because only the computed similarity is stored in the matrix. This also raises privacy issues, because every peer in the system has to store the rating vectors of all users. In addition, the recalculation of the matrix every time a new rating is added is not very efficient and increases hardware demands. It is preferable to make use of the item similarity values calculated so far as intermediate results.

PocketLens modifies the basic item-based CF algorithm to compute the item-item matrix. Thereby, all the concerning elements of the matrix do not have to be completely recalculated. This is achieved by storing the intermediate results with every item similarity pair in the item-item matrix. PocketLens uses cosine similarity and stores the dot products and the lengths of the rating vectors for every item pair. Thus, it is possible to easily update the item-item matrix with new ratings: the new rating just has to be added to the dot products and the vector length. The whole data model does not have to be recomputed. Especially reapplying all rating vectors used so far is not necessary. Hence, the PocketLens algorithm allows to delete the rating vector of other users after integrating their ratings into the model.

3 A Decentralized Recommender System for PDAs

We have implemented and tested a decentralized, item-based CF approach for PDAs based on PocketLens [9]. The idea is that users rate items on mobile devices. Then rating vectors are exchanged with other peers and each PDA calculates the item-item matrix based on the received ratings. The system recommends items based on the model and the local user's ratings only. Our approach improves some problems of PocketLens that have become clear when applying the algorithm in our scenario.

While item-based CF does have advantages with regard to computational complexity to generate recommendation, storing the model – the item-item matrix – does demand more storage capacity in comparison to user-based CF.

Rating vector **Versioned rating vector**

Item	Rating
A	4
C	1

Item	Versioned ratings
A	$(1,5) \leftarrow \underline{(4,4)}$
B	$(2,4) \leftarrow \underline{(5,0)}$
C	$\underline{(3,1)}$

Fig. 1. Rating vector vs. versioned rating vector

Therefore, we have optimized the storage model for usage on PDAs. PocketLens does allow for the easy insertion of new users with their rating vectors into the model, but it does not account for updating the matrix with new or changed ratings from existing users. In a distributed scenario, though, users may frequently exchange ratings and add other users updated rating vectors to their model on the PDA.

To fulfill the second requirement of extensibility, we have extended the concept of a rating vector to reproduce older values. The rating vector consists of an item identifier and the rating value. Our new rating vector introduces a serial respective version number and the corresponding value for representing the rating of an item. We call this resulting data structure a versioned rating vector (Fig. 1). The version-rating tuples of an item are implemented as linked lists starting from the newest rating to allow efficient access.

A change of a rating results in a new entry in the values list of the corresponding item. The history of old values is preserved. When integrating a versioned rating vector into the model, the system saves the version of the rating vector. The version of the vector is defined as the highest version number of any rating's value. When updating the item-item matrix with a vector with new or additional ratings, the system can first undo the effects of the formerly used version of this vector and thus restore the old model. Then the new ratings can be integrated into the matrix just like a new rating vector [9]. The rating vectors of other users can be deleted after insertion into the model, it is still not necessary to store the rating vectors of each user on every PDA.

In the example in Figure 1, item A has received a new rating (4) with a higher version number and therefore an old rating (1) is overwritten but not lost. The rating for item B has been deleted, while the rating for item C has not been changed.

As far as the storage optimization is concerned, we have analyzed the properties of the algorithm and found many duplicate entries in the item-item matrix. That means the computed item similarity value was identical for many item pairs. The elements of the matrix in the PocketLens algorithm require at least 20 bytes of memory in a densely populated matrix. Replacing an element with a pointer (4 bytes) to a duplicate entry reduces the storage requirement for this matrix element by 16 bytes. We have evaluated this approach with the 'MovieLens' data set [5] containing 100000 ratings by 943 users for 1682 movies. We were able to reduce the storage requirement from about 45 megabytes to 32 megabytes.

Fig. 2. Rating and recommending images on a PDA

Besides the reduction by using pointers, we can additionally modify the definition of identical matrix elements. PocketLens uses float values for item similarity and vector size as matrix elements. If we assume that two elements are duplicates if they are very similar but not quite identical, we can increase the number of 'duplicates' found and thus further improve the storage efficiency. The resulting model quality suffers a little, but we have found that cutting the float value after some decimal points does indeed reduce the memory requirements while hurting recommendation quality only marginally if at all. It is possible to adapt this impreciseness of duplicate search dynamically to be able to handle much larger models than without this feature. If we compare the field elements of the item-item matrix with an inaccuracy of just 0.05, the data model can be further reduced to 20 megabytes. This is small enough to be completely loaded into the memory of our test PDAs.

The search for duplicates is done only when a memory threshold is exceeded. The insertion of a rating vector into the model took about 1.5 seconds on average on our PDAs. The search for duplicates needs the main share of this time, which can be done in the background. The algorithm needs only a fraction of the 1.5 seconds for actually inserting a rating vector into the item-item matrix.

The system has been implemented to recommend images as an application example on PDAs for the Windows Mobile operating system [9]. The system displays one image with the option to rate it on a scale from one to five stars (Fig. 2). The user can choose whether the system displays random or recommended images. Random images are used to let the user initially rate a couple of items. It is also possible to change the rating of an image, if it has been rated before. Afterwards, the user gets recommendations based on the algorithm explained above. The user can click on 'Weiter' (German for 'next') for the next — random or recommended — image.

The image recommender has been tested with 13 users and HTC P3600 PDA or similar mobile devices. The test also included a shared public display for group

recommendations [9]. The item set consisted of 63 images. The users were told to initially rate about 15-20 randomly selected images on their PDAs. The corresponding rating vectors were subsequently exchanged, item-item matrices were calculated on the PDAs and users were able to generate recommendations on their devices. First of all, the test proved that our application ran smoothly and provided comprehensive recommendations in the explained scenario on PDAs. Both the computation of the item-item matrices and the generation of recommendations ran without performance problems. Users tested the generated models by judging the recommended images and gave positive feedback in a questionnaire [9].

4 Context-Aware Recommendations

In this section we will now explain the options we are currently working on to contextualize the existing PDA recommender that was explained in the last section. We are currently fine-tuning and implementing the approach in the application domains that follow in Section 5 of this paper.

4.1 Context Definition

First, it is important to define what context means in our scenario. We follow the context definition by Dey et al.: 'Context is any information that can be used to characterize the situation of entities (i.e. whether a person, place or subject) that are considered relevant to the interaction between a user and an application, including the user and the application themselves' [4]. That means, context is very dynamic and transient. By contrast, user profile information such as preferences or ratings are somewhat static and longerlasting. In a mobile application domain, a context model could include location, movement, lighting condition or current availability of network bandwidth of mobile devices, for example.

The assumption is that the generation of the model, i.e. the construction of the item-item matrix, is rather independent from context because the ratings express the preferences of the user. Nonetheless, to contextualize the construction of the item-item matrix, the idea is to compare the context attributs when a rating was made and use this information when calculating item similariry, i.e. item pairs that were rated similarly in a comparable context receive the highest values in the model.

In our mobile scenario, context is important when a recommendation is made, because a user wants a recommendation at a certain time in a certain location. To do so, a context snapshot is taken and used to recommend items that are best suited for the current context in combination with the model of item-based CF.

4.2 Context-Aware, Hybrid Item Recommendation

To generate recommendations, the algorithm calculates the predicted rating of candidate items based on the model and the rating vector of the active user. We will call this predicted rating the *cf-score*. The list of k items with the highest scores

can be then shown to the user. For the image recommender (cf. Section 3 and Fig. 2), the system shows the image with then highest score of all non-rated images.

Our idea is to use the following linear combination of the *cf-score* with a 'context-score' (*ctx-score*) to contextualize the recommendations:

$$score = a * cf\text{-}score + b * ctx\text{-}score \quad (1)$$

a and b are the weights to balance the collaborative and context components. Different application scenarios may require different weights, examples are given in the next section (cf. Section 5). *cf-score* or *ctx-score* returns values in the range [-1 ... 1], so these functions have to be normalized according to these values. Items with negative scores will not be considered for recommendation. A *cf-score* or *ctx-score* of zero means that the item is either evaluated as moderate, or the algorithm has been unable to generate a score, for example when an item has not yet been rated. The linear combination of the scores was designed to allow for separate calculation of the two components and weigh them according to different requirements (see Section 5)).

In terms of hybrid recommender systems [3], this approach is a weighted hybrid recommender. Other options are possible, including a cascading hybrid recommender. In this case, an initial recommender would generate some intermediary results using only the available ratings. In a second step, the results are further filtered and ranked by considering the 3rd dimension of context. However this contextual pre- or postfiltering [1] can also be implemented with our weighted hybrid method.

4.3 Context Score Function

Context is modeled by a vector C whose elements represents the various context attributs $c_1, c_2, \ldots, c_3$. For example, (c_1, c_2) = (longitude, latitude) to model the geographical position using GPS coordinates or (c_3) = current time. Different attributes can model very different things. All items have associated context attributes, e.g. the location and opening times of a restaurant. That is similar to item attributes that characterize the item space in content- or case-based filtering, e.g. price, color, weight etc. of a product.

The goal of the context score function is to model the similarity of two context vectors, one representing the current context of the active user and the other one an item under consideration. What similarity constitutes depends on the considered attributes and the application domain. The context score function returns a normalized value with -1 representing the worst value and +1 the best value, i.e. the user context is identical to an item context, according to the model.

The context score function can also be used to rule out items that are not suitable at all in the current context. For example, if the user want a recommendation for a restaurant at a given time, restaurants that are closed should not be included in the list of recommended restaurants:

$$ctx - score = \begin{cases} +1, & \text{if restaurant is open} \\ -1, & \text{if restaurant is currently closed} \end{cases} \quad (2)$$

In this example, the context score function will return -1 and the overall score of the item will be negative if the restaurant is closed. The same can be done for location. Thereby some items will be reasonable close to the current user's context, so the context score function for two context vectors C1 and C2 will be a variant if the following in most cases:

$$ctx - score = \begin{cases} +1, & \text{C1 and C2 are (nearly) identical} \\ x, & \text{x = distance(C1, C2)} \\ -1, & \text{C1 and C2 are (too) different} \end{cases} \quad (3)$$

distance(C1, C2) is normalized to [-1 ... +1], with zero representing a mediocre value. A combined context score function to integrate different context attributes is also possible.

5 Application Scenarios

In this section, we explain two real world application scenarios we are currently working with start-up companies in two separate projects.

5.1 Mobile Tourist Guide

The first one is a mobile tourist guide developed by voxcity s.r.o. and jomedia s.r.o. The idea is to rent out a mobile device with GPS positioning capabilities to support tourists. The guide is currently available for the Czech city of Prague (see www.voxcity.de). The mobile application plays audio, video, pictures and (HTML) text of tourist attractions based on the current position, traveling direction and speed.

The plan is to extend this application with options to rate items (i.e. the multimedia files) and generate recommendations not only based on location but also the collaborative filtering model. Since the mobile devices are not permanently connected to a network in this scenario, our decentralized approach appears well suited. When a customer rents a device, it will have the model of item similarity according to past ratings on it. The tourists can use the guide and also rate items such as audio clips. These ratings can then be used to generate CF recommendations as described above. In this case, the context score function will have a higher weight, because we still want to recommend multimedia files that are related to attractions in the current vicinity. On the other hand, the CF recommendation process can point the user to items (in our case, tourist attractions) that are a little farther away but are recommended according to the CF model. When a user returns the mobile device, the model can be updated with new ratings on other devices and used for other users. One additional idea is to acquire implicit ratings according to observed behaviour when using the device, e.g. a user playing an entire audio clip instead of skipping over it would constitute a positive rating.

5.2 Mobile Exhibition Guide

The second application scenario is a navigation and information system that is being developed by EXO Exhibition Overview GbR. It allows the visitors of trade fairs, conferences and similar events to orient themselves at the venue. Visitors of exhibitions can search for products, exhibitors or places of interest and navigate there. Moreover, the visitor can use several additional functions on the device, such appointment schedule, virtual business cards or collecting electronic versions of available print media on the event. The indoor positioning is done by a bluetooth-based infrastructure, which is also used to transfer data between the mobile devices and a central data source. The system will be available at the Munich Trade Fair Center starting 2010.

In this scenario the context, e.g. the exact location of exhibition booths, is less important. Thus, the context score function will distinguish between exhibition halls and prefer items that are nearby when recommending, but not completely rule out any item. The collaborative part, i.e. the *cf-score* function, is more significant in this scenario. It allows the device recommend items might not have been aware of before, but which fit her previous rating profile.

On the other hand, for recommending exhibition events (e.g. talks), the temporal context is very important. This can be modeled in our approach using the context score function. In this case, the context score will return -1 for past events or expired information, and a high score for upcoming events.

6 Related Work

In addition to the already discussed PocketLens, there are a few related approaches with regard to decentralized CF. For example, Berkovsky et.al. propose a distributed recommender system that partitions the user-item matrix based on domain-specific item categories [11]. However, additional information about the application domain of the items is needed.

One fundamental solution to contexualize recommender systems is the multidimensional approach by Adomavicius et.al. [2]. This model enhances the user-item data model to a multidimensional setting where additional dimensions constitute different context attributes such as location and time. They propose a reduction-based approach with the goal to reduce the dimensions, ideally to $n = 2$. Then, traditional recommender techniques can be used to generate item lists. But to our knowledge, the approach has not been applied in a mobile scenario setting similar to ours.

Magitti is a recent system to recommend leisure activities on mobile devices [10]. Their approach is to combine and weigh very different recommenders such as CF, a distance model, content preference and others. However, they do not use item-based CF, which we believe is well suited for application on a mobile device.

7 Conclusion

The goal of this work is to design a context-aware recommender system for mobile devices in the explained application scenarios. The approach is based on decentralized, item-based collaborative filtering. Thereby, mobile devices exchange rating vectors of their respective users, calculate local matrices of item similarity and utilize them to generate recommendations. We are currently working on contextualizing the recommender by using a weighted combination of the collaborative filtering score with a context score function. It is important to note that in our solution, the context information about a user never leaves her personal mobile device, so the approach can provide context-aware recommendations without raising issues of location privacy, i.e. a server being able to track a user's movements.

References

1. Adomavicius, G., Tuzhilin, A.: Context-aware recommender systems. In: Tutorial at ACM Conference on Recommender Systems, RecSys 2008 (2008), http://ids.csom.umn.edu/faculty/gedas/talks/RecSys2008-tutorial.pdf
2. Adomavicius, G., Sankaranarayanan, R., Sen, S., Tuzhilin, A.: Incorporating contextual information in recommender systems using a multidimensional approach. ACM Transactions on Information Systems 23, 103–145 (2005)
3. Burke, R.: Hybrid web recommender systems. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) Adaptive Web 2007. LNCS, vol. 4321, pp. 377–408. Springer, Heidelberg (2007)
4. Dey, K., Abowd, D., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human Computer Interaction 16, 97–166 (2001)
5. Herlocker, J.L.: An algorithmic framework for performing collaborative filtering. In: 22nd Annual international ACM SIGIR Conference on Research and Development in Information Retrieval, Berkeley, CA (1999)
6. Kobsa, A.: Privacy-enhanced personalization. Communications of the ACM 50(8), 24–33 (2007)
7. Miller, B.N., Konstan, J.A., Riedl, J.T.: PocketLens: Toward a personal recommender system. ACM Transactions on Information Systems 22(3), 437–476 (2004)
8. Sarwar, B., Karypis, G., Konstan, J.A., Riedl, J.T.: Item-based collaborative filtering recommendation algorithms. In: 10th International Conference on World Wide Web (WWW 10), Hong Kong, China (2001)
9. Woerndl, W., Muehe, H., Prinz, V.: Decentral item-based collaborative filtering for recommending images on mobile devices. In: Workshop on Mobile Media Retrieval (MMR 2009), MDM 2009 Conference, Taipeh, Taiwan (2009)
10. Ducheneaut, N., Partridge, K., Huang, Q., Price, B., Roberts, M., Chi, E.H., Bellotti, V., Begole, B.: Collaborative filtering is not enough? Experiments with a mixed-model recommender for leisure activities. In: Houben, G.-J., McCalla, G., Pianesi, F., Zancanaro, M. (eds.) UMAP 2009. LNCS, vol. 5535, pp. 295–306. Springer, Heidelberg (2009)
11. Berkovsky, S., Kuflik, T., Ricci, F.: Distributed collaborative filtering with domain specialization. In: ACM Conference on Recommender Systems (RecSys 2007), Minneapolis, MN (2007)