A Hybrid Recommender System for Context-aware Recommendations of Mobile Applications

Wolfgang Woerndl\textsuperscript{1}, Christian Schueller\textsuperscript{2}, Rolf Wojtech\textsuperscript{1}

\textsuperscript{1} Technische Universitaet Muenchen, Institut fuer Informatik
\{woerndl,wojtech\}@in.tum.de
\textsuperscript{2} UnternehmerTUM GmbH
schueller@unternehmertum.de

Abstract

The goal of the work in this paper is towards the incorporation of context in recommender systems in the domain of mobile applications. The approach recommends mobile applications to users based on what other users have installed in a similar context. The idea is to apply a hybrid recommender system to deal with the added complexity of context. We have designed and realized the application to test our ideas. Users can select among several content-based or collaborative filtering components, including a rule-based module using information on point-of-interests in the vicinity of the user, and a component for the integration of traditional collaborative filtering. The implementation is integrated in a framework supporting the development and deployment of mobile services.

1. Introduction

The sheer volume of data that exists in the internet today makes it difficult for users to find and access relevant information. Personalization of content is a technique to reduce the omnipresent information overload. Thereby, information is customized according to users’ needs and preferences. Often, recommender systems using collaborative or content-based filtering are applied. In a mobile scenario, information personalization is even more important, because of the limitations of mobile devices regarding displays, input capabilities, bandwidth etc.

For example, a traveling user with a PDA, smartphone or pocket PC needs access to a train timetable, the weather report at her destination or a recommendation for a restaurant in her vicinity. It is desirable to personalize not only using pre-defined user profiles, but also her context such as the current location. According to Dey et.al., context can be defined as “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” [3].

However, context has been rarely incorporated into personalization research so far [1]. The ultimate goal of our research is to investigate how the integration of context into personalization systems can be realized in the domain of mobile applications. Personalization in our scenario implies the selection of services (a mobile restaurant guide, for example) as well as customization of the services (a particular restaurant in the vicinity of the user, for example). In this paper, we describe a system that recommends mobile applications based on a hybrid recommender system and utilizing the context, basically the current location, of the user. Our recommender system is integrated in a framework supporting the development of mobile applications.

The rest of the paper is organized as follows. First we introduce our “play.tools” framework for mobile applications. In section 3 we discuss how to integrate context into recommender systems from a more theoretical perspective. Then, we describe the design and implementation of our context-aware recommender system, including the user interface. In sections 5 and 6, we finally discuss related work and conclude our paper.
2. Background

2.1. Developing mobile applications

More and more humans cannot imagine a life without a mobile phone, whether privately or vocationally. These phones are not only used for voice transmission, but also mobile applications like location-based services or m-commerce applications are deployed at an increasing rate. The development of marketable and innovative mobile applications represents a huge challenge for software developers. The causes for this are to find, among other things in, unknown user/customer needs, insufficient standards, incorrect programming interfaces and extremely short product life cycles at mobile devices and the corresponding software [9]. Particularly the access to system near functions of mobile devices, the positioning and the development of intuitively operated user interfaces are still very complex, because up to now, appropriate software frameworks are missing, which encapsulate these functionalities suitably.

Therefore, the framework play.tools was created to reduce the complexity during prototypic implementation and subsequent market tests for context-sensitive mobile applications. Thereby, the substantial generic components of a context-sensitive mobile application (user interface, position determination, supply of geographical information, persistence, etc.) are totally enclosed in a modular architecture. The developers do not have to worry about technical details of the generic function variety of context-sensitive mobile applications. They can completely concentrate on the development of the software components.

A further problem of mobile applications is the downstream acquisition of feedback by market tests. For that purpose, a distribution platform with a deployment server was developed. There, interested users can find relevant applications more easily.

2.2. The play.tools framework

Figure 1 depicts a high level overview of the distribution platform and the interaction of the main components in our framework. Application programmers design and implement client modules and server applications (services) using the provided generic modules. These applications may communicate with other services and/or use a database to store items. An administrator then registers her application with the deployment server (step “1” in Fig. 1). Users can access the deployment server and download client modules on their mobile devices. Typically, the client program interacts with the server to provide a service, e.g. a mobile tourist guide. Information about users (profiles) is kept separately from the services, in order that personal information can be reused for different applications and is kept under the control of the user (privacy).

3. Context in recommender systems

Before describing the design and implementation of our recommender system in section 4, we discuss the incorporation of context into recommender systems more generally.

3.1. Recommender data model and context

A recommender system tries to predict the relevance of information items for a user. This is based on information about the user, meta data associated with items and/or implicit or explicit ratings for items made by a group of users. The traditional model can be summarized as follows [1]:

Given is a set of \( n \) items \( I = \{i_j : 1 \leq j \leq n\} \), and a set of \( m \) users \( U = \{u_k : 1 \leq k \leq m\} \). Each user may be described by a \( t \)-dimensional vector of attributes, which is also called user profile. The items can have associated meta data, in form of a \( s \)-dimensional vector...
(item description). The goal of the personalization process is to recommend items \( i \) to an active user \( u_a \) that would be of interest to the user.

Thereby, a content-based algorithm matches the meta data of items with the user profile of the user \( k \) without taking other users into account. The collaborative filtering (CF) approach relies on ratings of items by users, for example a grade how much a particular user likes a given product. A rating \( r_{j,k} \) formally is an element in the 2-dimensional item-user matrix. A CF recommender is based on finding similar users, based on their previous ratings. A variety of different algorithms for content-based or collaborative filtering have been proposed [1].

The idea is to introduce context into the recommendation process. In the example of the introduction, a personalization system could recommend a train timetable to a travelling user by matching the user context (“inside/near train station”) with an item description. This could be done using a content- or knowledge-based approach. But also collaborative filtering could be applied. For example, if several users have downloaded mobile games while waiting for a train, a context-based collaborative filtering system would recommend games to another user, even if “games” is not related to “train station”.

The first question is now, how to incorporate context into the data model of recommender systems and collaborative filtering. Context should not be included in the user profile because it is very dynamic. A rating made in one context may not be valid in another. On the other hand, user profile information is more static, for example the preferred language of the user. So far, CF has mostly been applied for applications for which the context is rather static, hence the recommendations do not change much over time [2]. Depending on the application area, user models can also be highly dynamic, but from our point of view a user profile contains information such as preferences, interests or knowledge (e.g. the user’s favourite type of restaurant), whereas the context model describes the current environment in which the user operates in (e.g. her current location). Therefore, we distinguish between user profile and context attributes.

Our approach for the integration of context into recommender systems is to associate each rating \( r_{j,k} \) for an item \( i \) by a user \( k \) with a context description. In other words, context adds another dimension to the item-user matrix of CF (Figure 2) [5]. A rating \( R \) is then a tuple of (item, user, context). The ratings are stored as part of the user profiles in our framework (Fig. 1).

![Figure 2. Introducing context in CF](image)

### 3.2. Context modelling and acquisition

Context is described as a vector of context attributes, analogous to users and items. Context attributes are the current location and mobile device when a user made a rating, for example. The similarity or identity of context values is not as clear as with users or items and may depend on the actual application domain. For example, for a tourist guide the “same” context might be constituted by a range of GPS coordinates.

At this time, we primarily use the current location of the user as context attribute. To retrieve the location, we are using GPS-enabled devices, e.g. a pocket PC, or mobile phones with an external GPS receiver, connected via a bluetooth interface. Our play.tools framework provides methods for retrieving and analyzing GPS coordinates on client and server.

We are also working on how to acquire additional context in a mobile scenario including generating high level context from sensor data. Sensors are thereby embedded into mobile devices. For example, we are experimenting with Bosch sensors (see http://www.bosch-sensortec.com) to retrieve and analyze acceleration and pressure\(^1\). The mobile device the user is currently using is another source of context (e.g. display size). Software running on the end user device could also be used, e.g. calendar or address book entries.

UserML is an example for an existing decentralized user model language [13]. UserML (or GUMO, the General User Model Ontology) can be used to exchange user model information between

\(^{1}\) A more thorough discussion of context acquisition is out of the scope of this paper.
applications. UserML distinguishes between the so-called “mainpart” attributes (for example, a statement about a user’s interest in football), and attributes describing the “situation” of the user (what we call context attribute). Our approach is rather independent from how the user profile and context attributes are actually modelled.

3.3. Context-aware collaborative filtering

How to utilize context in recommender systems? Annie Chen has proposed a solution to introduce context into collaborative filtering [2]. She tries to determine which ratings are more relevant for a given context by computing the similarity of context. This is difficult, since context is very heterogeneous, i.e., different types of context attributes exist. Therefore, A. Chen makes the assumption that if two user ratings for an item are similar for two different context values, then the two context values are relevant to each other. She then calculates the correlation between two context values using Pearson’s correlation coefficient. When generating a prediction, the context similarity determines how relevant ratings for a given context and user are.

One drawback of this approach is that it is necessary to compute the similarity of contexts which may be difficult in most application domains. However, some kind of context aggregation is necessary. In addition, a large amount of data may be needed to generate meaningful results. Collaborative filtering often suffers from a sparsely populated item-user matrix and the „ramp-up“ problem. That means, it is difficult to compute ratings for new users or items (or contexts). UbiMate (http://ubimate.hopto.org/) is a mobile city guide to generate context-aware recommendations and collect data for Annie Chen’s approach.

The ramp-up problem could be solved by applying a content-based filtering approach. Thereby, the description vectors of items are compared with the profile of a user and her current context. An advantage is that the ramp-up problem is eased. The recommendation process can better include new users or items, i.e. mobile applications. On the other hand, additional information has to be provided, for example information for which context an item is recommendable. Another idea, which we realized in our solution, is to combine different algorithms in a hybrid recommender system.

3.4. A hybrid recommender system for mobile applications

A hybrid recommender system combines different recommender systems to improve information retrieval. The combination can be done using several alternatives, for example: weighted, switching, mixed, feature combination or augmentation or cascading [4].

Our idea is to combine collaborative filtering with other recommenders to deal with the added complexity of context (Fig. 2). From a conceptional point of view, this means reducing the complexity of the item-user-context matrix by applying a cascading hybrid recommender. That means, first only two dimensions of the matrix (Fig. 2) are analyzed, and in a second step the third dimension is considered in addition. In more detail:

1. Use content- or knowledge based filtering to find relevant items based on context, for example taking the current end user device and location into account
2. Apply collaborative filtering to rank and additionally filter the result set from step 1

This can be done vice versa, i.e. first using collaborative filtering to generate initial items – without considering context –, and then using the knowledge base of applications to figure out which items resp. mobile application are actually relevant in the current context. An obvious example is to recommend applications, which work on the current end user device.

We decided to realize different recommender algorithms. E-Commerce sites such as Amazon often display the results of several collaborative or content-based recommenders on one product page. We follow this idea in our approach (next section) and provide different recommenders to the user. An advantage is that users get an explanation about the recommendation process because they select a particular algorithm.

4. System design, implementation and test

We have designed and realized a hybrid recommender system to recommend mobile applications to users [5], on the basis of the considerations in section 3 of this paper. The implementation has been integrated in the play.tools framework that is outlined in section 2.
4.1. Recommender modules

As explained above, the approach includes letting the user decide which of several recommenders one she wants to use. We have implemented the following recommender components:

- **CFAppRecommender**: Apply collaborative filtering to generate results
- **LocationAppRecommender**: This is a “context-item” recommender, applications that we installed in a similar location are recommended
- **PoiAppRecommender**: Knowledge-based approach based on “point-of-interests” in the current location of the user
- **RandomAppRecommender**: Show a random list of applications

The recommender modules are explained in more detail in the next chapter. We distinguish between applications that have to be installed on the device, and others, usually Web based information portals. All recommender modules do not recommend applications that have already been installed or rejected by the user before. If the user installs an application, he thereby makes a positive rating for this item that is used for the collaborative filtering.

4.2. Design and implementation

Our recommender system consists of a thin-client J2ME program and a server application within the play.tools framework. The client realizes the communication with the server and the dialog with the user, the actual recommendations are generated on the server.

The core of the server program is a *ApplicationManagerBean* that manages the mobile applications and propagates the recommender requests to the actual recommender components. The recommender modules, e.g. the **CFAppRecommender**, inherit an interface from a *AbstractAppRecommenderBean*. Using this design, new recommenders can be integrated in our approach very easily. The recommender modules use methods in the *ApplicationManagerBean* to store and retrieve data (data access object pattern), e.g. ratings, using a MySQL database.

Administrators can register their mobile applications in the system using a Web interface. Applications have a title and a flag whether this program has to be installed or not. In addition, each mobile application has a description (to be shown to the user, Fig. 6), an URL and a (optional) trigger. The URL points either to the Web/WAP start page, or to a JAR resp. JAD file to install the mobile program on the device.

The **LocationAppRecommender** recommends applications other users liked in the current location before. To do so, the mobile client module communicates its GPS coordinates to the server part. The server application determines the similarity of contexts by applying a configurable algorithm to calculate the distance of the current position to other positions among the stored context ratings. The method selects mobile applications that were installed in the area by other users, not used by the active user before and ranked by the number of positive ratings. Thus, the context is used to select items and, in a second step, the items are ranked taking the ratings of other users into consideration. This recommender can be generalized to a “ContextAppRecommender”, but we have only realized the integration of user location yet.

For the **CFAppRecommender**, we have evaluated several available software libraries for collaborative filtering and chosen the “Taste” engine (http://taste.sourceforge.net). Taste provides a set of components from which one can construct a customized recommender system from a selection of algorithms including the rather novel Slope One algorithm [12]. The **CFAppRecommender** analyzes the item-user data without considering context in the first step, but could filter the results, i.e. recommended mobile applications, according to devices capabilities such as display size in a second step (not implemented yet).

The POI recommender was implemented using an interface to Point-of-Interest services that our play.tools framework provides. Thereby, we query a POI service to retrieve POIs in the vicinity of the user. At this time, we use the ArcWeb web service by ESRI (see http://www.esri.com/software/arcwebservices/index.html). Unfortunately, this service currently has a response time of up to 20 seconds. The **PoiAppRecommender** does not recommend Point-of-Interests but recommends mobile applications based on POIs in the vicinity of the user using triggers.
Figure 3. Configuring a POI trigger

Figure 3 shows the user interface to configure the triggers. The administrator can select types of Point-of-Interests and specify within what circumference of an actual POI an application is recommended. The types in Fig. 3 are some of the return POI type values of the ArcWeb service. The POI types represent the context attributes that are used for this recommender.

The advantage of this recommender is that administrators can specify exactly when an application is suitable (rule-based recommender). On the negative side, the registration of applications requires additional effort.

4.3. User interface

In this chapter we describe the user interface of our application. The screens are intended to be as simple as possible. The user interface is in German only at this time.

A user can start the client program and log in. She then gets a form where she can select one of our four recommenders (Fig. 4). In this screen the user gets a list with a non-technical explanation of the available recommenders. For example, the CFAppRecommender (2. row) reads: “Users with a comparable taste chose …”.

As result of selecting one recommender process, the user receives a list of recommended mobile applications on the next screen (Fig. 5). The list is ranked, i.e. the “best” or most suitable application – according to the used recommendation algorithm – is on top. Then, she can browse the list and choose an application she is interested in.

By selecting one application, the detailed description is shown (Fig. 6) and the user can either install or use the application, or select “Nicht wieder empfehlen” (do not recommend anymore) to express that she does not want to use this application. The user can always click “Zurück” (back) to go back to the previous screen.
The actions of the user are recorded to be used in subsequent recommendations. If the user starts an application, this information is stored as a positive rating for the CFAppRecommender. The ratings include available context information. At this time, we record the GPS coordinates when the application is started. Other than the option to express dislike (Fig. 6) in an application, we do not let users explicitly rate applications, because this would presumably be too cumbersome on a mobile device. But explicit item (application) ratings could be integrated in our system.

4.4. Tests

We have tested our system using 7 users and 6 mobile applications. For example, The PoiAppRecommender has been tested at several locations in Munich, Germany and recommends applications according to POIs in the area (e.g. museums, cinemas) and the configured triggers. The tests have shown that the components of our implementation work as specified and explained in this paper.

We have not done an evaluation with real users so far. At this time, we do not have enough actual end user applications in our system yet. We are currently working to improve and extent the play.tools framework, including refining the recommender system, and also building more applications. We will then conduct a user study to test the effectiveness and usability of our recommender system in a real world scenario.

5. Related work

Besides the approach by Annie Chen [2] which has already been discussed in chapter 3.3., there are a few other context-sensitive recommender systems. An implementation of the algorithm of Chen could possibly be integrated in our approach as recommender module.

CoMeR is a system to support context-aware media recommendations for smart phone [10]. It is similar to our approach because CoMeR also uses a hybrid recommender system to generate recommendations for mobile devices. The hybrid recommender consists of content-based approach to evaluate media items against user preference context, a Naïve Bayes classifier to match with situation context (location) and a rule-based approach to customize media delivery towards device capabilities. However, their approach does not incorporate collaborative filtering. CoMeR is also very much tailored towards media items.

Other related work includes Yap et.al. who present a framework for acquiring and managing context for recommender systems [6]. They also separate the contextual concerns from the actual recommender modules. [7] describes an approach to incorporate context into recommender systems, however in the domain of activation network formalisms. Kruk and Decker introduce an approach for “semantic social collaborative filtering” based on the FOAF standard for describing people [8]. Their focus is more on distributed user profile management and utilization of semantic social networks. [11] proposes a multidimensional approach to deal with context in recommender systems as an extension of the item-user model. The paper introduces a multidimensional rating estimation method to deal with the additional dimensions.

[14] surveys the field of context-aware mobile computing research. A typical application is a conference assistance that helps an attendee organizing her conference schedule based on location, time and schedule. They also give an overview of acquiring context using sensors and modelling the context information for use in personalization applications.

6. Conclusion

The basic idea of our work is to apply a hybrid recommender system to deal with the added complexity of context integration in recommender systems. We have designed and realized an application to recommend mobile applications to test our ideas. Users can select among several content-based or collaborative filtering components. The implementation is integrated in the play.tools
framework supporting the development and deployment of mobile services we are developing.

Future work includes the extension and refinement of the play.tools framework as well as our application recommender. In particular, we are working on the acquisition of more and better context attributes in our mobile domain. So far, we are primarily using location as context. We also are implementing an application to context-aware and personalized syndicate other information items to mobile users, not only recommendations for applications.

7. References


[12] D. Lemire and A. Maclachlan, “Slope One Predictors for Online Rating-Based Collaborative Filtering”, *Proc. SIAM Data Mining (SDM ’05)*, 2005
