User Preferences Initialization and Integration in Critique-Based Mobile Recommender Systems

Quang Nhat Nguyen
eCTRL, ITC-irst
via Solteri, n. 38
38100 Trento, Italy
quang@itc.it

Francesco Ricci
eCTRL, ITC-irst
via Solteri, n. 38
38100 Trento, Italy
ricci@itc.it

ABSTRACT

Recommender systems are decision support tools aimed at assisting users in finding products that best suit their preferences. The success of a recommendation session depends significantly on how, at the beginning of the interaction, the system initializes its representation of user's preferences. In mobile recommender systems, guessing an initial representation of user's preferences is even more difficult because of some limitations of mobile devices as well as characteristics of mobile users. In this paper we propose an approach for user preferences initialization that exploits a range of available knowledge sources related to the user. In this approach personalized recommendations can be generated using both a persistent and a context-dependent user model.

Keywords

User preferences initialization, user modelling, mobile recommender systems

1. INTRODUCTION

Nowadays e-commerce web sites store and offer product catalogues of thousands of items with different characteristics and types. When catalogues are queried by user's search requests often a potentially overwhelming set of options are retrieved. In this case, users must be able to filter out irrelevant products, to compare options, and to select the best one(s). Recommender systems ([9, 3]) are aimed at addressing this problem, suggesting to the user those products which best suit his needs and preferences in a particular situation and context.

For instance, travel recommender systems are aimed at providing support to travellers at any time during their trip: when they build their pre-travel plan, during their move to, or their stay in, the selected destination, and possibly when the trip finishes [10]. In this paper we focus on the on-tour scenario, and on the initial representation of user's preferences in a mobile travel recommender system. On-tour recommendations are delivered through mobile devices, when the user is searching for some interesting travel product (e.g. a restaurant or an event), possibly complementing his pre-travel plan.

In recommender systems, the system's initial guess of the user's preferences keeps a very important role, since it affects directly the quality of the first recommendation results. If the first results are good enough, then it is more likely that the user will go on interacting with the system; otherwise, this may cause the user to terminate the process. This problem becomes more challenging for mobile recommender systems, where both the time for serving a request and the interaction duration are tighter than those in the web-based environment.

Recommender systems usually derive the initial guess about user's preferences from the user's persistent (i.e., long-term) model. A user's persistent model contains the information (about the user) which persists along a relatively long time span (i.e., through many consecutive recommendation sessions). A user's persistent model is typically initialized at registration time, and revised later on through system usage. To construct an initial model of a new user, recommender systems typically ask the new user, either to provide domain related preferences and demographic information (e.g., in [13]), or to rate a set of sample items (e.g., in [14]), or to provide some like/dislike examples (e.g., in [7]). Then the user's persistent model is periodically revised by explicitly or implicitly eliciting the user's additional preferences. In the "explicit" approach, user's preferences are collected by user's judgment (i.e., ratings) to the recommended items. These ratings are typically used to determine like-minded users (in the collaborative filtering approach [8]), or interesting items (i.e., the content-based filtering approach [4]). In the "implicit" approach, user's preferences are obtained through the analysis of user's web site navigation behavior [6, 15].

In a recommendation session the system automatically exploits the user's persistent model and builds the initial (default) representation of his preferences. In this way some user's information is exploited, and input for a particular request is reduced. However, this approach has three disadvantages. First, there is a potential mismatch between past and current preferences; especially when these change often and quickly in a context-dependent way (e.g., in the travel and tourism domain). The second disadvantage is that the system usually needs to collect a relatively large amount of data, from each single user, before building a reasonably good representation of his preferences. The third disadvantage is that past usage data mining requires lot of on-line computational resource and time. These two last disadvantages become serious limitations, when one considers the mobile devices' limited capability in computation and in exchanging data.

Because of these limitations, some new recommender systems have focused on the exploitation of context-dependent information, i.e., information about the user and the task, that is considered valid within only the scope of the current recommendation session. Context-dependent user information comprises not only the space-time information of the user ([1]), which is fundamental for mobile recommender systems, but also other chunks of user's related data, such as: current needs, group composition, budget, etc. The traditional way to acquire user's context-dependent preferences, not only those related to space and time, is by explicit input through a graphical user interface. This approach is very often used in web-based systems (e.g., the Entree system [2] or the NutKing system [11], etc.). Acquiring user's session-specific preferences can reduce the errors made by the system in understanding what the user really wants. In contrast, this approach has the disadvantage of requiring some user effort (input), that even increases using mobile devices (limited screen size and limited input modality). Moreover, users can specify their preferences only if they 1) have enough knowledge about the problem domain, and 2) are willing to do that, i.e., if they are convinced that they will benefit from that.

Persistent user models and context-dependent preferences are two faces of the same coin. Nonetheless, although many recommender systems build a user's persistent model, only a few refine and complement the user's persistent model with context-dependent information acquired and updated during the recommendation session. Almost all the recommender systems that build persistent models revise them periodically; but this cannot cope with the changes introduced by the context of a particular recommendation session. In mobile recommender systems, the integration of persistent and context-dependent user information in the process of building the initial representation of user's preferences is a challenging problem because of two reasons. First, a high accuracy of the initial representation usually requires a high cognitive effort by the user. Second, the initialization process must take into account both the limitations of the mobile environment and the characteristics of mobile users.

In this paper we propose a preference initialization methodology that will be integrated and tested into the MobyRek recommender system [5]. The proposed methodology has the following characteristics:

- Different knowledge sources of user's data are exploited and integrated in the initialization of user's preferences representation.
- A persistent user model and a context-dependent model are integrated.
- Both of these models are revised by exploiting critiques given by user during recommendation sessions. Critique-based elicitation of user preferences, by interleaving elicitation with recommendation, seems to be more effective in pushing the user to elicit his needs while keeping the interaction alive.
- Our approach supports different types of mobile users, whose characteristics range between two extremes, i.e. those that prefer simple and fast interaction styles, hence accepting even moderately accurate recommendations, and those that prefer highly accurate recommendations, even at the cost of a more complex interaction.
- In our model we scale the importance assigned by the user to his preferences, and we distinguish between "must have" and "optional" preferences.

The rest of the paper is organized as follows. Section 2 introduces the formal representation of travel products, and user preferences. Section 3 describes the on-tour recommendation process. In Section 4 we describe our approach to the problem of initializing user preferences. In Section 5, we discuss some open issues and in particular about the characteristics of the application domain that make the proposed approach applicable. Finally, the conclusion and the future work are given in Section 6.

2. PREFERENCES REPRESENTATION

In our approach, a travel product x is a vector in an n-dimensional vector space $X = \prod_{i=1}^{n} X_i$, where X_i is the domain of the i-th feature. X_i has one of the following data types: *real*, *finite integer*, *nominal (symbolic)*, *symbol-set*, or *text*.
Let's make an example from a restaurant recommendation problem and assume that restaurants are represented in the 6-dimensional vector space: $X =< Name, Type, Location, MaxCost, OpeningDays, Characteristics >$. For instance, the restaurant $x = (\text{"}LaBerta\text{"}, \{pizzeria\}, \text{"}Trento\text{"}, 20, \{7, 1\}, \{parking, smoking\_room\})$ has name $x_1 = \text{"}LaBerta\text{"}$, type $x_2 = \{pizzeria\}$, location $x_3 = \text{"}Trento\text{"}$, maximum cost $x_4 = 20$, opening days $x_5 = \{7, 1\}$, and characteristics $x_6 = \{parking, smoking\_room\}$.

Given this product representation, user's preferences are represented as a composite query containing three components: a *logical query*, a *favorite pattern*, and a *feature importance weights* vector.

- The *logical query* models conditions that must be satisfied by the recommended products. The logical query is a conjunction of logical constraints.

$$Q_L = \bigwedge_{j=1}^{m} c_j; \qquad (1)$$

where c_j is a constraint on a feature. A constraint involves only one feature, and a feature may appear only in one constraint. Each feature type has a corresponding constraint type. For example, the logical query $Q_L = (x_3 = \text{“}Trento\text{”}) \wedge (x_5 \supseteq \{7, 1\})$ means that the user wants to see only those restaurants in Trento that are open on Saturday and Sunday.

- The *favorite pattern* models wish conditions that the recommended products should satisfy. Differently from must conditions, wish conditions allow trade-offs to be made. The favorite pattern is represented in the same vector space of the travel products.

$$p \in X; X = \prod_{i=1}^{n} X_i \tag{2}$$

  A value $p_i = ?$ denotes that the favorite value of i-th feature is unknown or irrelevant. For example, the favorite pattern $p = (?, \{spaghetteria\}, ?, ?, ?, ?)$ means that the user prefers spaghetti restaurants to others, and has no “wish” preferences on other restaurant features.

- The feature importance weights model how much each feature is important with respect to the others. Weights are real numbers between 0 and 1 ($w \in [0, 1]^n$). For example, the weight vector $w = (0, 0.6, 0, 0.4, 0, 0)$ means that the user considers the restaurant type as the most important feature, then the cost, and the rest as not important.

3. THE RECOMMENDATION PROCESS

In our main application scenario, on-tour support is required when a traveller, who has possibly built a pre-travel plan before leaving, is at the selected destination (or on the move to). On-tour support is provided by our MobyRek system ([5]) in co-operation with a pre-travel planning aid system (NutKing [11]). NutKing is a web-based recommender system that combines content-based and collaborative-based filtering methods to build pre-travel recommendations. The co-operation between these two systems, MobyRek and NutKing, is based on the reuse of pre-travel information, in terms of knowledge of the users' decisions, in the new context of the on-tour support.

An on-tour recommendation session starts when an on-the-move traveller requests the MobyRek system to find some interesting travel product, and ends when the traveller either selects a travel product or quits the current session with no product selected. An on-tour recommendation session evolves in cycles; in each cycle a set of recommendations is produced and shown to the traveller. The problem solving architecture of the recommendation process consists of four logical components: Initialization, Interaction, Adaptation, and Retainment (Figure 1). In this paper, we focus the discussion on the first component, i.e., the Initialization, the other components are described in [5].

Given a user's request, MobyRek constructs an initial representation of the user's preferences by exploiting: a) the user's current space-time position, b) the pre-travel plan, c) past on-tour selections of a community of users, and d) explicit initial preferences. The system's initialization of the user's preferences, which is the topic of this paper, will be discussed in-depth in Section 4.

The system uses the initial representation of the user's preferences to compute a first recommendation set. In the computation of a recommendation, the system firstly filters out those products which do not satisfy the logical query (Q_L), and then ranks the Q_L-matching products according to their similarity to the pair $< p, w >$ (i.e., the favorite pattern and the feature importance weights vector, respectively). The ranking is done so that the more similar to $< p, w >$ a product is, the higher that product appears in the ranked list. In case of ties, the cheaper product is ranked higher. Only the k best products in the ranked list (i.e., those which satisfy Q_L and are most similar to p) are shown to the user as recommendation result for the current cycle. The cut-off value (k) depends on the screen size of the traveller's mobile device and should be determined to avoid scrolling.

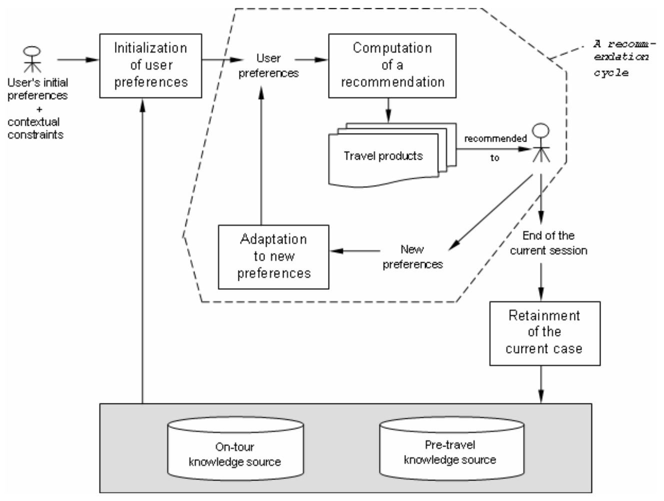

User's initial preferences + contextual constraints
Initialization of user preferences
User preferences
Computation of a recommendation
Travel products
recommended to
A recommendation cycle
Adaptation to new preferences
New preferences
End of the current session
Retainment of the current case
On-tour knowledge source
Pre-travel knowledge source

Figure 1: MobyRek recommendation model.

At a cycle, the user is supposed to browse the recommended products and see the details of these products. In response the user can execute one of the three actions: selection, critique, or quit.

A *selection* action is done when the user is satisfied with one of the recommended products and accepts the suggestion. This product is therefore added to his travel notes, and the current session ends successfully.

A *critique* action is done when the user is somewhat interested in one of the recommended products, but one (or some) feature(s) of this product does not completely satisfy him. Hence, the user criticizes the product, and specifies his preference on these unsatisfactory features (e.g., “I want restaurants cheaper than this one”). Such critiques help MobyRek to adapt the previous representation of the user's preferences, and to compute a new recommendation set, which is closer to the user's needs, based on this adapted representation. The adaptation depends on the type of the critique expressed by the user and on the type of the criticized feature.

A *quit* action happens when none of the products recommended so far satisfy the user, and he does not proceed any further. The current session terminates with a failure.

When an on-tour recommendation session finishes, either successfully or with a failure, it is retained as a case for future reference. In this way, all past recommendation sessions can be exploited by the system in the future. In particular, when an on-tour recommendation session finishes it is always retained as an on-tour case in the MobyRek case base; and if the user already built a pre-travel plan a link to the corresponding pre-travel case in the NutKing case base is established.

On-tour cases are represented by four components.

$$CB^{on-tour} = pT \times Q^0 \times SoC \times Acceptance \qquad (3)$$

where pT models a link to the pre-travel plan (if existing), Q^0 models the initial representations of user's preferences, SoC models the sequence of critiques in a recommendation session, and $Acceptance$ models final recommendation results.

Pre-travel cases ([11]) are represented as follows:

$$CB^{pre-travel} = TW \times TB \times U \times R \qquad (4)$$

where TW models user wishes, i.e., general preferences and user queries; TB models the travel bag, i.e. the travel products selected in the pre-travel stage; U contains some socio-demographic user data; and R models users' subjective evaluations, i.e., users' ratings to the selected products which are contained in the travel bag.

4. USER PREFERENCES INITIALIZATION

In the MobyRek recommendation model (Figure 1), the quality of the first recommendations plays a very important role because it heavily affects the user's decision. If the first recommended products are suitable, or at least relevant, then it is likely that the user will go on. Otherwise, the current recommendation session may stop with a failure even at the first cycle.

In MobyRek the initialization of the traveller's preferences is executed whenever the user requests a travel product hence starting a new recommendation session. We propose to exploit the following knowledge sources during the initialization process:

- The space-time constraints extracted from user's location and access time information.
- The pre-travel plan, of the current trip, already built by the user (i.e., using the pre-travel planning system NutKing) and the pre-travel plans of other similar users.
- The products/services selected by the user, and other similar users, in their past usage of MobyRek.
- The preferences explicitly specified by the user at the time of the request.

We note that the knowledge sources available for the MobyRek system's initialization are not only those related with past system usage, but also information about the current context, i.e., the user's space-time position and explicit preferences specified by the user.

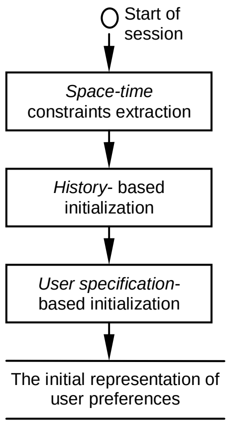
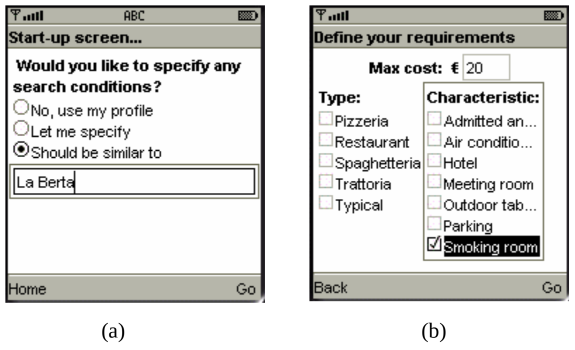

Start of session
Space-time constraints extraction
History- based initialization
User specification-based initialization
The initial representation of user preferences

Figure 2: Preferences initialization process.


Start-up screen...
Would you like to specify any search conditions?
○ No, use my profile
○ Let me specify
● Should be similar to
La Berta
Home Go

(a)

Define your requirements
Max cost: € 20
Type: ☐ Pizzeria ☐ Restaurant ☐ Spaghetteria ☐ Trattoria ☐ Typical
Characteristic: ☐ Admitted an... ☐ Air conditio... ☐ Hotel ☐ Meeting room ☐ Outdoor tab... ☐ Parking ☑ Smoking room
Back Go

(b)

Figure 3: Preferences initialization: Screenshots from MobyRek GUI.

MobyRek user's preferences initialization process is divided into three phases (see Figure 2). In the first phase, the system extracts the *space-time constraints* from the user's current context. In the second phase, the system exploits the knowledge contained in the user's past on-tour cases and pre-travel plan (of the current trip) in order to build the *history-based representation* of the user's preferences. In the third phase, the history-based representation is partially overwritten by the *initial preferences explicitly specified by the user.* Note that the third phase is performed (i.e., activated) only if the user is willing to specify some initial preferences at the time of the request.

The initialization process of user's preferences representation is depicted in Figure 3. Users are provided different options for controlling the MobyRek system's initial search. The first option in Figure 3a is for users who do not want to specify any initial conditions, whereas the last two options are for those who want to specify them and hence to have more control on the system's search process. Note that this is the user interface of the previous version of MobyRek that does not integrate the above mentioned information sources, but can illustrate how some explicit preferences are acquired.

4.1 Space-Time Constraints Extraction

In a recommendation session, as we noted in the introduction, the user's context refers to all the information about the user that relates to the current information-search/ recommendation problem. Regarding the space-time location of the user, this can be automatically detected with the help of technologies, such as GPS, Wi-Fi, Bluetooth, Infrared, etc. In our prototype we use a GPS device via a Bluetooth connection with a Nokia 6600 smart phone.

After detection, space-time information must be validated by the user to assure that this is correct and that the user is not searching for recommendations applicable to another day or place. When space-time information is undetectable, incorrectly detected, or unreliably detected, then the user is supposed to manually provide the correct information.

In mobile recommender systems, the space-time information plays an important role in limiting the system's search to only those products which are relevant to the user's current context. For example, if a user is in a city and searches for a restaurant for lunch, the user of course does not care for restaurants in other cities. Because of this, the space-time information is encoded into the logical query (Q_L) to discard such products that do not satisfy the space-time constraints (e.g. restaurants not open or too far from the user's position).

4.2 History-based Initialization

This initialization procedure is automatically done by the system, without any interaction with the user. The system exploits users' past recommendation sessions to find the past situation most similar to the user's current one, and then extracts the preference pattern from this (past) recommendation session.

Since the preference pattern is extracted from a similar session, these (old) preferences may be incorrect. For example, let's assume that a user had lunch at a restaurant x^* when he visited Trento. After a while, he returns to Trento, and searches again for a restaurant. It is plausible that the user would like a restaurant having similar features to that of x^*, but less likely that he goes back to x^* again. Therefore, in our approach the history-based extracted preferences are considered as wish conditions rather than "must have" constraints. Hence, the history-based initialization phase modifies the favorite pattern (p) and the feature weights vector (w), but does not affect the logical query (Q_L).

4.2.1 Favorite Pattern Initialization

In the remainder of this subsection, we shall call T_1 the product type (e.g., restaurant) the user is searching.

We now describe how the favorite pattern (p) is initialized in MobyRek by exploiting past recommendation sessions. The system checks if the user 1) had built a pre-travel plan[1] for the current trip, and/or 2) has ever accepted a T_1-typed on-tour recommendation in a previous trip[2]. Four cases are possible (see Figure 4).

[1]A pre-travel plan is represented as a case in the pre-travel case base, as shown in Equation 4.

[2]Note that, as discussed in Section 3, *not* all on-tour recommendation sessions terminate successfully (i.e., with a product selection).

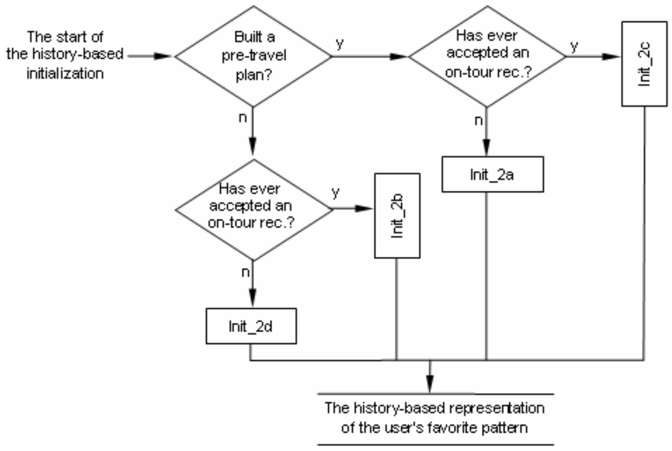

The start of the history-based initialization
Built a pre-travel plan?
y
n
Has ever accepted an on-tour rec.?
y
n
Init_2c
Init_2a
Has ever accepted an on-tour rec.?
y
n
Init_2b
Init_2d
The history-based representation of the user's favorite pattern

Figure 4: The history-based initialization of the user's favorite pattern.

In case the user had built a pre-travel plan, but has never accepted any T_1-typed on-tour recommendation, then the initialization procedure $Init\_2a$ is activated. In that, the system finds the user who built a pre-travel plan most similar to the current one, and the on-tour products selected by that "similar" user (if present) are exploited to initialize the current user's favorite pattern (p). Essentially, this is a collaborative filtering-based approach where, since we do not exploit products' rates ([8, 14]), only the first neighbor (user) is used to guess the recommendation. The initialization procedure $Init\_2a$ is detailed in the following algorithm.

```
procedure Init_2a(CB, Plan, $T_1$)
  Input:
      CB: the pre-travel case base.
      Plan: the current user's pre-travel plan.
      $T_1$: the type of products to be recommended.
  Output:
      A product pattern.

  Step 1. Identify the set of the candidate cases $CC$.
      CC = {c | (c ∈ CB) ∧
                    (c links to a $T_1$-typed on-tour selection)}
  Step 2. For each of the candidate cases, compute its similarity to
  the user's pre-travel plan.
      $Sim(c, Plan)$; where $c \in CC$
  Step 3. Choose among the candidate cases the one that has the
  highest similarity value.
      $c^* = argmax_{c \in CC}\{Sim(c, Plan)\}$
  Step 4. Return the product $x^*$ which is the ($T_1$-typed) on-tour
  selection corresponding to the case $c^*$.
end procedure
```

In the procedure $Init\_2a$ above, Step 2 executes a similarity computation between two pre-travel plans (i.e., cases). This similarity computation is discussed in detail in [11, 12]. The favorite pattern (p) is then initialized by the product x^* (returned by Step 4).

$$p_i = x_i^*; \qquad \forall i, i = 1..n \tag{5}$$

In case the user did not build a pre-travel plan, but accepted in a previous session some T_1-typed on-tour recommendation(s), the initialization procedure $Init\_2b$ is activated. This procedure retrieves from the on-tour case base (see Equation 3) the T_1-typed product most recently accepted by the user. The user's favorite pattern (p) is initialized by this product, using Equation 5.

In case the user had built a pre-travel plan for the current trip and also accepted some T_1-typed on-tour recommendation(s), the initialization procedure $Init\_2c$ is activated. In this procedure, the user's favorite pattern (p) is initialized by an average combination of the two favorite patterns:
. p^{pre} (returned by the procedure $Init\_2a$); and
. p^{on} (returned by the procedure $Init\_2b$).
In particular, the favorite pattern (p) is initialized using the following formula. For each feature f_i (i=1..n) :

$$p_i = \begin{cases} \lceil (p_i^{pre} + p_i^{on})/2 \rceil & \text{if } f_i \text{ is finite integer;} \\ (p_i^{pre} + p_i^{on})/2 & \text{if } f_i \text{ is real;} \\ \begin{cases} p_i^{pre}, & \text{if } p_i^{pre} = p_i^{on}; \\ ?, & \text{otherwise.} \end{cases} & \text{if } f_i \text{ is nominal;} \\ \begin{cases} (p_i^{pre} \cap p_i^{on}), & \text{if } (p_i^{pre} \cap p_i^{on}) \neq \emptyset; \\ ?, & \text{otherwise.} \end{cases} & \text{if } f_i \text{ is symbol-set.} \end{cases} \tag{6}$$

Finally, if the user neither built a pre-travel plan nor has ever accepted any T_1-typed on-tour recommendation, the initialization procedure $Init\_2d$ is activated. In that, the user's favorite pattern (p) is initialized as a blank one.

$$p_i = ?; \qquad \forall i, i = 1..n \tag{7}$$

The element value p_i=? means that the favorite value for feature f_i is unknown. We note that this "special" value ("?") has minimal similarity with all the other values.

4.2.2 Feature Importance Weights Initialization

The feature importance weights (w) are initialized exploiting the history of the user's interactions with MobyRek. The basic idea is that the initial (in a session) weight of a feature is proportional to the frequency of user critiques to that feature. That is, the more a feature has been criticized (by the user) the more that feature is important (to him). We recall that critiques are given by users to express their preferences regarding the system's recommended products (see Section 3).

In the remainder of this subsection, an on-tour recommendation session is called session for short, and the following notations are applied.

U: the set of users who have expressed at least one critique;

$S(u_j)$: the set of all (historically ordered) sessions of user u_j ($u_j \in U$);

$w_i(u_j, s_k)$: the importance weight of the i-th feature, computed using the data in session s_k of user u_j ($u_j \in U$, and $s_k \in S(u_j)$);

$w_i(u_j)$: the importance weight of the i-th feature, computed using the data of all sessions of user u_j ($u_j \in U$); and

$w_i(U)$: the importance weight of the i-th feature, computed using the data of all sessions of all users.

Given a user u_j, the feature weights initialization is computed using either the data of the user's sessions or the data of all users' sessions. If the number of critiques given by the user is large enough, the feature weights initialization is done at a user level. Otherwise, the system refers to all users' sessions and hence, the feature weights initialization is done at the population level.

Firstly, the system computes the importance weight of feature f_i, given session s_k of user u_j.

$$w_i(u_j, s_k) = \frac{1}{\lambda_k} . \sum_{l=1}^{\lambda_k} \frac{Ctz(f_i, u_j, c_l)}{\alpha^{(\lambda_k - l)}}; \tag{8}$$

where:
c_l: a recommendation cycle of session s_k ($c_l \in s_k$);
λ_k: the length (i.e., the number of recommendation cycles) of session s_k;
$Ctz(f_i, u_j, c_l)$ =1, if at recommendation cycle c_l user u_j had criticized on feature f_i,
=0, otherwise;
α: a parameter used to increase ($\alpha > 1$), or decrease ($\alpha < 1$), the importance of latest critiques (i.e., those appear later in the session s_k).

Let's make an example referring to user u_1 and his session s_2. For sake of simplicity, let's assume that there are six features, f_i ($i = 1 \ldots 6$). If the sequence of features criticized (by user u_1) in session s_2 is $[f_1 \rightarrow f_2 \rightarrow f_4 \rightarrow f_3 \rightarrow f_2]$, then the importance weight of feature f_2 (given session s_2 of user u_1) is:
$w_2(u_1, s_2) = \frac{1}{5}[\frac{0}{\alpha^4} + \frac{1}{\alpha^3} + \frac{0}{\alpha^2} + \frac{0}{\alpha} + 1]$.

Secondly, the system computes the importance weight of feature f_i over all the sessions of user u_j.

$$w_i(u_j) = \frac{1}{\parallel S(u_j) \parallel} . \sum_{k=1}^{\|S(u_j)\|} \frac{w_i(u_j, s_k))}{\beta^{(\|S(u_j)\| - k)}}; \tag{9}$$

where $\beta(> 1)$ is a parameter that shapes how fast the importance of an old session decreases with the time. It is assumed that a more recent session should play a more important role than an older one [16]. Note that in Equation 9, s_1 is the oldest session.

In case the number of critiques the user u_j had given (in his past on-tour recommendation sessions) is large enough (e.g., this number is greater than a predefined threshold value), all elements of the feature weights vector (w) are initialized using Equation 9.

Otherwise, if the number of user u_j's past critiques is insufficient (new user), then the system exploits the usage data of all the users who had expressed at least one critique. In this case, all elements of the feature weights vector (w) are computed using the following formula:

$$w_i(U) = \frac{1}{\parallel U \parallel} . \sum_{u_j \in U} w_i(u_j) \tag{10}$$

For a new user, the MobyRek system has no (or unreliable) knowledge of the user's preference about the relative importance of the features. Therefore, MobyRek assumes that the (new) user has the "default" preference about features importance which is constructed by exploiting the usage data of the community of users (as in Equation 10). This "default" preference about features importance will be refined as the user interacts with MobyRek.

Once all elements of the feature weights vector (w) are initialized, using either Equation 9 or Equation 10, then their values are normalized with the length of w.

4.3 User Specification-based Initialization

In the third phase of the initialization process (see Figure 2), there are two situations corresponding to whether or not the user is willing to specify some initial preferences.

In case the user does not want to specify any initial preferences, the initialization process finishes and the initial representation of the user's preferences contains:
. the logical query (Q_L) with the space-time constraints;
. the favorite pattern (p) derived from the history-based representation of the user's preferences; and
. the feature weights vector (w) derived from the history-based representation of the user's preferences.

In case the user would like to specify some preferences, then these explicit preferences must overwrite any guess of the system.

Given an explicitly stated preference, its type determines which of the three components of the user preferences representation (Q_L, p, w) must be overwritten. In our approach, users are provided with two options for explicitly specifying their initial preferences: *sample-oriented* or *feature-oriented.*

In a sample-oriented start-up (i.e., the third option in Figure 3a) the user can locate those travel products which are similar to a known (i.e., consumed) one. For example, the user may want to find those restaurants similar to a particular restaurant at which he had lunch last week. Hence, the sample product specified by the user serves as a starting point for the system's search process [2]. The initialization process finishes and returns the initial representation of the user's preferences which contains:
. the logical query (Q_L) comprising the space-time constraints;
. the favorite pattern (p) replaced by the sample product (using Equation 5); and
. the feature weights vector (w) taken from the history-based representation of the user's preferences.

In a feature-oriented start-up (see Figure 3b), the user can specify the values of some particular product features. For instance, he can set a maximum price to 20 Euro. Given the preferences specified by the user, the system performs the following tasks.

- For each of the explicit preferences, either the logical query (Q_L) or the favorite pattern (p) is overwritten. In the current version of MobyRek, for each preference specified by the user either the logical query or the favorite pattern is modified. Hence, for instance, preferences on the price are converted in logical constraints and those on restaurant characteristics are encoded into the favorite pattern (Figure 3b). In the next version, we intend to support users in specifying if a preference has to be interpreted as a must or a wish condition (whatever feature is involved). This should help MobyRek to better capture user's initial preferences.
- The feature weights vector (w) is overwritten so that the system increases the weight value of those features explicitly specified by the user. This is because these features are more important to the user than the remaining ones. Let's call $w^{history}$ the feature weights vector returned by the second (i.e., history-based) phase of the initialization process. By the user's explicitly initial specification, the feature weights vector (w) is overwritten using the following formula.

$$w_i = \begin{cases} \gamma.max_{k=1..n}\{w_k^{history}\}, & \text{if } i\text{-th feature is specified;} \\ w_i^{history}, & \text{if otherwise.} \end{cases} \tag{11}$$

where $\gamma(>1)$ is a parameter which defines how much different (more relevant) should be the importance of the features specified by the user and those not. The feature weights vector (w) is then normalized.

5. OPEN ISSUES AND DISCUSSION

The approach proposed in the previous Section assumes that there is a relationship between the preferences expressed in past recommendation sessions and those expressed in the current session. This is the reason why, in the proposed approach, the initial representation of the user's preferences is firstly built automatically by the system, and afterward is overwritten by the user's explicit preferences.

However, for application domains where user's explicit preferences could be very different from those expressed in the past; this assumption about a relationship between past sessions and the current one could not hold.

Let's illustrate this point with an example. Let's suppose that the system had produced the history-based representation of the user's preferences consisting of:
. the logical query (Q_L) containing the space-time constraints;
. the favorite pattern $p^{history} = (p_1, \ldots, p_n)$; and
. the weights vector $w^{history} = (w_1, \ldots, w_n)$.

Now, let's suppose that the user, at the beginning of the interaction, specifies two wish conditions with respect to the first and the second features. According to the overwriting strategy proposed in Section 4, the initial representation of the user's preferences becomes:
. the logical query (Q_L) still containing the space-time constraints;
. the favorite pattern is $p^{overwritten} = (p'_1, p'_2, p_3, \ldots, p_n)$; and
. the weights vector is $w^{overwritten} = (w'_{12}, w'_{12}, w'_3, \ldots, w'_n)$.

However, the above representation may not reflect exactly the user's preferences in the current recommendation session, and maybe a better representation could be:
. the logical query (Q_L) still containing the space-time constraints;

. the favorite pattern is $p = (p'_1, p'_2, ?, \ldots, ?)$; and
. the weights vector is $w = (0.5, 0.5, 0, \ldots, 0)$.

The argument is raised because the preferences on the features not explicitly mentioned by the user (i.e., from the third to the n-th), which are automatically initialized by the system, may be dissimilar from the user's current needs. Hence, their usage in the favorite pattern (p) and in the feature weight vector (w) may lead the system's search to a misleading direction.

Because of this, for those application domains where the user's preferences in the current session are not correlated with those expressed in past interactions, it is better to discard any system guess when the user expresses some preferences, and rely totally on these explicit preferences.

6. CONCLUSION AND FUTURE WORK

An important and common problem for every recommender system is the precision of the system's initial guess of user's preferences. How this problem is addressed affects significantly the success of recommendation sessions. In this paper, we have proposed an approach for the initialization of the user preferences representation which 1) exploits different sources of user related knowledge and 2) combines a persistent user model with a context-dependent one.

We are now implementing the proposed approach for user preferences initialization into MobyRek. We shall carry out experiments to validate the appropriateness of this approach in the context of a critique-based recommender system. Regarding the evaluation of the quality of first-cycle recommendations, our future experiments will address the following questions. Firstly, how likely it is that, after a first recommendation cycle, the user will select a product, or go on to a next cycle, or quit the interaction? Secondly, how effective it is the proposed approach compared to other solutions that: a) exploits only users' persistent model, b) exploits only users' context-dependent model, and c) randomly generates an initial recommendation set. A secondary issue will be the analysis of the contribution to the recommendation quality brought by the exploitation of pre-travel plans (i.e., *Init_2a* and *Init_2c* in Figure 4). The validation here proposed will be performed both 1) by involving real users and 2) by means of simulation procedures that statistically model the user action, and are parameterized by the data acquired during the user empirical evaluation.

7. REFERENCES

[1] G. D. Abowd, A. K. Dey, R. J. Orr, and J. Brotherton. Context-awareness in wearable and ubiquitous computing. In *Proceedings of the 1st International Symposium on Wearable Computing, ISWC'97*, pages 179–180, 1997.

[2] R. Burke. Knowledge-based recommender systems. *Encyclopedia of Library and Information Science*, 69(32):180–200, 2000.

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

[4] K. Lang. Newsweeder: Learning to filter news. In *Proceedings of the 12th International Conference on Machine Learning, ICML'95*, pages 331–339, 1995.

[5] Q. N. Nguyen, D. Cavada, and F. Ricci. On-tour interactive travel recommendations. In *Proceedings of the 11th International Conference on Information and Communication Technologies in Travel and Tourism, ENTER2004*, pages 259–270, 2004.

[6] D. M. Nichols. Implicit rating and filtering. In *Proceedings of the 5th DELOS Workshop on Filtering and Collaborative Filtering*, pages 31–36, 1997.

[7] M. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27:313–331, 1997.

[8] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the Conference on Computer Supported Cooperative Work, CSCW'94*, pages 175–186, 1994.

[9] P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[10] F. Ricci. Travel recommender systems. *IEEE Intelligent Systems*, 17(6):55–57, 2002.

[11] F. Ricci, B. Arslan, N. Mirzadeh, and A. Venturini. Itr: A case-based travel advisory system. In *Proceedings of the 6th European Conference on Case Based Reasoning, ECCBR'02*, pages 613–627, 2002.

[12] F. Ricci, A. Venturini, D. Cavada, N. Mirzadeh, D. Blaas, and M. Nones. Product recommendation with interactive query management and twofold similarity. In *Proceedings of the 5th International Conference on Case-Based Reasoning, ICCBR'03*, pages 479–493, 2003.

[13] E. Rich. User modeling via stereotypes. *Cognitive Science*, 3(4):329–354, 1979.

[14] T. Tran and R. Cohen. Hybrid recommender systems for electronic commerce. In *Knowledge-Based Electronic Markets, Papers from the AAAI Workshop, AAAI Technical Report WS-00-04*, pages 78–83, 2000.

[15] A. M. A. Wasfi. Collecting user access patterns for building user profiles and collaborative filtering. In *Proceedings of the 5th International Conference on Intelligent User Interfaces, IUI'99*, pages 57–64, 1999.

[16] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101, 1996.