Pushing Browser-Based Services to Mobile Users: A Context-Aware Service Recommender for Smart Environments

Henry Song, Hao-hua Chu, Masaji Katagiri

DoCoMo Communications Laboratories USA, Inc.
181 Metro Dr., Ste. 300, San Jose, CA 95110
{csyus, haochu, katagiri}@docomolabs-usa.com

Abstract

Smart environments, in which ubiquitous computing resources can assist users with their real world tasks, have recently attracted growing interest. A problem in smart environments is that the number of services available to users in these environments is limited. Meanwhile, large selection of browser-based services exists on the Internet. However, mobile users in these smart environments who like to access browser-based services on the Internet are burdened with information searching and filtering on mobile devices with small display size and slow, costly connections. In this paper, we describe a *context-aware service recommender* system that recommends *relevant* browser-based services to mobile users in smart environments, and *pushes* those services to users' mobile devices. We formulate a new *<user, context, service>* dataset to represent relevance rating data that are used by the service recommender to make personalized recommendation. We present a new class of context-based collaborative filtering algorithms that recommend services based on this data.

1 Introduction

Recently, there has been a growing interest in building smart environments and pervasive computing systems [3, 4, 9] that can seamlessly integrate our everyday lives with artifacts of computing and communication capabilities in our surrounding environment. In order to provide such seamless user experience, the smart environment must be able to determine the *current user context*: user location, orientation, time of day, etc., and decide on appropriate actions. In existing smart environments, these actions are supported by *context-aware* services or applications that are built specifically for each environment. A smart environment system provides contextual events to the interested context-aware services that may handle these events.

Because context-aware services are built specifically for each smart environment, the number of available services is limited by each environment. However, there is a much wider selection of environment-independent *browser-based services*[1] that exist on the Internet. Users should be able to use these browser-based services in smart environments as easily as local, context-aware services. At the same time, service providers should be able to deploy such global browser-based services that can be used in any smart environments without any environment-dependent customizations.

Because these services are not designed for any particular smart environment, they lack any notion of user context in an environment, and conversely, the smart environment has no notion of the applicability or usefulness of such a service. In particular, *context events* provided by a smart environment are not handled by global browser-based services. For example, in a smart grocery store equipped with sensors, there is no way for a user to incorporate a web-based consumer report service into the activity of shopping. The smart environment does not know to invoke a consumer report search when it senses that a user has picked up an item, and the consumer report service has no other way of activating itself.

[1] We define browser-based services to be web-based services whose main communication channel is HTTP and users interact with them through browsers on client devices.

The ability for services to be found effortlessly, for services to be *pushed* to users, rather than for users to manually search for them, is an important step toward creating a better user experience of using web-based services in smart environments, because use of computing services in everyday tasks must require minimal effort from users. It would be immensely inconvenient for users to search for services while holding a basket of groceries, or while standing in line at the checkout counter, on a mobile device with a small display, over a slow wireless network. Therefore, a push model rather than a pull model is essential in getting services to mobile users.

To address these issues, we have designed a *context-aware service recommender* that can infer triggering conditions for any browser-based services in smart environments, and push personalized recommendations of relevant services to users based on *current user context*.

1.1 Scenario

The following scenario further illustrates the motivation for a context-aware service recommender:

Jane needs to go grocery shopping. Before going to the store, she prepares a shopping list using a shopping list service. She then goes to a supermarket that is a smart environment in which location detection sensors are installed throughout the store, and all price tags on the products contain embedded sensors. She brings a mobile device with her, to access the shopping list and other services, and to use *service recommendation agent* for service recommendation.

The store first detects when Jane enters. This triggers the recommendation agent to notice that the shopping list service is relevant to the environment, and that it can be used together with the store's local map service to help Jane find the locations of her needed items.

Jane follows the map on her mobile device to find a bottle of milk on her shopping list, and picks it up. The embedded sensor in the price tag on the bottle then triggers Jane's agent to recommend two services: a web consumer report service and a local coupon service. She selects the consumer report service first. A consumer report recommending the brand of milk is then displayed on her mobile device. She then selects the coupon service to see if there is a store coupon on this or any other milk. The coupon service does report a store discount on this item, so she puts the milk into her cart and continues shopping.

After completing her shopping, Jane goes to an automatic checkout counter. A scanner at the checkout counter scans items in her shopping cart and computes the total price. This triggers the recommendation agent to recommend the store coupon service and Jane's credit card service. Jane uses the coupon service to receive the store discounts, and her credit card service to pay the bill.

In this scenario, it is clear that service recommenders that push browser-based services to users, and consider context information, can be very helpful in daily tasks.

1.2 Collaborative Filtering

We base our context-aware service recommender on *collaborative filtering*, which is the most successful recommendation technique currently used by e-commerce recommender systems. In these systems, users and items (products) are considered the axes of a two-dimensional matrix, in which each element of the matrix is a rating. A collaborative filtering algorithm fills in empty elements in the matrix with predicted ratings, and the recommender system recommends the items with the highest ratings. We describe this technique here.

Pearson Correlation

Collaborative filtering may use any of several metrics for determining the correlation between two sets, in a population of sets. One successful formula is called *Pearson Correlation*:

$$c_{X,Y} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2} \sqrt{\sum_i (y_i - \bar{y})^2}} \qquad (1)$$

In this formula, x_i and y_i refer to individual, corresponding values in sets X and Y, and $\bar{x}$ and $\bar{y}$ refer to averages of those sets. In e-commerce applications of collaborative filtering, each set typically represents either an individual user, or an individual

item, and the values in each set represent the user's ratings of the items.

There are other similar formulas for determining correlation between two corresponding sets, such as cosine-based correlation [11], mean-squared difference, and Spearman correlation. Without the loss of generality, we use Pearson correlation in the remainder of the paper.

User-Based Collaborative Filtering

User-based collaborative filtering fills in the unrated entries in the user-item matrix by first forming a *k*-nearest user neighborhood of similar users to the active user, and predicting a missing rating based on a weighted average of ratings from the *k*-nearest user neighborhood. The weighting assigned to each neighborhood rating is proportional to the degree of similarity between the active user and the neighbors. One such weighted average prediction, given by Herlocker, et al [7], is as follows:

$$P_{au,ai} = \overline{r_{au}} + \frac{\sum_{u \in U} c_{au,u} \cdot \left(r_{u,ai} - \overline{r_u}\right)}{\sum_{u \in U} c_{au,u}} \qquad (2)$$

In this formula, u is a user in set U of all users, au is the *active* user (the user for whom the formula is filling in item ratings), $r_{u,ai}$ is the rating given to active item ai by a user u, $\overline{r_u}$ is the average rating given to all items by a user, and $c_{au,u}$ is the correlation between two users, au and u, as given by Equation (1).

Item-Based Collaborative Filtering

Item-based collaborative filtering is another method of predicting ratings. It forms a *k*-item neighborhood of similar items and uses active user's ratings on item neighbors to predict ratings on target items. One such prediction is given by Sarwar, et al [11]:

$$P_{au,ai} = \frac{\sum_{i \in I} c_{ai,i} \cdot r_{au,i}}{\sum_{i \in I} \left| c_{ai,i} \right|} \qquad (3)$$

In this formula, i is an item in set I of all items, ai is the active item (the item for which the formula is filling in user ratings), $c_{ai,i}$ is the correlation between two items as given by Equation (1), and the rest of the symbols are the same as above.

1.3 Challenges

Our service recommender is analogous to the product and media recommender systems employed by Amazon.com, Netflix, TiVo, and many others – services are equivalent to items. However, there is one major difference: *user context* is an additional dimension to the matrix that our recommender algorithms must analyze. Note that our user context is not the same as past user behavior (e.g., browsing, purchase, or rating history). Instead, it is about associating each user behavior with a context that can be detected by sensors in smart environments. For example, Amazon.com does not track user context under which a user makes a purchase, e.g., buying a technology book at office, or buying a toy at home. We believe that such user context information can help to improve the quality and relevance of recommendations. Amazon.com could recommend technology-related books when the user were at office, and toy-related products when the user were at home.

We define the dimension of context to be folded <*environment, event*> pairs. That is, every context event (such as entering a store, arriving at a checkout counter, picking up a product) in every smart environment is considered a point of context. Therefore, the matrix we consider can be visualized as in Figure 1.

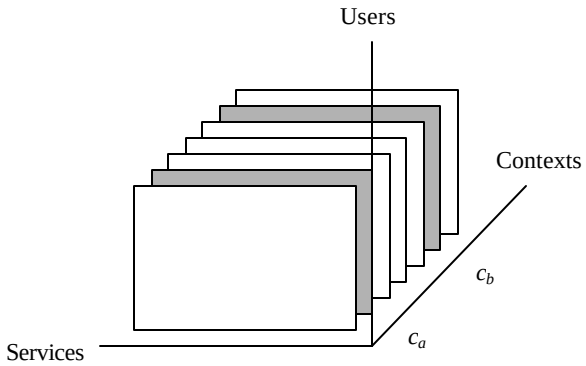

Figure 1: Three-Dimensional <*user, context, service*> Matrix

This additional dimension allows us to explore similar contexts, in addition to similar users and services, to determine predictions on relevant services. For example, if a user is in the context c_a, a service s is commonly invoked in context c_b, and contexts c_a and c_b are closely correlated, and then the service s could be recommended to the user. The basic challenge that this paper addresses is to provide a class of new algorithms with the ability to consider this additional dimension and its semantics.

It has been found that existing recommender systems may have sparse dataset [7]. Accordingly, qualities of recommendations in existing recommender systems that use collaborative filtering algorithms may be poor due to insufficient numbers of rated entries. We believe that our context-aware service recommender may face the same data sparsity problem as existing recommender systems. We address data sparsity challenge by including more rated entries from multiple, similar contexts. For example, Jane may go to a grocery store that she has never been to before. A traditional recommender system would have no information as a basis to recommend services in this new environment. This problem is referred to as the *cold-start* in traditional recommenders. However, by analyzing similar contexts to find that this store is similar to the two supermarkets Jane usually visits, our context-aware recommender system avoids the cold-start problem and can recommend services similar to those used in other supermarkets.

1.4 Contributions

We present two contributions in this paper:

- An architecture for context-aware service recommender systems for smart environments;
- A new class of algorithms, called *context-based algorithms*, for use in these recommender systems.

We have not yet deployed our service recommender system, so we do not yet have a real dataset to evaluate the quality of our recommendation. In this paper, we try our best to justify our algorithms by examples and analogous results from traditional recommender systems.

2 Modeling the Context-Aware Service Recommender

We begin by presenting the architecture we use to model our system in Figure 2. We assume that smart environments install sensors that can detect user context and transmit low-level context data to a context server. All transmission of context information between components in the architecture uses secure connections (e.g., SSL) for privacy protection. The context server in the smart environments translates sensor data into application-level *context events*, which are sent to a user's mobile device (assumed Internet-capable). This is a well-accepted architecture [3] which we extend here. Context events contain, among other things, the user identity, the type of the event, and the smart environment identity (e.g., *<"Jane", "Pick up milk," "Supermarket A">*).

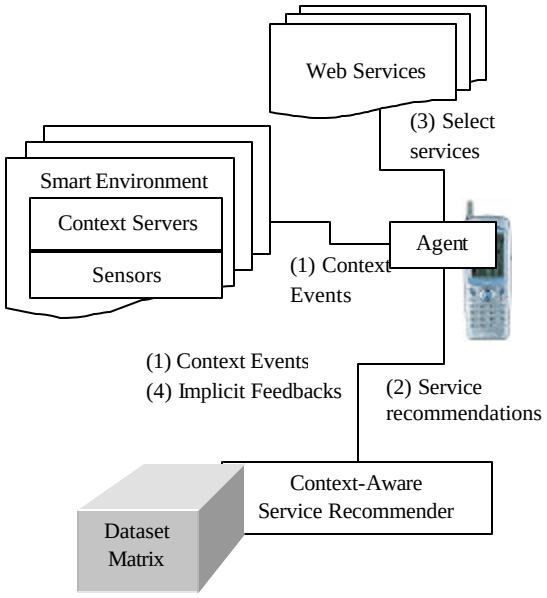

Figure 2: Context-Aware Service Recommender

Context events are handled by a *service recommender agent* running on the user's mobile device. The agent is a thin client for the service recommender. The *service recommender* is a browser-based service, accessible from the Internet. The recommender agent relays the context events to the service recommender. The purpose of the service recommender is to recommend

other browser-based services that may be relevant to the user's context. The service recommender contains a *service relevance rating dataset* (referred to as dataset or matrix in this document), which is the matrix of relevance ratings for services used by users in different contexts, described in Section 1.3.

Once a context event is sent to the service recommender, the service recommender uses the context event as input to an algorithm that searches the dataset for services relevant to that context event (1). The recommender returns a list composed of a combination of services that a user has used, and determined to be relevant in that context before, and services that have not been used in that context, but which may be relevant based on information from similar contexts, users, or services. This appears to a user as a list of N top-rated services, displayed in the agent on the mobile device (2). The user may select any number of recommended services from this list of services and invoke them (3). The agent monitors which recommended services are invoked, and which are not, and relays that information as *implicit feedback* to the service recommender (4). The service recommender uses this feedback to adjust its ratings in the dataset.

2.1 Ratings and Implicit Feedback

As mentioned before, the dataset used by the service recommender is a three-dimensional <*user, context, service*> matrix. Each point in this matrix is a triple containing a numeric *rating* (between 0, for the least relevant services, and 1, for the most relevant services), the number of invocations of a service, and the number of times the service was recommended. We will denote ratings as R, the number of invocations of a service as $N_{positive}$, and the total number of times a service is recommended as N_{total}. We derive *support* of a rating from N_{total}, meaning how much confidence we have on a rating. These variables will be used throughout the paper to determine how we compute predictions from ratings, and how ratings change over time. Points in the matrix may also be empty (i.e. contain no data); an empty data point indicates that a given user has not used a given service in a given context.

Unlike existing recommender systems that ask users for explicit ratings on items, our service recommender infers ratings from users' *implicit feedback*, which is determined by monitoring whether or not recommended services are used. Implicit feedback is a key element of the system's design: if explicit feedback on the actual relevance of recommended services were required from users, the service recommender would be too intrusive and cumbersome to use in daily tasks. Therefore, we evaluate the quality of recommendations and the relevance of services using implicit feedback. There are two types of implicit feedback in our system: positive and negative, and they are used as shown in Table 1.

	Recommended Services	Non-recommended Services
Selected Services	Positive	Positive
Ignored Services	Negative	(none)

Table 1: Implicit Feedback

Services that are recommended and selected by the user are given positive feedback, indicating that the recommendation was correct, and the service is relevant to the user's context. Services that are recommended but ignored by the user are given negative feedback, with the reasoning that for a sufficiently small list of recommended services the user has had the chance to review and ignore each of them. Services that are not part of the top list of recommendations, but are explicitly searched for and used by the user are given positive feedback. Lastly, no implicit feedback can be inferred from services that are neither recommended nor used, because the user has not had the opportunity to review them.

Implicit feedback is created by the recommendation agent for every context event that triggers the recommendation of a list of services. The list of services is partitioned into those receiving positive feedback and those receiving negative feedback. This implicit feedback is used by the recommender to compute new ratings and update existing ratings in the dataset, to reflect user's behavior on services in a context. We have found a number of ways to calculate a rating from implicit feedback. One such formula is:

$$R = \begin{cases} aR_{prev} + (1-a) & \text{if service is selected} \\ R_{prev} - b & \text{if service is ignored, and } R_{prev} > b \\ 0 & \text{if service is ignored, and } R_{prev} \le b \end{cases} \quad (4)$$

In the first part of the formula, ratings are increased exponentially by weighting the old rating with a factor of a ($0 < a << 1$), and adding $(1 - a)$ if the service is invoked. In the second and the third parts of the formula, the ratings are decreased linearly by subtracting a constant b ($0 < b << 1$) if the service is not selected. If the rating reaches 0, then it remains at that level. This formula ensures that positive feedback results in fast rating increases and negative feedback results in slow, steady rating decreases.

2.2 Privacy

Like any recommender systems, user privacy is an important issue. There are three entities involved in the recommender systems – *smart environment operator(s)*, *service recommender provider(s)*, and *users*. There are two pieces of information that can be subject to privacy protection – *context events* generated by the smart environment and sent to service recommender indirectly through the user's mobile device, and *implicit feedbacks* sent from the user's mobile device to the service recommender. Since both context events and feedbacks go through the user's mobile device, the user can control whether to share them with the service recommender. The recommender agent running on the mobile device uses a simple permission-based scheme. The user can choose not to share any implicit feedbacks with the service recommender. However, given the lack of feedbacks from the user, the service recommender can only provide non-personalized popularity-based service recommendation to the active user. The user can also choose not to share any contexts from some specified smart environment with the service recommender. However, the lack of context disables service recommender because the context is a necessary input for the service recommender.

3 Prediction and Recommendation

In this section, we describe a new class of collaborative filtering algorithms that are used in the context-aware service recommender to generate predictions and recommendations. We call them *context-based collaborative filtering* algorithms, and they can provide personalized service recommendations. The inputs to the algorithms are the active user and his or her current context. Based on the inputs, the algorithms predict ratings for the active user in the current context. The algorithms then combine the existing ratings (if any) and the predicted ratings to derive a final prediction. The algorithms generate the top-N service recommendation according to final predictions.

Context-based collaborative filtering algorithms improve on algorithms in traditional recommender systems by leveraging the additional context dimension in a number of ways. The context-based algorithms can use multiple, similar contexts to select better, higher quality user or service neighborhoods, and use existing ratings from these neighborhoods to predict ratings in the current context for the active user. The context-based algorithms also apply additional weightings to existing ratings in the dataset from multiple contexts to compute the predictions. We believe that these additional weightings are generalizable to any prediction schemes that derive ratings from implicit feedback and draw predictions from ratings in multiple contexts.

We divide the context-based collaborative filtering algorithms into the following three steps: (1) computing similarities between contexts, (2) forming user or service neighborhoods from multiple contexts, and (3) predicting ratings. The detail of each step is explained in the Sections 3.1 to 3.3.

3.1 Computing Context Similarities

Context-based algorithms compute context similarities such that the algorithms can draw predictions from multiple, similar contexts. To compute similarity between two contexts, c_i and c_j, a simple method would be to apply Pearson correlation between two slices of the matrix corresponding to these two contexts. Since the original Pearson correlation works on two vectors rather than two matrices, we need to transform each matrix by folding columns or rows in the matrix slice into a single, long vector. The problem with this simple method is that the original dataset may be very sparse, so there may be very few *co-rated* entries between two context matrix slices. Co-rated points in two sets are defined as those corresponding points in each set that are non-empty. For example, in the sets {a, -, c, d} and {A, B, -, D}, the co-rated entries are the first and fourth. Since the accuracy of the Pearson

correlation depends on the number of co-rated entries between two context matrix slices, sparse matrix results in inaccurate correlation values. To solve this problem, we introduce two methods to reduce data sparsity in the original matrix: *service categorization* and *user aggregation*.

Service Categorization

Individual browser-based services are grouped into categories, and each service category acquires an aggregate rating computed as the average of ratings of services in the category. The result is that the size of the service dimension is significantly reduced. The dataset therefore becomes less sparse, and the number of co-rated entries between contexts increases such that Pearson correlation may be used successfully. Once the dataset has been made denser, we can then fold the context slices into context vectors, and apply the Pearson correlation.

User Aggregation

All the users' ratings on a service in a given context are averaged into an aggregate rating. The intuition is that when calculating the similarity between two contexts, it is sufficient to use average users' ratings on services. The result is that the original <*user, context, service*> matrix is reduced in dimension to a <*context, service*> matrix. Again, the density of data in this matrix is greater, so we can directly apply Pearson correlation to find the context similarities.

These two methods are complementary to each other, meaning that they can be applied either together or separately. If one method does not sufficiently reduce the data sparsity in the original dataset, the other method may be applied to the partially-reduced dataset to further reduce it. Also note that this reduced dataset is *only* used to determine similarities between contexts and is not used in the final computation to predict missing ratings. This ensures that aggregation, which explicitly removes personal user preferences and/or service specifics from the dataset, does not have any negative effect on the final recommendation quality.

The final result of this step is to generate a *context correlation table*, containing the calculated similarities between every context. The process of reducing the dataset through service categorization and/or user aggregation and then computing Pearson correlations between each context in the reduced dataset is computationally intensive, so it is done offline at regular intervals (e.g., once per day). This yields an acceptably accurate table because context events, smart environments, and the similarities between them are relatively static – they are very unlikely to change significantly between computation intervals.

3.2 Forming User or Service Neighborhood from Multiple Contexts

Context-based collaborative filtering algorithms have two approaches to generate predictions for ratings on services. The first approach is to form a *k*-nearest user neighborhood based on ratings from multiple contexts. The second approach is to form a *k*-closest service neighborhood, again based on ratings from multiple contexts. Rating predictions are then derived from existing ratings in the user or service neighborhood. These two approaches are analogous to the user-based and item-based approaches in traditional recommender systems (see Section 1.2).

In comparison with traditional recommender systems, the context-based algorithms are able to select higher quality user or service neighborhoods by making use of existing ratings from multiple contexts. High quality means that neighbors have high similarities or correlations with the active user or service. However, we must do additional computation to achieve this. The process of forming user or service neighborhoods in context-based collaborative filtering has three steps: creating a user or service correlation table, determining eligible users or service, and determining the closest neighbors.

The first step of creating a user or service correlation table can be accomplished by folding each two-dimensional user or service slice of the three-dimensional matrix into a user or service vector and applying Pearson correlation on the resultant vectors. This process is similar to how context correlation is computed in Section 3.1. The result of this step is to generate a *user or service correlation table* that contains correlations between every pair of users or services in the dataset. If the dataset is too sparse, it is possible to aggregate contexts to increase the data density; however this may result in less-personalized recommendations. Like the context correlation table,

the user or service correlation table is also computed offline because users and services are considered relatively static over time.

Because the collaborative-filtering recommendation formulas require data which directly pertains to the target service whose rating is being predicted, our second step is to determine which users or services are *eligible* to participate in the recommendation calculation:

- A user is eligible if it has a rating on the target service, in one or more similar contexts;
- A service is eligible if it is rated by the active user, in one or more similar contexts.

If the user did not have a rating on the target service or if the service were not rated by the active user, that user or service would be ineligible to serve as a neighbor, meaning that it would not be useful in the final prediction calculation described below in Section 3.3.

After ineligible users or services are eliminated, the final step is to find the *k*-nearest eligible neighbor users or services to the active user or service. We describe one possible method to find *k* eligible user or service neighbors. It selects neighbors from the *m*-closest contexts to the current context, based on a lookup in the context correlation table. Eligibility of neighbors is determined by restricting similar contexts to these *m*-closest contexts. Neighbors are ranked according to their correlation with the active user or target service, determined by a lookup in the user or service correlation table.

Once the neighbors are ranked, the top *k* neighbors with highest rankings are selected to form the user or service neighborhood.

3.3 Calculating Predictions

The last step of the context-based collaborative filtering algorithms is to predict ratings for the active user in the current context. The predicted rating is computed as a *weighted average* of existing ratings from similar users, services, or contexts.

The key element in prediction is to assign an appropriate weight to each user in the *k*-closest user neighborhood, to each service in the *k*-closest service neighborhood, or to each context in the *m*-closest context neighborhood. In addition to weightings used in existing collaborative filtering [7, 11], the context-based algorithms have the following considerations:

Support Weighting (w_{sup}): Since ratings are derived from cumulative, implicit feedbacks we would like to place more *support* on ratings that are based on larger sample sizes than those that are based on smaller sample sizes. The support of a rating is a function of the number of feedbacks as discussed in Section 2.1. It is given by:

$$w_{sup} = \begin{cases} \dfrac{N_{total}}{N_{threshold}} \text{ if } N_{\text{total}} < N_{threshold} \\ 1 \end{cases} \qquad (5)$$

In this formula, $N_{threshold}$ is determined through experimentation. If the total number of feedbacks in a context is less than the threshold, the existing rating is adjusted by the support weighting.

Context Similarity Weighting (w_{sim}): User or service neighborhoods are selected from multiple contexts. When we compute prediction on a service for the active user in the current context using ratings from user or service neighborhoods, we need to account for varying context similarities between the current context and its *m*-closest context neighborhood from which ratings are drawn. This weighting is obtained by looking up in the context correlation table.

Context Significance Weighting (w_{sig}): One issue in predictions based on multiple contexts is the *amount of trust* (*significance*) on correlations between contexts. We believe that it may be common for the current context to have highly similar context neighbors that are based on very few numbers of co-rated services. The more data point we have to compare, the more we can trust that the correlation is the true representative of the relation between two contexts. We believe that the accuracy of prediction can be further improved if we adjust those ratings that are based on too few samples. The weights to predictions are adjusted according to a context significance weighting given by:

$$w_{sig} = \begin{cases} \dfrac{M}{M_{threshold}} \text{ if } M < M_{threshold} \\ 1 \end{cases} \qquad (6)$$

We denote M as the number of co-rated services between two contexts in the reduced dataset calculated in 3.1, and $M_{threshold}$ is an experimentally determined threshold. This weighting gives preference to correlations based on adequate sample size. Because the context significance weightings are computed from context correlation table that is pre-computed, we can pre-compute the context significance weightings and store them in a *context significance table*. The algorithms simply look up the table at runtime.

3.3.1 Prediction based on User Neighborhood

We will first show how context-based algorithms compute the prediction from k-nearest user neighborhood in m-closest contexts. We start with the original formula given by Equation (2) that has been shown to perform well in predicting ratings. Then we modify it with multiple contexts by incorporating the additional support weighting on each rating, and context similarity, context significance weightings on each context in m-closest context neighborhood:

$$P_{au,as,ac} = \frac{\sum_{u \in U, c \in C} R_{u,as,c} \cdot W}{\sum_{u \in U, c \in C} W} \quad (7),$$

where $W = w_{sim(au,u)} \cdot w_{sup(u,as,c)} \cdot w_{sim(ac,c)} \cdot w_{sig(ac,c)}$

We denote $P_{au,as,ac}$ as the prediction on a target service as for the active user au in the current context ac using ratings from its k-nearest user neighbors. W is a combined weighting from: (1) user similarity weighting $w_{sim(au,u)}$ between the active user au and a user u who is in the user neighborhood; (2) support weighting $w_{sup(u,as,c)}$ of the rating on the target service as for a user u in a context c, which is one of the m-closest contexts; (3) context similarity weighting $w_{sim(ac,c)}$ between the current context ac and the context c; and (4) context significance weighting $w_{sig(ac,c)}$ between the current context ac and the context c. This method computes a prediction by performing a weighted average from the user neighborhood.

3.3.2 Prediction based on Similar Services

Next, we show how context-based algorithms compute predictions from k-closest service neighborhood. As above, we use a reduced-size dataset of k-closest service neighbors in m-closest contexts, and we must consider the correlation between contexts. We start with the original weighted sum formula of Equation (3) and modify it with multiple contexts by incorporating the support weighting on each rating, context similarity weighting, and context significance weighting on each m-closest context:

$$P_{au,as,ac} = \frac{\sum_{s \in S, c \in C} R_{au,s,c} \cdot W}{\sum_{s \in S, c \in C} W} \quad (8),$$

where $W = w_{sim(as,s)} \cdot w_{sup(au,s,ac)} \cdot w_{sim(ac,c)} \cdot w_{sig(ac,c)}$

$R_{au,as,ac}$ is the rating of the active user au on the target service as in the current context ac. W is a combined weighting from confidence weighting, service similarity weighting, context similarity weighting, and context significance weighting.

3.4 Top-N Service Recommendation

Once the rating predictions are computed, the recommender uses a combination of the calculated prediction and the existing rating to derive a final rating on the target service for the active user in the current context. The reason is that if an existing rating has a low support, we would like to use a support weight adjusted rating for top-N service recommendation. We use the following formula to derive the final prediction:

$$P_{final} = (1 - w_{sup}) \cdot P + w_{sup} \cdot R \quad (9)$$

We denote P_{final} as the final prediction. w_{sup} is the support weight on the existing rating R (if rating is not empty), and P is the prediction computed from Section 3.3. The recommender then returns top-N service recommendation with the highest final predictions.

4 Related Work

Collaborative filtering recommender systems [2, 7, 11] are widely accepted technique in electronic commerce, but use little to no context information in generating recommendations. Our use of context information in the service recommender is the significant difference from these existing systems.

Context-based infrastructure for smart environments [3] describes a new abstraction that separates the acquisition and interpretation of context data from the application of context data by context-aware applications. Since this architecture has been well accepted in the research literature, we use it as the basis for formulating our problem space. Our context server in the smart environment shown in Figure 2, corresponds to the aggregated modules of context widgets, server, and interpreters in [3]. Our service recommender is a context-aware application that subscribes to the application-level context events from the context server.

Interactive Workspaces [6] explores new possibilities for people to work together in technology-rich spaces with computing and interaction devices on many different scales. The goal is to design a new architecture that makes it easy to create and add new display and input devices, to move work of all kinds from one computing device to another, and to support and facilitate group interactions. The system adopts an infrastructure-centric approach to ubiquitous computing. The philosophy of Interactive Workspaces is similar to our context-aware service recommender in that both systems incorporate existing services rather than new services designed specifically for the smart environment. However, our service recommender differs in that it is concerned with pushing services to users, rather than integrating disparate services with a smart environment.

CoolTown [9] is a context-aware (location-aware) ubiquitous system that offers a web model for supporting nomadic users. Each physical entity in the CoolTown system has a web resource that allows a user to browse and interact. Users of CoolTown system carry mobile devices, such as PDAs with wireless Internet access. When a user approaches an entity and points a mobile device at it, the URL of the entity's corresponding web resource is transferred to her mobile device via IrDA or RF radio. The user's mobile device then fetches the web resource in a browser. Like our context-aware recommender, CoolTown provides web-based services to users in smart environments. However, in CoolTown, physical objects in the smart environment and web-based services are tightly and statically coupled, meaning that users do not get personalized services by interacting with the entities; whereas in the context-aware service recommender, the objects and services are decoupled by the context events and recommendation process, such that different users may get different service recommendations when interacting with the same object.

The GUIDE [3] is context-aware tourist guide system that provides city visitors with relevant services to their current contexts. The GUIDE allows visitors, using handheld devices with wireless connection, to view their location-based information and to create tailored tours of the city attractions on browsers. Like our context-aware service recommender, GUIDE system provides web-based services to users in smart environments. However, services need to be designed and integrated specifically with the GUIDE system, and choices of these services are limited. In contrast, our context-aware service recommender does not limit choices of web-based services that can be used in smart environments.

5 Conclusion and Future Work

Recommending context-relevant services can help to improve user experience in using services on mobile devices in ubiquitous smart environments. It benefits mobile users by pushing the relevant services to them instead of requiring them to search and filter information on mobile devices, where it is inconvenient, slow, and costly. It is a step toward our vision of *seamless integration* between web-based services and smart environments, creating a *seamless user experience* where using web-based services is easy and effortless for users. In this paper, we have described a context-aware service recommender system that recommends relevant browser-based services to mobile users in smart environments. We have accomplished this by creating a new class of context-based collaborative filtering algorithms that compute predictions in an expanded *<user, context, service>* matrix, by forming user or service neighborhoods selected across multiple contexts.

An important factor in our context-aware recommender is implicit feedbacks. We have shown that ratings can be derived by observing which services are used or ignored by users. Another important consideration is data sparsity and the consequent cold-start problem.

We have shown that we can form better recommendations than existing systems by taking advantage of data from multiple contexts, and by reducing data sparsity by aggregating users or services. Finally, we have contributed a set of prediction formulas that use additional weightings to describe support for ratings derived through implicit feedback, similarity between contexts, and confidence in context similarities. We believe that these additional weightings are generalizable to other prediction schemes that use implicit feedback or multiple contexts.

We are in the process of implementing the system, and plan to deploy it in our laboratory and in other environments with interested third parties to obtain a real-world dataset. This would allow us to evaluate the different approaches in the design. We have described that there are several experimentally determined thresholds that affect the weightings in our algorithms; the selection and tuning of these parameters determines the quality of recommendations, and the computational performance of the algorithms.

There are many future directions that we would like to improve our system. Automated service composition, such as SWORD [12], has been attracting growing attention in the web and ubiquitous computing research communities, and we would like to be able to integrate this research with the service recommender. It is possible that viable service compositions could be *personalized* by using the same context-based collaborative filtering techniques we introduce here. These personalized, composed services would offer a higher degree of serendipity to users: they may discover new, relevant compositions of services that they would not otherwise discover. We would like to leverage the research results from existing service composers to extend the capabilities and usefulness of our system.

Another problem is the lack of automated service invocation. That is our service recommender currently requires users to manually supply context information, although detected by the smart environments, as the input parameters to browser-based services. We would like to enhance our system such that it can automatically extract context data and supply correct input data to services for users, similar to XForm approach by Barton ed al. [1]. This will provide another level of seamless integration between web-based services and smart environment with better seamless user experience.

References

1. John Barton, Tim Kindberg, Hui Dai, Bodhi Priyantha, and Fahs Al-bin-ali, "Sensor-enhanced Mobile Web Clients: an Xforms Approach", *submitted to WWW 2003, http://www.hpl.hp.com/personal/John_Barton/#WebBasedUI.*
2. John S. Breese, David Heckerman, and Carl Kadie, "Empirical Analysis of Predictive Algorithms for Collaborative Filtering", *In Proc. of the 14th Conference on Uncertainty in Artificial Intelligence (UIA-98), pp 43-52*, July, 1998
3. Keith Cheverst, Nigel Davies, Keith Mitchell, and Adrian Friday, "Experiences of Developing and Deploying a Context-Aware Tourist Guide, The GUIDE Project", *In Proc. of MobiCom'00*, Aug. 2000
4. Anind K. Dey, Gregory D. Abowd and Daniel Salber, "A Context-Based Infrastructure for Smart Environments", *In Proc. of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE'99)*, Dec. 1999.
5. W. Keith Edwards, Mark W. Newman, Jana Swdivy, Trevor Smith, and Shahram Izadi, "Challenge: Recombinant Computing and the Speakeasy Approach"", *In Proc. of ACM MobiCom'02*, Sept, 2002
6. Armando Fox, Brad Johanson, Pat Hanrahan, and Terry Winograd, "Integrating Information Appliances into an Interactive Workspace", *IEEE Computer Graphics and Applications, 20:3* (May/June, 2000)
7. Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl, "An Algorithmic Framework for Performing Collaborative Filtering", *In Proc. of the 1999 Conference on Research and development in Information Retrieval*, Aug. 1999.
8. Emre Kiciman, Laurence Melloul, Armando Fox, "Position Summary: Towards Zero-Code Service Composition", *In Proc. of Eighth Workshop on Hot Topics in Operating Systems (HotOS)*, May, 2001
9. Tim Kindberg, John Barton, "A Web-Based Nomadic Computing System", *Computer Networks, Vol. 35, No. 4, pp 443-456*, Mar. 2001
10. A.K. Jain, M.N. Murty, and P.J. Flynn, "Data Clustering: A Review", *ACM Computing Surveys (CSUR), Vol. 31, Issue 3*, Sept. 1999
11. Badrul Sarwar, George Karypis, Joseph Knostan, and John Riedl, "Item-based Collaborative Filtering Recommendation Algorithms", *In Proc. of the 10th International World Wide Web Conference (WWW10)*, May, 2001.

12. R. Shankar, Armando Fox, “SWORD: A Developer Toolkit for Web Service Composition”, *In Proc. of 11th World Wide Web conference (WWW’02)*, May, 2002.