5-2005

Dynamically-optimized context in recommender systems

Ghim-Eng YAP

Ah-hwee TAN
Singapore Management University, ahtan@smu.edu.sg

Hwee-Hwa PANG

Citation

YAP, Ghim-Eng; TAN, Ah-hwee; and PANG, Hwee-Hwa. Dynamically-optimized context in recommender systems. (2005). *Proceedings of the 6th International Conference on Mobile Data Management (MDM'05), Ayia Napa Cyprus, May 9 - 13*. 265-272.
Available at: https://ink.library.smu.edu.sg/sis_research/6567

Dynamically-Optimized Context in Recommender Systems

Ghim-Eng Yap, Ah-Hwee Tan
School of Computer Engineering
Nanyang Technological University
Nanyang Avenue, Singapore 639798
{yapg0001, asahtan}@ntu.edu.sg

Hwee-Hwa Pang
Institute for Infocomm Research
21 Heng Mui Keng Terrace
Singapore 119613
hhpang@i2r.a-star.edu.sg

ABSTRACT

Traditional approaches to recommender systems have not taken into account situational information when making recommendations, and this seriously limits the relevance of the results. This paper advocates context-awareness as a promising approach to enhance the performance of recommenders, and introduces a mechanism to realize this approach. We present a framework that separates the contextual concerns from the actual recommendation module, so that contexts can be readily shared across applications. More importantly, we devise a learning algorithm to dynamically identify the optimal set of contexts for a specific recommendation task and user. An extensive series of experiments has validated that our system is indeed able to learn both quickly and accurately.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*relevance feedback, selection process*; H.1.2 [**Models and Principles**]: User/Machine Systems—*human information processing*; I.5.5 [**Pattern Recognition**]: Implementation—*interactive systems, special architectures*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*performance evaluation (efficiency and effectiveness)*; I.2.6 [**Artificial Intelligence**]: Learning—*Induction*

General Terms

Human Factors, Algorithms, Design, Experimentation, Performance

Keywords

recommender system, machine learning, user feedback, context weight

1. INTRODUCTION

A *recommender* is a system capable of ranking a list of similar items with respect to one or more given criteria. There are many applications of recommender systems, notably in the information retrieval field, where criteria are submitted as queries and the most relevant documents are returned. With the proliferation of mobile E-services, recommenders in various forms have been the subject of many researches due to their obvious commercial values (e.g. [23]). The two main recommendation techniques in use today are the *content-based* approach and the *collaborative filtering* approach [17].

In *content-based* recommendation, the system suggests to users items that best fit a specified set of criteria. The system has to understand all the main features describing the items of interest in order to rank them in terms of their relevance to the criteria. Such systems typically maintain a notion of user profile that is highly specific to the problem domain e.g. profiles for *research-paper* recommenders typically reflect only the user's research interests, although the sharing of profiles among autonomous agents has been suggested [18]. On the other hand, *collaborative filtering* works by seeking the opinions of a community of users to assist individuals in that community to better identify contents of interest from a potentially overwhelming set of choices [19]. It relies on majority judgment within the community, and is largely based upon the notion of stereotypical behaviors and common interests. Hybrid systems exist today that use combinations of these two techniques to improve performance.

These traditional approaches suffer from a serious shortcoming: the set of features for consideration is *static* (fixed at design time) and *entirely task-specific* in nature. For instance, a user's query on a restaurant recommender could be "*restaurants with vegetarian food*". A conventional recommender would simply rank all the known restaurants based on whether vegetarian food is available. Unknown to the application designer, and hence to the recommender, is that it is raining outside and the user would have preferred a nearer eating place. Clearly the system is not able to provide the best recommendations due to its detachment from the current situation; what is lacking in these traditional approaches is an *awareness of the contexts*. According to [7],

[Context is] any information that can be used to characterize the situation of an entity, where an entity can be a person, a place, or an object relevant to the interaction between the user and application, including the user and ap-

plication themselves. A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

Over the years, many researchers have actively tried to realize the potentials of context-awareness e.g. in the more recent field of context-based information retrieval [15]. However, previous work on context-aware computing mostly support only *static* context, e.g. [1, 6, 10, 11, 12, 16, 22]. In all these applications, the composition of context is fixed at design time and no attempt is made to dynamically reduce or expand this set of contexts via learning through usage. In view of this, we propose a generic framework that enhances the existing recommender systems with a task-oriented, context-aware model. Our main contributions are described as follows.

- We advocate a formal distinction between *context instantiation* and *recommendation*. Separation of these concerns via an interactive, component-oriented design allows for a greater flexibility and clarity in role-division. As such, our framework is *generic* and readily applicable to diverse problem domains. For example, an instantiation module can serve multiple different recommenders simply by plugging in appropriate context taxonomies. This modular design promotes efficiency and frees developers to focus on the task at hand.
- We propose the dynamic optimization of contexts for making recommendations to particular users. Systems would benefit from an intelligently-selected set of contexts in a myriad of ways. For example, the application designer may have specified contexts not regarded as relevant by a user. So our system would prune away the least relevant contexts while expanding it with more relevant ones overlooked by the designer. By instantiating only the most significant set of contexts, recommendation resources and time would be saved and the results would be more accurate.

The remainder of this paper is organized as follows: we present the details of our generic framework in Section 2, followed by the description of a sample implementation to realize this framework in Section 3. Section 4 presents our findings from experimental evaluations of this implemented system. In section 5, we compare our research to a number of related work. A discussion on some possible directions for future work in Section 6 concludes the paper.

2. CONTEXT-AWARE RECOMMENDER FRAMEWORK

Figure 1 illustrates our proposed framework for domain-relevant context-aware recommendation systems. The modules within the framework are described below.

2.1 External Data Sources for Context

We treat the acquisition of raw context data and the associated reliability, security, and privacy issues as highly interesting but outside the scope of this paper. As such, we have modelled the scenario where all contextual information are readily available from external providers, e.g. those services depicted in Figure 1.

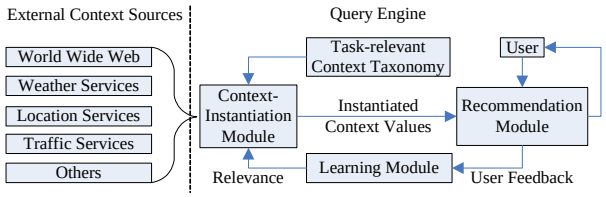

Figure 1: Our Proposed Framework

2.2 The Query Engine

With reference to Figure 1, the user directly interacts only with the recommendation module. In each recommendation cycle, the user identifies the best recommended item and the choice is fed back into our learning module. The components that make up our query engine are namely the *context taxonomy*, the *context-instantiation module*, the *recommendation module*, and the *learning module*.

Each application is expected to define its own specific *taxonomy of contexts*. For instance, for the task of *buying a car*, the specifications of different car models could be considered as part of the relevant taxonomy. We can then learn to personalize the relative weights of these context parameters over time, so that eventually we are able to identify the most significant parameters for that particular user and use only this optimal set for subsequent recommendations.

The *context-instantiation* module resolves the contextual values with external context sources. It is neither concerned nor involved in the actual ranking of the parameters, but is entirely responsible for reliably instantiating the contexts dictated by the learning module and then passing on these values to the recommendation module. Since different sets of contexts could be specified as being relevant for each recommendation, the *recommendation module* must be able to reliably act upon any given subset of the complete taxonomy in making its recommendations.

Our *learning module* captures the user feedback on the satisfaction level of each recommendation and uses it to fine-tune the composition of the context set. By examining the properties of items chosen by a user when different sets of context values are presented, the learning module can identify through the actual usage the optimal relative parameter rankings for that user. This set of most significant contexts should stabilize over time such that it accurately reflects the user's real interests.

3. REALIZING THE FRAMEWORK

We demonstrate our proposed framework via a sample implementation involving the context-based recommendation of restaurants.

3.1 Realizing the *Context Taxonomy*

For this sample application, we represent the contextual information in XML format, and capture the structure of the context taxonomy in a DTD *document type definition*. The

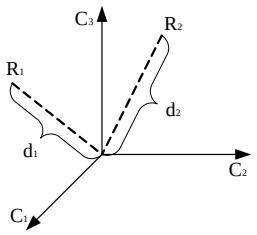

Figure 2: A Simple Context Space

communication of contextual data between our context instantiation module and each of the external sources is then performed based on the definitions in this document. Using this elegant arrangement, we implement several SOAP services for providing contextual information including *location*, *weather* and restaurant-related data on a J2EE server. Each context is treated as a separate entity that could be relevant to the recommendation. This makes the application amenable to future addition and removal of contexts.

We have chosen DTD for context representation because we are looking to structured web services for future data interactions. This is in line with the rapid emergence of various standard DTDs for vertical applications, including the proposals from *Open Buying on the Internet* (OBI) Consortium, *Commerce XML* (cXML) group, Microsoft's *BizTalk Framework*, and RosettaNet's *EConcert* specifications [20].

3.2 Realizing the *Recommendation Module*

For ranking restaurants with respect to *dynamically changing* sets of contexts, we propose the concepts of *rank-conscious contexts* and *context space*. Each rank-conscious context maintains an internal logic for computing the rank value of any restaurant with respect to itself. Whenever a recommendation request arrives at the recommender module, it would go through each of the contexts to obtain its ranking value for each known restaurant. The values computed are represented as a vector of restaurant rankings with respect to that particular context. This vector is then normalized and multiplied by the relative context weight. Following that, a context space is formed with each context as a distinct dimension, and restaurants are represented as position vectors in this space. These positions' distances from the origin can now be compared meaningfully, with restaurants closer to the origin being ranked higher since they satisfy the weighted contextual values better. Figure 2 illustrates a simple context space with three contexts to be considered (axes C_1, C_2, C_3) and two restaurants to be ranked (R_1 and R_2). d_1 and d_2 denote the distances of the restaurants to the origin O.

Our context-based restaurant-ranking mechanism works as follows: Let K be the number of restaurants, N be the number of context parameters, R_k denotes restaurant k, and C_n denotes context n. Let $rv_{k,n}$ be the ranking value of R_k with respect to C_n and $rv'_{k,n} \in [0,1]$ be the corresponding normalized ranking value. Denoting w_n as the relative weight of C_n, the weighted position vector of R_k is given by $\vec{R_k} = [w_1(rv'_{k,1}), w_2(rv'_{k,2}), \cdots, w_n(rv'_{k,n}), \cdots, w_N(rv'_{k,N})]$. Applying the *Euclidean measure* with respect to the origin, the *degree* to which R_k satisfies the weighted set of the context values is given by

$$d_k = \|\vec{R_k}\| = \sqrt{\sum_{n=1}^{N} (w_n(rv'_{k,n}))^2} \quad (1)$$

We assert that R_i ranks higher than R_j if $d_i < d_j$.

As mentioned previously, the computation of ranking values for restaurants is implemented as internal logic within each context parameter. This escalates individual context from a passive piece of information to an active entity that can decide for itself how well each restaurant matches its desired value. Carrying the ranking logic within the contexts and visualizing the restaurants in a context space allow our recommendations to be easily made based on different combinations of contexts. Two sample predicates employed in our application are shown below:

Predicate 1 Rank value rv_k for R_k w.r.t. C_{has_toilet}

Require: *desired*: "*true*"|"*indifferent*"
  if *desired* = "*true*" **then**
    if *actual* = "*true*" **then** $rv_k = 0.0$ //smaller, better
    else $rv_k = 1.0$
  else
    $rv_k = 0.0$ //indifferent

Predicate 2 Rank value rv_k for R_k w.r.t. $C_{cleanliness}$

Require: *desired*: "*good*"|"*average*"
  if *desired* = "*actual*" **then**
    $rv_k = 0.0$ //perfect fit
  else if *actual* = "*bad*" **then**
    $rv_k = 1.0$
  else if *actual* = "*good*" **then**
    $rv_k = 0.0$ //better than expected
  else
    $rv_k = 0.5$ //not that good, just average

3.3 Realizing the *Learning Module*

Our learning process involves the extraction of ranking constraints from user selections. Let r be the ranked list of restaurants presented to the user based on the current set of context values. The learning process requires the user to make a single selection s from r. In our design, the individual restaurant's information is readily available to the users so that they can make informed choices. Instead of presenting all the items in r for users to examine, we note the interesting observation by Silverstein et al [21] that users tend to scan only the top ten items in a ranked list of links. We expect a similar user behavior for recommenders in general, hence we decide to present just the top-k recommendations.

What kind of information can we extract from this single selection made by the user? Consider the scenario where three restaurants are recommended and the user makes a single selection among these three restaurants. Note that we do not require the user to *re-order* the presented list, which means that we do not have the complete relative rankings of all items presented in r, but simply know that among the k presented items in the list r, the user has decided that

s best matches the current context values. For example, if the user has chosen the second restaurant, we know that this user favors it over the other two restaurants. We cannot, however, deduce anything about her preference between the first and third restaurants. In general, we cannot deduce anything about the user's preference between any pair of non-selected items. Denoting the user's preferences as $r^* = \{(R_i, R_j) : R_i \text{ is considered a closer match to the current set of context values than } R_j\}$, we can deduce in this particular example that $(R_2, R_1) \in r^*$ and $(R_2, R_3) \in r^*$.

We generalize the above process of extracting pairwise preferences from a single user selection as follows: For a presented ranking, if R_i is chosen, extract pair-wise preferences $r^* = \{(R_i, R_j)\ \forall j,\ 1 \leq j \leq k, j \neq i\}$. With this clear idea of the kind of information that can be extracted from user feedbacks, we can now state the focus of our work more concisely as that of *investigating the effective learning, modelling and application of relative significance among context parameters for a typical recommender system.* This enables us to formulate our problem within the class of *linear ranking functions* as follows:

Given a set S of T training samples each consisting of a set of context values P_t and its corresponding target ranking r_t^*, i.e. $S = \{(P_1, r_1^*), (P_2, r_2^*), \cdots, (P_T, r_T^*)\}$, learn the relative weights vector $\vec{w}$ of the contexts such that as many of the following inequalities are satisfied as possible:

$$\begin{aligned} &\forall (R_i, R_j) \in r_1^* : \vec{w}\phi(P_1, R_i) > \vec{w}\phi(P_1, R_j) \\ &\cdots \\ &\forall (R_i, R_j) \in r_T^* : \vec{w}\phi(P_T, R_i) > \vec{w}\phi(P_T, R_j) \end{aligned} \tag{2}$$

where $\phi(P_t, R_k)$ is the measure of how closely R_k satisfies the values of P_t. Using Eq. 1, the similarity value $\phi(P_t, R_k)$ for sample t is computed as $1 - d_k$ with $w_n = 1.0\ \forall n,\ 1 \leq n \leq N$. Each of the above inequalities in Eq. 2 is known as a *constraint*. We can re-scale $\vec{w}$ and rearrange Eq. 2 as

$$\begin{aligned} &\forall (R_i, R_j) \in r_1^* : \vec{w}(\phi(P_1, R_i) - \phi(P_1, R_j)) > 1 \\ &\cdots \\ &\forall (R_i, R_j) \in r_T^* : \vec{w}(\phi(P_T, R_i) - \phi(P_T, R_j)) > 1 \end{aligned} \tag{3}$$

3.3.1 Learning via Support Vector Machine

We are now in a position to employ any of the available *linear binary classification methods* for learning the relative weights of contexts. Without prejudice to other methods, we choose the popular *support vector machine* (SVM) as our learning approach. The SVM tries to find the optimal hyperplane that separates positive data points from the negative ones while maximizing the *margin of separation*. Denoting d_+ (d_-) as the shortest distance from the separating hyperplane to the closest positive (negative) example (the *support vectors*), this margin is defined as $d_+ + d_-$ [3]. Since our concern lies not in the design of learning SVM nor the analysis of any learning method in particular, we adopt the *Ranking SVM* built by Joachims [14], and reformulate our learning problem as a *constrained optimization problem*:

Find the optimal values of $\vec{w}$ and *slack variables* $\vec{\xi}$ such that they minimize the *cost function*

$$\delta(\vec{w}, \vec{\xi}) = \frac{1}{2}\vec{w}^T\vec{w} + C\sum_{t=1}^{T}\xi_t \tag{4}$$

subject to the constraints

$$\begin{aligned} &\forall (R_i, R_j) \in r_1^* : \vec{w}(\phi(P_1, R_i) - \phi(P_1, R_j)) > 1 - \xi_1 \\ &\cdots \\ &\forall (R_i, R_j) \in r_T^* : \vec{w}(\phi(P_T, R_i) - \phi(P_T, R_j)) > 1 - \xi_T \end{aligned} \tag{5}$$

and

$$\xi_t \geq 0 \quad \forall t, \quad 1 \leq t \leq T \tag{6}$$

where C is a positive parameter controlling the tradeoff between the *margin width* and the *training error* and r_t^* is the observed partial rankings in sample t.

4. EXPERIMENTAL EVALUATION

Human users make decisions by considering the complex interplay of aspects relevant to situations. Let's focus on our sample scenario of choosing restaurants, a typical situation in which we have to make a best-effort selection among available choices and a task in which recommenders are meant to assist us. If the entire set of restaurants is placed before a user who is required to identify the best choice among them, s/he would most likely concentrate only on comparing attributes of interest. This optimal set of contexts that is of real importance to a user is what this part of our work is trying to learn and model.

Our application scenario involves a total of 64 context parameters and 15 restaurants. A typical recommendation round in our experiments (a *sample*) involves context values being generated and presented to a user, who is asked to indicate the recommended restaurant that best matches these values. From each sample, we extract the pairwise preference rankings and employ the SVM^{light} for learning [13], using the tool's default value for tuning parameter C in all our tests.

4.1 Off-line Learning

We begin with a series of off-line learning experiments to verify the correct formulation of our learning problem. Besides determining the smallest suitable sample size for reliable learning, we also wish to investigate the effects on our learning of displaying just the top-3 restaurants in each round. Through this latter set of tests, we aim to confirm two assertions: (1) user's opinion on non-displayed restaurants is not needed for our learning since what we want to learn is his contextual preferences when ranking restaurants and not his preferences among the restaurants, and (2) ensuring that all restaurants have been unbiasedly presented is unnecessary for our formulated learning problem.

Table 1: Results for Simulated User. "$a|b|c$": results are averaged over a sets of b examples, each with c recommendations displayed; LOO: leave-one-out accuracies from SVM^{light}; Fm: F-measure

Test	Training Acc. (%)			CV (%)	LOO (%)		Fm
	Best	2^{nd}	3^{rd}		Recall	Prec.	
12|100|15	83.3	15.8	0.83	76.8	99.1	99.2	0.71
3|1000|15	85.3	14.4	0.37	84.2	99.2	99.5	0.71
12|100|3	90.7	9.00	0.33	80.2	87.8	91.8	0.64
3|1000|3	95.1	4.80	0.10	93.6	95.7	97.2	0.67

startLocation
returnLocation
restaurantCategory
restaurantPaymentAvailableByCash
restaurantCleanliness
restaurantHasVarietyOfFood
restaurantAveragePriceOfAMeal

Figure 3: Specified Contexts of Interest

Table 2: Results for Human User

Test	PFR (% 30 rds)	LOO (%)		Fm
		Recall	Prec.	
1|100|3	93.3	89.9	91.2	0.57
3|60|3	71.1	89.8	90.3	0.57
3|30|3	68.9	85.3	88.6	0.62
3|10|3	-	83.5	84.6	0.41

4.1.1 Learning with a Simulated User

First, we simulate a human user in the sample collection phase by explicitly specifying the contexts of interest. In each round, our program simulates the following actions:

Step 1 Request for a new set of context values

Step 2 Choose the best recommended restaurant

Step 3 Generate pair-wise preferences

The main contribution of our simulation comes in step 2. A human user with the specified context interests would have had to tediously consider each recommendation and choose the one best-fitting each set of the context values. Our program simulates this process by ranking the displayed restaurants using a context space formed by the contexts of interest. This simulation enables large data sets to be created with ease while maintaining a high consistency in decisions. To make our simulation more meaningful, we asked the human subject of our real-user tests to specify the set of interested contexts. We show his choices in Figure 3.

We obtain several data sets based on this simulation process. For each set, we measure the training accuracies in terms of the percentage of examples with the desired outcome matching the top-most (*Best*), 2^{nd}, and 3^{rd} recommendation. In addition, we obtain the 5×2-fold cross-validation (CV) accuracy as well as analyze the predicted relative ordering of contexts based on the learned weights. Average *F-measure* values are then computed in terms of the similarity between the seven contexts with the largest absolute weights and the user-specified set of interested features. Our observations are summarized in Table 1.

We observed that although increasing the sample size to 1000 brought about notable improvements in all the accuracies, the average *F-measure* values showed that the actual proportion of interested contexts among the top seven contexts with the largest learned absolute weights did not improve significantly, regardless of whether all or just the top three restaurants were presented. These tests clearly show that our system's learning ability for a sample size of 100 is comparable to that for a much larger sample size of 1000, and that our earlier assertions on not having to present all available items are empirically sound. We conclude that our formulation of the learning problem is correct and that our learning approach is indeed effective.

4.1.2 Learning with a Human User

Next, we investigate whether our proposed learning improves the recommendations from the perspective of a real user, as well as whether an even smaller set of learning examples might be sufficient. Our simulations have shown that 100 samples are sufficient for learning when the user is consistent in his choices. We now investigate whether such consistency is reasonable to expect of a real user. For this, 100 randomly generated sets of context values are presented to the user together with the top-3 recommendations. These form the base set of examples for our subsequent off-line learning tests.

We split the set of 100 samples into smaller subsets to analyze if a smaller sample size is sufficient. After learning off-line from each of these data sets, we make 30 rounds of recommendations based on just the top-7 contexts with the largest learned absolute weights. The proportion of these 30 recommendations, for which the user indicated the top-most recommended restaurant as best-matching the interested parameters, is the *positive feedback ratio* (PFR). This metric gives an indication of how well the learned system performs from the perspective of the user. The results of these experiments are presented in Table 2.

Our results show that the average values for all the metrics dropped as we decreased the sample size. Interestingly, the F-measure of the learned model remained at around 60% (4 interested among top 7 contexts) as the sample size fell to 30, but plunged to just 40% (3 out of 7) for samples of size 10. We note too that the reduction in size from 100 to 60 caused a 22.2% drop in feedback ratios, but a further cut to 30 resulted in just a small further drop of 2.2%. These

observations suggest that the smallest sample size allowing a level of learning comparable to that of 100 samples is around 30.

In comparison, the computed PFR for our 100 base samples with no learning is just 55%, clearly suggesting that the 68.9% ratio at sample size 30 is a significant improvement. Based on all these observations, we are confident that our proposed approach of learning the relative significance among contexts can indeed be applied to significantly improve the quality of recommendations.

4.2 Online Learning

Next, we extend our experiments to an online learning scenario, adapting the ideas from the incremental algorithm of Domeniconi and Gunopulos [8] for our learning module. Our algorithm considers the user's selections for various sets of context values as a *stream of data* arriving in intermittent batches of size b, and performs learning only on the most recent w batches. At time t, the training set, which can be viewed as a queue, contains batches B_1^t, B_2^t,$\cdots$, and B_w^t, where B_w^t is the most recent batch. At time t+1, B_1^t is discarded and B_2^t to B_w^t are moved forward to form B_1^{t+1},$\cdots$ and B_{w-1}^{t+1}. The latest data batch joins the queue as B_w^{t+1}. This algorithm ensures a training data set of maximum size $w \times b$, significantly smaller than the entire history of past examples. In addition, it captures any shifts in contextual preferences by discarding old examples and learning only from the latest batches.

Our learning algorithm is shown in Algorithm 1 below, in which b denotes the fixed batch size, w denotes the number of recent batches to be learned, W denotes the desired learning window size ($W = w \times b$), and S denotes the example queue, where $|S| \leq (w+1) \times b$ at any time. For a better control over the adaptation rate, we introduce a predefined value *abs_min* as the absolute minimal number of examples for which the first learning would occur.

Algorithm 1 Incremental Learning

Require: $W \geq abs\_min$, $b \leq abs\_min$
Require: abs_min modulo $b = 0$
loop
  gather incoming data batch B, append B to S
  if *first learning not done* **then**
    $min = abs\_min$
  else
    $min = W$
  if $|S| < min$ **then**
    proceed on to next round
  prune examples of S until $|S| = min$, compute *current* PFR on S
  if *current* PFR < *desired* PFR **then**
    if *current* PFR ≥ *previous observed* PFR **then**
      train on S, rank contexts based on learned weights, suggest pruning contexts with |weight| < $a \times$ *largest* |*weight*|
    else
      if *contexts were removed in a previous round* **then**
        suggest restoring these removed contexts
      else
        suggest restoring to full context set

The best value for pruning coefficient a is observed to be 0.33 from our off-line tests. We recall that the user has a list of interested contexts (Figure 3). We require the user to always specify that these interested contexts be kept in consideration, so that we can define the common *saturation point* as when the system *can no longer trim off any contexts other than those interested*. Further samples can then be gathered to see if any of the interested contexts can be removed to yield the optimal set. Due to space constraints, we present only two of our online tests in this paper. For both tests, where only the top-3 recommendations were displayed, the user identified *restaurantCategory* as the most important context and this yielded some interesting observations.

Table 3: Online Results w=10, b=10, abs_min=30

Cycle	Sample Size	No. of Contexts	Observed PFR
1	30 (1-30)	64	0.0
2	100 (1-100)	7	0.7
3	100 (11-110)	5	0.8
4	100 (21-120)	5	0.9
5	100 (31-130)	5	1.0

Table 4: Online Results w=3, b=10, abs_min=10

Cycle	Sample Size	No. of Contexts	Observed PFR
1	10 (1-10)	64	0.1
2	30 (1-30)	16	0.6
3	30 (11-40)	11	0.77
4	30 (21-50)	9	0.7
5	30 (31-60)	11	0.7
6	30 (41-70)	7	0.83
7	30 (51-80)	5	0.97

Table 3 shows the results of the first test. Our saturation condition was satisfied at the end of the first learning cycle involving only 30 samples. The most important context was correctly identified to be *restaurantCategory*, and our system suggested to remove all the other contexts. Analyzing his choices for these 30 samples, we found this suggestion to be sound as the user had mostly chosen a restaurant based on only its category. At the end of the second learning involving the first 100 cases, the system suggested to remove two of the interested contexts, namely *payment available by cash* and *average price of a meal* (The user confirmed subsequently that these were not considered at all throughout the test). Our results show that the user had chosen none of the top-most recommendations when all contexts were considered, but had actually selected all of the top-most recommendations after 30 samples were learned.

We then repeat the experiment with *abs_min* reduced to 10 to improve our online learning's adaptation rate. Our results are summarized in Table 4. Our learning on the first 10 samples significantly improved the PFR from 0.1 to 0.6. A further learning involving the first 30 samples brought the ratio to 0.77. However, the learning at the end of cycle 3 pruned away more contexts, and this caused the feedback ratio to fall slightly to 0.7. At this point, our recovery system kicked in and the last two removed contexts were restored. Subsequent cycles saw the further trimming of contexts and we reached our defined saturation point after cycle 5. After a further learning cycle, the system correctly suggested to

remove the *payment available by cash* and *average price of a meal* contexts, which are in fact the same two that our first test eliminated from the interested contexts.

This series of experiments show that indeed our system is able to correctly identify the user's real interests and hence find the optimal set of contexts among all available parameters. Furthermore, we are able to significantly improve the adaptation rate of our system without compromising the effectiveness of learning. We conclude from this extensive series of both off-line and online experiments that our system is effective in learning and applying the relative significance of context parameters within our recommender framework.

5. RELATED WORK

Dunlop et al described in [9] a palmtop application *CityGuide* for restaurant recommendation based on the match of restaurant types to a user's past preferences as well as ratings given by reviewers with similar preferences. They depended on the two user-specified filters of *food type* and *price* for profiling, and did not employ extensive learning to optimize the context set as what we have done in this work.

Tung et al demonstrated in [23] a prototype design of a software agent that was capable of recommending travel-related information based on the contexts of a user. Their recommendation procedure involved a dialogue between the user and the agent for modifying constraints given to the agent. The initial constraints were entered manually by the user as his or her preferences (e.g. *budget*, *food-type*, *atmosphere*, *smoking* or *non-smoking*). If a recommendation that fully satisfied all these constraints did not exist in the system's restaurant directory, the agent tried to relax the constraints one at a time and proposed these changes to the user, who had a choice of accepting any one of the proposed relaxations. The one restaurant in the directory that best fitted the relaxed set of constraints was then presented. Their system worked only with a statically defined set of contexts which could not be altered or optimized through usage.

A separate paper by Brunato et al proposed a middleware layer that collected a historical database of user position and URL usage information and then analyzed these to discover the links' spatial usage patterns [2]. A preference metric that reflected *where* and *how often* each link had been accessed by previous users was then computed. URLs were then ranked and recommended to the users based on their current location. Like us, they suggested the implicit gathering of usage feedback through user behaviors, e.g. whether any link was clicked, and which link was clicked. However, they considered only the current user location for recommendation, unlike our work which is much more comprehensive and can be readily applied to any contexts that are relevant to the recommendation task.

To the best of our knowledge, the only research that explicitly attempted to employ user modelling techniques within the domain of context-aware computing were the recent series of works by Byun and Cheverst [4, 5]. They argued that the user's preferences could be readily induced from the context history using user modelling and machine learning techniques, and that these modelled behaviors could be used together with the current contexts in supporting proactive environmental adaptations. In [5], the authors described an experiment to demonstrate the learning of user's preferences from the context history for controlling an office environment. They defined a small static set of contexts, and collected the changes in these contexts as a time-stamped context history. From this history, they induced rules for representing the user's preferences regarding the status of window in various situations. They however did not make any attempt to dynamically optimize the user contexts as what we have proposed and extensively investigated.

6. CONCLUSION AND FUTURE WORK

Conventional recommender systems do not consider situational information and this seriously limits the relevance of their results. This paper advocates context-awareness as a promising approach to enhance recommenders' performance. We present a framework that separates contextual concerns from recommendation, so that contexts can be readily shared across applications. More importantly, we devise a learning algorithm that dynamically optimizes the context set for a specific recommendation task and user, and validate through extensive experiments that our system is capable of learning quickly and accurately.

We are extending this work in several directions. Firstly, we would like to investigate the possibilities of meaningfully applying a user's contextual preferences learned for a particular recommender to diverse problem domains in other systems. Along this line, we consider the extension of our approach to modelling groups of users instead of just individuals as an interesting direction.

Secondly, the issue of interdependencies among contexts should be resolved. The status of certain contexts may affect the importance of other contexts during decision-making. For example, the location of a restaurant should gain a greater importance when the weather is bad.

Thirdly, our restaurant recommender has been implemented as a desktop application. We would like to experiment with deployment on a mobile platform like the Smartphone and the PDA in order to study the user interaction issues.

Finally, we have investigated the *dynamic reduction* of the context set for recommendation considerations. It would be useful to be able to also *dynamically expand* the set of contexts for improving the recommendation performance.

7. ACKNOWLEDGMENTS

The reported work is partially supported by an A*Star SERC TSRP Grant No. 042-111-0061 and an A*Star Graduate Scholarship (PhD) to Ghim-Eng Yap.

8. REFERENCES

[1] J. E. Bardram. Applications of context-aware computing in hospital work - examples and design principles. In *Procs of the ACM Symposium on Applied Computing*, pages 1574–1579, March 2004.

[2] M. Brunato, R. Battiti, A. Villani, and A. Delai. A location-dependent recommender system for the web. Technical report, DIT-02-0093, University of Trento,

Dept of Information and Communication Technology, November 2002.

[3] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, pages 121–167, 1998.

[4] H. E. Byun and K. Cheverst. Exploiting user models and context-awareness to support personal daily activities. *Workshop in UM2001 on User Modeling for Context-Aware Applications*, 2001.

[5] H. E. Byun and K. Cheverst. Utilising context history to provide dynamic adaptations. *Journal of Applied AI*, to appear early 2004.

[6] J. D. Carswell, K. Gardiner, and M. Neumann. Wireless spatio-semantic transactions on multimedia datasets. In *Procs of ACM Symposium on Applied Computing (SAC)*, pages 1201–1205, March 2004.

[7] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. Technical report, GIT-GVU-99-22, June 1999.

[8] C. Domeniconi and D. Gunopulos. Incremental support vector machine construction. In *IEEE International Conference on Data Mining (ICDM)*, pages 589–592, 2001.

[9] M. Dunlop, A. Morrison, S. McCallum, and et al. Focussed palmtop information access combining starfield displays with profile-based recommendations. *Mobile and Ubiquitous Information Access Workshop 2003*, pages 79–89, 2004.

[10] K. Fujinami, T. Yamabe, and T. Nakajima. "Take me with you!": A case study of context-aware application integrating cyber and physical space. In *Procs of the 2004 ACM Symposium on Applied Computing (SAC 2004)*, pages 1607–1614, March 2004.

[11] D. Goren-Bar and T. Kuflik. Don't miss-r - recommending restaurants through an adaptive mobile system. In *Procs of IUI'04*, pages 250–252, January 2004.

[12] R. Hexel, C. Johnson, B. Kummerfeld, and A. Quigley. "Powerpoint to the people": Suiting the word to the audience. In *Procs of the 5th Conf on Australasian User Interface*, volume 28, pages 49–56, 2004.

[13] T. Joachims. Making large-scale svm learning practical. *Advances in Kernel Methods - Support Vector Learning*, 1999.

[14] T. Joachims. Optimizing search engines using clickthrough data. In *Procs ACM SIGKDD Int Conf Knowledge Discovery and Data Mining (KDD'02)*, pages 133–142, 2002.

[15] G. J. F. Jones and P. J. Brown. Context-aware retrieval for ubiquitous computing environments. *Mobile and Ubiquitous Information Access Workshop 2003*, pages 227–243, 2004.

[16] N. Kern, B. Schiele, H. Junker, and et al. Wearable sensing to annotate meeting recordings. *Personal and Ubiquitous Computing*, 7(5):263–274, October 2003.

[17] P. Massa and B. Bhattacharjee. Using trust in recommender systems: An experimental analysis. In *iTrust2004 International Conference*, pages 221–235, 2004.

[18] S. E. Middleton, N. R. Shadbolt, and D. C. De-Roure. Ontological user profiling in recommender systems. *ACM Transactions on Information Systems*, 22(1):54–88, January 2004.

[19] P. Resnick and H. R. Varian. Recommender systems. *Commun. ACM*, 40:56–58, 1997.

[20] G. Shankar. The xml standards landscape: Xml holds promise for better business-to-business communication. *http://www.infoworld.com/*.

[21] C. Silverstein, M. Henzinger, H. Marais, and et al. Analysis of a very large altavista query log. Technical report, SRC 1998-014, Digital Systems Research Center, 1998.

[22] L. Terrenghi and A. Zimmermann. Tailored audio augmented environment for museums. In *Procs of the 9th International Conference on Intelligent User Interface*, pages 334–336, 2004.

[23] H. W. Tung and V. W. Soo. A personalized restaurant recommender agent for mobile e-service. In *Procs of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'04)*, pages 259–262, 2004.