IEEE Intelligent Systems
www.computer.org/intelligent

Acquiring and Revising Preferences in a Critique-Based Mobile Recommender System

Francesco Ricci and Quang Nhat Nguyen

Vol. 22, No. 3
May/June 2007

Acquiring and Revising Preferences in a Critique-Based Mobile Recommender System

Francesco Ricci and Quang Nhat Nguyen, *Free University of Bozen-Bolzano*

Few Web-based recommender systems have been designed for mobile users. A critique-based recommendation methodology aids the acquisition and revision of user preferences in a mobile recommender system.

Many e-commerce Web sites offer numerous services, so a product search could return an overwhelming set of options. Without system support, filtering irrelevant products, comparing alternatives, and selecting the best option can be difficult or impossible—especially for users connecting to the Web through a mobile device. Recommender systems are personalized applications that can address this problem and suggest products that suit the user's needs.[1]

However, a major problem in designing effective recommender systems is finding an effective, reliable method for acquiring and revising user preferences. Users rarely know all their preferences at the start. They usually form them during decision making, so they should be allowed to revise them during the process, using a range of interaction styles.

We designed a product recommendation methodology and implemented it in MobyRek, a mobile-phone recommender system that helps users search for travel products. MobyRek supports limited asking and answering of questions and is based mostly on critiques.[2–4] This on-tour (that is, during vacation) support system cooperates with NutKing, a Web-based recommender system that helps users build pretravel plans. The on-tour support is exploited when a mobile traveler (who might have created a pretravel plan) is on the way to or at the selected destination.

Our design meets two general requirements. First, the product recommendations are relevant to the user's specific preferences. Second, the user-system interaction is simple, requiring minimal time to obtain a useful recommendation.

Our recommendation methodology

In our approach, the user can critique a system recommendation at each cycle (see the "Related Work in Product Recommendation" sidebar for previous approaches). System recommendations appear in the displayed ranked results list. A user makes a critique when one feature of a recommended product is somewhat unsatisfactory or very important. So, the user specifies his or her preference (for example, "I want a less expensive restaurant than this") or makes a positive comment ("I like dining in the garden"). Such feedback helps the system adapt how it represents the user's preferences and compute a new recommendation set that's closer to the user's needs. The adaptation depends on the critique type (that is, "must" or "wish" critiques), the criticized feature (numeric or nominal—that is, taking values in a prespecified finite set of possibilities), and the critique operator (for example, "smaller" in the critique "I want a less expensive restaurant than this"). So, adaptation rules are problem and user dependent.

Our model provides system support for on-the-go consumers selecting products or services. To ensure that recommendations are relevant to user needs and that user interaction is simplified, the model

1541-1672/07/$25.00 © 2007 IEEE
Published by the IEEE Computer Society

Related Work in Product Recommendation

In traditional product-recommendation approaches, systems collect user preferences by explicitly asking the user. The system then exploits the acquired preferences to activate the specific recommendation algorithm. Although these preferences tend to be reliable, this approach has several disadvantages. First, users must have enough knowledge about the problem domain to make their preferences explicit according to the product model (for example, product attributes) the system supports. Second, uncertain or incomplete preferences can become clear as users interact with the system and better understand what they want and what products are available, meaning that they can't be asked before the system provides some recommendations. Third, many users are reluctant to reveal their preferences until they receive some benefit from the system.

To overcome these problems, researchers have proposed methodologies for deriving user preferences by analyzing a user's navigation behavior on a mobile device.[1] Although these approaches require considerably less user effort, they must interpret the user action (for example, clicking on a hyperlink) and translate it into the user's preferences. Moreover, implicitly stated preferences tend to be imprecise and noisy. Precisely collecting users' preferences while minimizing their effort is a major challenge for recommender systems in mobile devices. The system design must find the right balance between having precise information and the cost of acquiring it. Explicitly querying the user focuses on the former, while user navigation mining focuses on the latter.

Another approach that has recently received much interest elicits user preferences through structured human-computer dialogue.[2–4] A user's goal when participating in such a dialogue is to find desired products, and the system's role is to help the user quickly and effectively find them. In conversational recommender systems, for example, at each interaction cycle, the system might either ask the user for a preference or propose a product. The user either answers the system's question or criticizes its proposal. In particular, the system can raise a selective question to refine user preferences when the number of candidate products is unsatisfactory—for example, when the system found no items (or too many items). When replying, the user can indicate, remove, or modify some conditions so that the system can retrieve a better result set.

Although many successful Web-based recommender systems exist,[5] only a few have been designed for mobile users[1,6–8] and, to our knowledge, none is conversational. Moreover, these applications run mostly on PDAs, not on mobile phones. Mobile phones have smaller screens, limited computing power, and limited keypads. Designing mobile applications presents complex technical and usability issues. A mobile system must implement a recommendation methodology that overcomes the mobile usage environment's limitations and accommodates mobile users' behavior.

References

1. M.D. Dunlop et al., "Focussed Palmtop Information Access through Starfield Displays and Profile Matching," *Proc. Workshop Mobile and Ubiquitous Information Access*, LNCS 2954, Springer, 2004, pp. 79–89.
2. R. Burke, "Interactive Critiquing for Catalog Navigation in E-Commerce," *Artificial Intelligence Rev.*, vol. 18, nos. 3–4, 2002, pp. 245–267.
3. L. McGinty and B. Smyth, "Deep Dialogue vs. Casual Conversation in Recommender Systems," *Proc. Workshop Personalization in eCommerce, 2nd Int'l Conf. Adaptive Hypermedia and Web-Based Systems* (AH 02), Univ. of Malaga Press, 2002, pp. 80–89.
4. M. Torrens, B. Faltings, and P. Pu, "Smartclients: Constraint Satisfaction as a Paradigm for Scaleable Intelligent Information Systems," *Constraints*, vol. 7, no. 1, 2002, pp. 49–69.
5. G. Adomavicius and A. Tuzhilin, "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions," *IEEE Trans. Knowledge and Data Eng.*, vol. 17, no. 6, 2005, pp. 734–749.
6. L. Ardissono et al., "Intrigue: Personalized Recommendation of Tourist Attractions for Desktop and Handset Devices," *Applied Artificial Intelligence*, vol. 17, nos. 8–9, 2003, pp. 687–714.
7. H.-W. Tung and V.-W. Soo, "A Personalized Restaurant Recommender Agent for Mobile E-Service," *Proc. IEEE Int'l Conf. E-technology, E-commerce, and E-service* (EEE 04), IEEE CS Press, 2004, pp. 259–262.
8. M. van Setten, S. Pokraev, and J. Koolwaaij, "Context-Aware Recommendations in the Mobile Tourist Application COMPASS," *Adaptive Hypermedia and Adaptive Web-Based Systems*, LNCS 3137, Springer, 2004, pp. 235–244.

- integrates long-term and session-specific preferences,
- integrates pretravel preferences collected via a Web-based recommender system, and
- exploits a critique-based conversational approach.

In that model, long-term preferences are collected both by mining past interactions with the mobile and pretravel recommender systems and by letting users explicitly define a set of stable preferences (for example, the payment method). Moreover, we've introduced a type of critique that lets users express additional session-specific preferences and specify their strength (must or wish).

We conducted an empirical evaluation with users, which proved the recommendations' relevance and the system's usability. Results show that eliciting user preferences through critiques and integrating various knowledge sources (such as pretravel decisions, similar travels, and explicit feedback) in a mobile application is effective and can help systems acquire users' personal information and deliver relevant recommendations.

You can't use critiques as the only method for acquiring user preferences—you must give users the option to provide their preferences explicitly, especially when interaction begins. Also, even in human-computer interaction with mobile applications, you can't achieve simplicity and usability by sacrificing the application's functionality. Users require complete, complex functionality from mobile applications.

The product and preferences models

Our approach represents a product as a feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, where a feature value x_i can be numeric, nominal, or a set of nominal values. For example, the representation of the restaurant $\mathbf{x}$ = (*Bouganville*, *1601*, {*pizzeria*}, *20*, {*air_conditioned*, *smoking_room*}, {*2*, *3*, *4*, *5*, *6*, *7*}, {*credit_card*}) means that

- the name (x_1) is Bouganville,
- the distance from the user (x_2) is 1,601 meters,
- the restaurant type (x_3) is a pizzeria,
- the average cost (x_4) is 20 euros,
- the characteristics (x_5) are air conditioned and smoking room,
- the days open (x_6) are 2, 3, 4, 5, 6, and 7 (that is, from Tuesday to Sunday), and
- the accepted method of payment (x_7) is credit card.

We can derive the user preferences model from multiple knowledge sources, but all of them are ultimately fully encoded in a product search query. Moreover, we distinguish between long-term and session-specific user preferences (see figure 1). The system derives the user's long-term or stable preferences, such as a preference for nonsmoking rooms, from several recent interaction sessions between the user and the system. These stable preferences remain true throughout the sessions. In contrast, session-specific preferences, such as a desire to eat pizza, are transient. In this article, we focus on session-specific preferences; our previous work discusses a proposed methodology for integrating both kinds of preferences.[5]

Session-specific preferences include both *contextual preferences* (such as space-time constraints) and *product feature preferences*. Contextual preferences characterize the user-request context, whereas product feature preferences express a user's tastes. In the restaurant recommendation problem, for example, space-time constraints guarantee that the recommended restaurant is open on the day of the request and isn't too far from the user. Product feature preferences might state that the user, for example, seeks low-cost restaurants or prefers pizza.

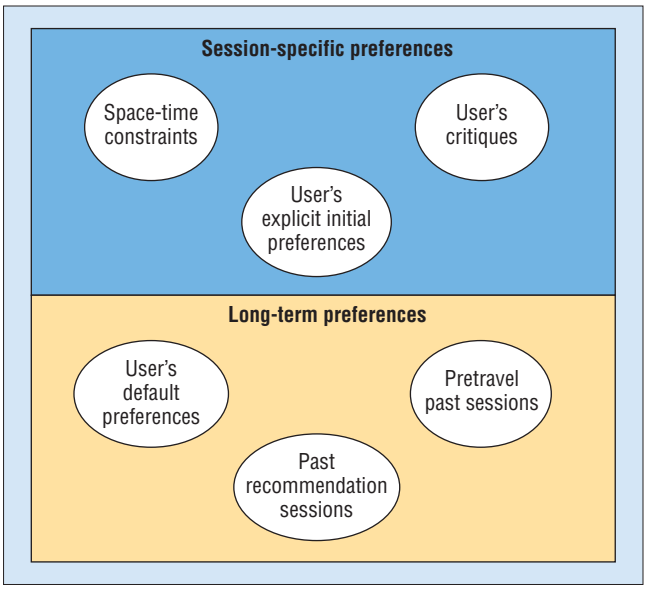


Figure 1. Session-specific and long-term user preferences.

The system acquires session-specific preferences in just two ways: the user's initial specifications and the user's critiques during the recommendation session. (We discuss preferences initialization and revision later.) Irrespectively from how a user generates or modifies a search query q, the query encodes the current system model of the user's preferences. This model, $q = (q_l, \mathbf{p}, \mathbf{w})$, has three components:

- The *logical query*, $q_l = (c_1 \wedge c_2 \wedge \ldots \wedge c_m)$, models the identified conditions that the recommended products must satisfy. It consists of logical constraints $c_1 \ldots c_m$, where each item refers to a single feature. Feature types have different constraint types.
- The *favorite pattern*, $\mathbf{p}$, encodes conditions that the recommended products should match as closely as possible. These wish conditions let the system make trade-offs. The favorite pattern is represented in the same vector space of the product representation—that is, $\mathbf{p} = (p_1, p_2, \ldots, p_n)$.
- The *feature importance weights vector*, $\mathbf{w} = (w_1, w_2, \ldots, w_n)$, models how important each feature is to the user with respect to the others, where $w_i \in [0, 1]$ is the importance weight of the ith feature.

For instance, the query

$$q = (q_l, \mathbf{p}, \mathbf{w}) = \big((x_2 \le 2{,}000) \wedge (x_6 \supseteq \{6,7\})\big), \big(?, ?, \{spaghetteria\}, ?, ?, ?, ?\big), (0, 0, 0.6, 0.4, 0, 0, 0))$$

models a user seeking restaurants within 2 km from his current position that are open on Saturdays and Sundays—preferably spaghetti restaurants. The most important features are the restaurant type followed by the cost; the user is indifferent about the other features. The system exploits the importance weights only in the context of the favorite pattern because the displayed products always satisfy the logical queries (we discuss this in more detail later). Moreover, the system uses weights when it must identify trade-offs—that is, when the system can't simultaneously satisfy the user's must preferences and must relax some of them to retrieve products.

The recommendation process

A recommendation session begins when a mobile user asks the system for a product recommendation, and it ends when the user selects a product or terminates the session without making a selection. A recommendation session evolves in cycles (see figure 2). At each critique cycle, the system shows recommended products (see figure 3a) that the user can browse (see figure 3b) and criticize (see figure 3c). At the next cycle, if the user has expressed a critique but hasn't selected a product, the system computes a new recommendation list and shows it to the user.

At start-up, the system offers three options for preferences and search initialization (see figure 3d):

- "No, use my profile" lets the system automatically construct the initial search query by exploiting long-term preferences.
- "Let me specify" lets the user explicitly specify initial preferences.
- "Similar to" lets the user specify a known and favorite product as the starting point of the system's search.

Given a user's choice for preferences initialization, the system integrates user input and long-term preferences to build a case that models the user-system interaction and contains an initial search query. This case describes several components:

- products selected before travel (using NutKing),
- the user's contextual information (that is, the user's position and the time of the request),
- the user's default preferences (for example, a nonsmoking room),
- preferences that the user explicitly specified at the beginning of the session,
- the system's initial representation of the user query,
- the sequence of critiques that the user gave in the session, and
- the user's product selection at the end of the mobile session.

The system exploits the first four components to build the initial query, which is transparent to the user. As we mentioned earlier, the query contains three components: the logical query, the favorite pattern, and the feature importance weights.

Initializing the logical query exploits only the user's session-specific preferences, not the long-term ones. This avoids overestimating the importance of preferences that can be only partially true in the user's current session. So, the initial logical query encodes only the space-time constraints and the must conditions the user explicitly specified at the beginning.

Initializing the favorite pattern is a two-phase process. First, the system exploits the knowledge in past similar recommendation sessions and the user's default preferences stored in the mobile device's memory to build the user's long-term preference pattern, $\mathbf{p}'$. Second, the system integrates $\mathbf{p}'$ with $\mathbf{p}''$, the initial wish preferences the user explicitly specified, to compute $\mathbf{p}$. In this combination, $\mathbf{p}''$, if present, overwrites $\mathbf{p}'$ because explicit preferences should always be considered more reliable than those the system infers.

The first initialization phase (that is, the exploitation of past cases and default preferences) has three steps:

1. Finding the past on-the-move recommendation session that's most similar to the current session.
2. Extracting the product (restaurant) the user selected in the most similar session.

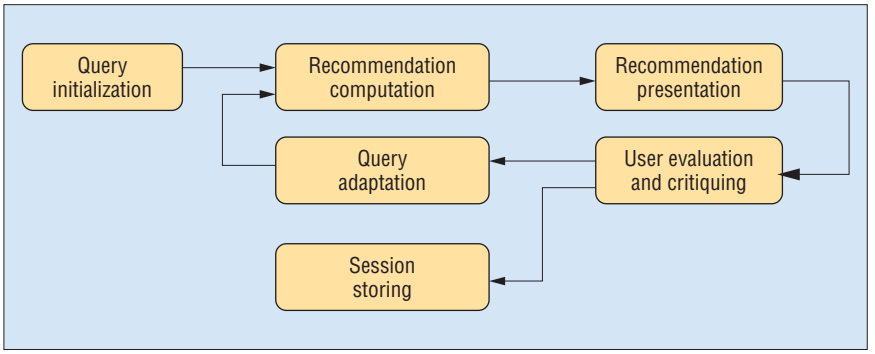


Figure 2. The major steps of the supported recommendation process.

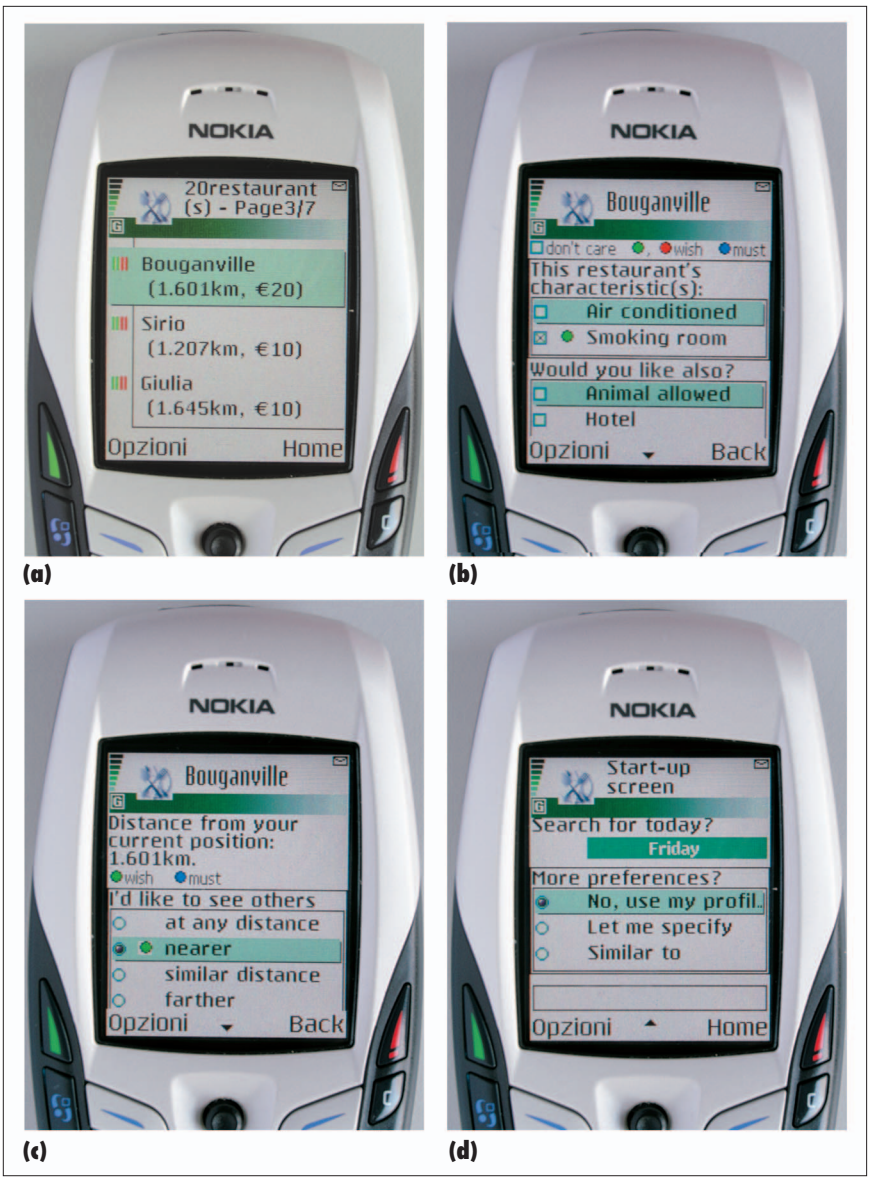


Figure 3. The MobyRek user interface (a) displays recommended products that the user can (b) browse and (c) criticize. At start-up, the system (d) offers options for preferences and initializing the search.

3. Merging the stable preferences to build the user's long-term preference pattern.

In this process, the whole case model contributes to the identification of a favorite pattern. So, for instance, the products selected before travel help the system identify a similar case and, consequently, compute a good favorite pattern. In other words, suppose user A selected a particular hotel before travel and that user B previously selected the same hotel and a restaurant. The system would consider that restaurant to be a good favorite pattern for user A. The favorite pattern has the features of a restaurant selected in another travel by a user who made similar pretravel selections, in a similar contextual situation, with similar default preferences.

Finally, the system initializes the feature importance weights vector by exploiting the history of user interactions with the system. A feature's initial importance weight is proportional to the frequency of user critiques of that feature. (A detailed description is outside the scope of this article; see our previous work for more information.[5])

Having built an initial query (that is, a logical component, favorite pattern, and feature weights), the system can now execute the query and compute a first recommendation list. In computing the list, the system first discards the products that don't satisfy the logical query q_l, then ranks the remaining products according to their similarity to the favorite pattern **p**. The more similar a product is to **p**, the higher it appears in the ranked list. Similarity is computed as the difference between the maximum similarity value (that is, 1) and the generalized weighted Euclidean distance between the pattern and the product. In this computation, **w** balances the contribution of different features and is found in the current user model—that is, in the current query definition.

In computing the recommendation list, if no products in the catalog satisfy q_l, the system searches for a minimum number of constraints that if discarded from q_l will make q_l satisfiable. In this relaxation, a constraint involving a less important feature is relaxed before one involving a more important feature, where importance is derived from past usage behavior and is stored in the vector **w**. The relaxation algorithm explores breadth-first all the possible subsets of constraints. First, it considers the relaxations involving a single constraint; if it finds no solution, it determines the relaxations involving two constraints (that is, the second level). For all the minimal successful relaxations (in term of the number of constraints relaxed), the system computes a risk function that estimates the risk to remove important user preferences:

$$Risk(rel, q_l) = \sum_{i \in F(rel)} w_i$$

where *rel* is a subquery of q (that is, the conjunction of constraints that are removed from q_l) and $F(rel)$ is the set of feature indexes of the features discarded by the relaxation *rel*. Next, the system chooses the relaxation that minimizes that risk function. It then converts the relaxed constraints to wish conditions and incorporates them in the favorite pattern (previous research presents similar approaches[2,6]). So, even if the system can't find products with the preferred feature values, it tries to minimize user dissatisfaction as much as possible. Our approach, based on extensive evaluation of all minimal relaxations, doesn't scale for more complex queries. Nevertheless, in the context of a mobile search, users typically require a limited number of must conditions, making our solution feasible in practice.

When the system shows the user the list of recommended products (see figure 3a), three situations can occur:

- After browsing detailed product information (see figure 3b), the user chooses one option and ends the session successfully.
- The user is unsatisfied with the recommended products and quits the session.
- The user is interested in a recommended product but isn't completely satisfied with it.

So, in the third situation, the user should critique the product—that is, further specify his or her preferences. Figure 3b shows that the user is interested in a smoking room. The user could also choose additional preferences such as "animal allowed" or "hotel" (that is, characteristics the product is missing) or request a restaurant that's closer (figure 3c).

When critiquing a recommended product, the user can assign a strength to the preference—that is, specify whether the preference is a must or a wish. This helps the system correctly exploit the user's critique. If the user specifies a must condition, the system zooms in a certain region of the product space. If he or she specifies a wish condition, the system refines the product ranking. After the user provides a critique, the system incorporates it in the query and updates the result list.

Even if the system can't find products with the preferred feature values, it tries to minimize user dissatisfaction as much as possible.

On the result list, each product has an icon to the left of the restaurant name, indicating how closely the product matches the user's preferences (see figure 3a). This helps the user quickly grasp how appropriate each recommended product is to the query and how the match between preferences and retrieved products changes in two successive cycles. Moreover, the products list displays the total number of recommendations and pages the user can assess.

The system groups all commands in a contextual menu ("opzioni"). Depending on the page, an appropriate (contextualized) set of system commands is available, and these commands are all grouped into that single menu. This makes the user interface consistent throughout the user's interaction session.

When a recommendation session ends, whether successfully or not, the system retains it as a case. So, the system can exploit past user recommendation sessions in making new recommendations for that user and similar future users.[5]

Critiques

When a user critiques a recommended product, he or she can focus on a particular feature. In general, when a critique on a product feature x_i is a must condition, the system updates the logical query component c_i. When a critique is a wish condition, the system includes

a new feature value p_i in the pattern component. So, for example, if the pattern was initially $\mathbf{p} = (?, 500, ?, ?, ?, ?, ?)$ and the user performs the critique in figure 3b, the pattern becomes $\mathbf{p} = (?, 500, ?, ?, \{smoking\_room\}, ?, ?)$. The system computes that feature's importance weight by exploiting the history of previous interactions. In general, features that are more often or more recently criticized have larger weights. See our previous work for details on computing and managing importance weights.[5]

In our approach, a critique has two roles. First, it can elicit a user preference that the product doesn't satisfy (enough), as in figure 3c. After critique incorporation and query reexecution, the system will likely discard the critiqued product or rank it lower. (Other systems also use critiques in this way.[2,4]) Second, a critique can express the user's interest in a feature the product has, as in figure 3b. In this case, the system will rank the critiqued product (and similar products) higher in the new ranked list.

The system supports three types of critiques.

NoPref critique

In this critique, the user states that he has no preference on a feature x_i. So, the system removes any logical constraint on x_i, sets p_i to "?," and sets w_i to a minimum value.

Must critique

A user states a must critique when she wants the system to recommend only products that satisfy the condition. The system uses such a critique to refine q_l and $\mathbf{w}$.

Must critiques have four operators: "greater," "equal," "smaller," and "contain." "Greater" and "smaller" are supported by numerical features, "equal" by both numerical and nominal features, and "contain" by nominal-set features only.

A must critique always creates a new logical constraint on the criticized feature. If q_l already has a constraint on the same feature, the two constraints are joined only if the union is satisfiable. Otherwise, the system keeps the most recent constraint and removes the old one.

For example, suppose that the user has previously specified a constraint on the restaurant cost $x_4 < 30$. Also assume that in the current cycle, she criticizes a recommended product's cost (10 euros), using the "greater" critique operator. Now the user's constraint on cost contained in q_l is $10 < x_4 < 30$. Moreover, when a must critique is incorporated in the query, the feature importance weight w_i increases, multiplied by a positive factor greater than one.

Wish critique

Such critiques include "greater is better," "equal is better," "less is better," and "contain is better." A critique stated as a wish condition always modifies $\mathbf{p}$. Imagine that the user states a "greater is better" critique (the system treats the other cases similarly). If the user states that he wishes x_i to be greater than e, the system sets p_i to $e + \delta_i$, where δ_i is a feature-dependent parameter. An analogous change occurs if the user states an "equal is better" or "less is better" critique. If the user states a "contain is better" critique on a nominal-set feature x_i and p_i was the favorite (set) value for that feature, the new set value becomes $p_i \cup s$, where s is the user's new favorite value.

When a query incorporates a wish critique on x_i, the feature importance weight w_i increases exactly as for a must critique.

Evaluation

We implemented our approach in MobyRek. We developed the system prototype for mobile phone users using Java 2 Micro Edition Mobile Information Device Profile 2.0 and performed a usability test to verify the prototype's functionality, efficiency, and convenience. The experiment involved 15 testers using a Nokia 6600 phone with a GPS receiver (for position detection). Nine testers had previously used a Nokia phone. All testers were 20 to 40 years old; nine were male and six were female.

We conducted the test in Trento, Italy, focusing on restaurant recommendations. We chose testers who had been living in Trento for some time because they could give us feedback and ratings on selected restaurants without having to visit them again. Moreover, to make the suggestions more useful, we added a routing function showing the path from the user's position to the selected restaurant.

The test procedure comprised three phases. During training, we introduced testers to the phone and the recommender system. In particular, we provided a sample recommendation session to illustrate system behavior. This phase typically lasted 15 minutes.

During testing, we asked testers to think about the characteristics they desired in a restaurant and then use the system to find such a product. To simulate a pretravel decision, we also asked testers to select a hotel from 10 choices. We used this choice to initialize the testers' pretravel plans before they began interacting with MobyRek. When testers selected a restaurant, we asked them to add it to their travel notes, which contain all travel products, services, and information that a user has selected for a trip.

During testing, we asked testers to think about the characteristics they desired in a restaurant and then use the system to find such a product.

During evaluation, testers evaluated the system's performance by completing a usability questionnaire. The questionnaire contained a free-text space for comments and a predefined list of 17 statements, many taken from the Post-Study System Usability Questionnaire.[7] Testers answered questions using a seven-point Likert scale where 1 is "strongly disagree" and 7 is "strongly agree."

Table 1 shows the testers' average ratings of the questionnaire statements. These ratings, which expressed the testers' subjective evaluation of the system's performance, confirmed our objective results. In particular, testers could effectively and quickly find their desired restaurant using the system (statements 3 and 4). They found the user-system interaction and the user interface pleasant and friendly. All testers found the critiquing function very useful and easy to use in helping them find their desired restaurant (statements 14 and 15). They explicitly mentioned that critiquing is a simple but effective way to quickly input new preferences and get a new recommendation with only a few clicks. In general, all testers said that they would definitely use the system on their mobile phones.

All testers completed the test scenario and found a restaurant that satisfied their needs. The testers generally rated the selected restaurants highly, with an average rating of 3.87 out of 5. A specific MobyRek function collected this rating. Just after the testers added

Table 1. Experimental results of the MobyRek system prototype.

Statement	Average	Standard deviation
1. Overall, I'm satisfied with how easy the system is to use.	5.47	0.83
2. This system was simple to use.	5.13	1.30
3. I can effectively complete my work using this system.	5.80	1.08
4. I can quickly complete my work using this system.	5.40	1.18
5. I feel comfortable using this system.	5.13	0.92
6. Learning to use this system was easy.	5.80	1.01
7. Information on using the system (such as online help, onscreen messages, and other documentation) was clear.	6.13	0.74
8. It was easy to find the information I needed.	5.73	0.88
9. The information the system provided was easy to understand.	5.53	1.13
10. The information effectively helped me complete the task and scenario.	5.73	1.16
11. The information on the system screens was organized clearly.	5.40	1.18
12. The system interface is pleasant.	5.20	1.32
13. I liked using the system interface.	5.20	1.01
14. I found it useful to critique a restaurant and get a new sorting of the offers.	6.33	0.72
15. I found it easy enough to critique a restaurant.	5.87	0.83
16. This system has all the functions and capabilities I expected it to have.	5.80	0.68
17. Overall, I'm satisfied with this system.	5.87	0.83

their selected restaurant, MobyRek showed a drop-down list containing six possible ratings:

- "no idea" (n/a),
- "perfect" (5),
- "well suited" (4),
- "suited" (3),
- "acceptable" (2), and
- "weakly accepted" (1).

We used a 1-to-5 scale because it's widely used in other recommender systems and has proved to be fine enough to collect product evaluations. So, testers were satisfied with the products they found, and the system convinced them that they made good choices. This is an important aspect in recommender systems—that is, the capability to convince a user that her choice is good given the recommendations and information presented during the overall human-computer interaction. Regarding the interaction length (the number of recommendation cycles needed to identify a good product), almost all testers found their desired restaurant within two to three cycles. The selected item was always ranked in the top position, and the average rank was 1.87.

The evaluation demonstrates that our critique-based recommendation approach is a viable design solution for mobile recommender systems. Critiques can be an effective method for acquiring and revising user preferences and can therefore involve users in multistage system interaction.

In addition, we analyzed the interaction logs, which showed that many users explicitly formulated their preferences at the beginning of interaction. So, this preferences acquisition modality must be supported in conjunction with the approach of trying to learn user preferences from implicit data. Furthermore, we also discovered that some users like to scan a large part of a result list and not just consider a few items, such as the top recommendations. Such users want to find "all good items" rather than "some good items," where the latter is a classic function of recommender systems. Searching for all good items is a more complex information-search function and, considering mobile devices' physical limitations, poses severe challenges to mobile applications' design and implementation. So, a critical issue becomes choosing the number of products to show at each recommendation cycle. On one hand, this number should be as small as possible to fit on the mobile device's screen so that the user doesn't have to scroll or jump between screens. However, having a small number of recommendations decreases the system's capacity to recall relevant products, which becomes a severe limitation when the user wants to see more or all options.

Moreover, we observed how the users' familiarity with traditional Web-based interfaces influences their expectation and acceptance of a system. Many users in our experiment initiated their search process traditionally, by searching for a product similar to one they knew or by specifying preferences at the beginning of the interaction. So, a mobile recommender system should support traditional approaches while offering more innovative ones, providing a seamless learning path and the possibility of combining them in the same interaction.

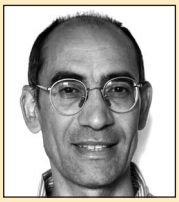
In the future, we plan to further develop our empirical studies to answer open questions. We hope to understand the impact of displaying full recommendation lists on user search behavior, compared to selectively shortened lists. We also hope to fully support manual query relaxation (that is, a user-driven process). Finally, we plan to measure

and analyze the quantity of content information, such as the number of items the users can and actually do scan in such interactions. ■

References

1. G. Adomavicius and A. Tuzhilin, "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions," *IEEE Trans. Knowledge and Data Eng.*, vol. 17, no. 6, 2005, pp. 734–749.
2. R. Burke, "Interactive Critiquing for Catalog Navigation in E-Commerce," *Artificial Intelligence Rev.*, vol. 18, nos. 3–4, 2002, pp. 245–267.
3. L. McGinty and B. Smyth, "Deep Dialogue vs. Casual Conversation in Recommender Systems," *Recommendation and Personalization in eCommerce, Proc. AH 2002 Workshop*, Univ. of Malaga Press, 2002, pp. 80–89.
4. M. Torrens, B. Faltings, and P. Pu, "Smartclients: Constraint Satisfaction as a Paradigm for Scaleable Intelligent Information Systems," *Constraints*, vol. 7, no. 1, 2002, pp. 49–69.
5. Q.N. Nguyen and F. Ricci, "User Preferences Initialization and Integration in Critique-Based Mobile Recommender Systems," *Proc. 5th Int'l Workshop Artificial Intelligence in Mobile Systems* (AIMS 04), 2004, pp. 71–78; http://w5.cs.uni-sb.de/~baus/aims04/cameraready/P11.pdf.
6. A. Felfernig and A. Kiener, "Knowledge-Based Interactive Selling of Financial Services with Fsadvisor," *Proc. 20th Nat'l Conf. Artificial Intelligence and 17th Innovative Applications of Artificial Intelligence Conf.* (AAAI 05 and IAAI 05), AAAI Press/MIT Press, 2005, pp. 1475–1482.
7. J.R. Lewis, "IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use," *Int'l J. Human-Computer Interaction*, vol. 7, no. 1, 1995, pp. 57–78.

The Authors



Francesco Ricci is an associate professor of computer science at the Free University of Bozen-Bolzano. His research interests include recommender systems, intelligent interfaces, constraint satisfaction problems, machine learning, case-based reasoning, and software architectures. He received his laurea in mathematics from the University of Padova. He's a member of the *Information Technology and Tourism Journal* editorial board and the ACM. Contact him at the Faculty of Computer Science, Free Univ. of Bozen-Bolzano, Piazza Domenicani 3, I-39100 Bolzano, Italy; fricci@unibz.it.

Quang Nhat Nguyen is a research assistant in the Faculty of Computer Science at the Free University of Bozen-Bolzano. His research interests include machine learning, data mining, soft computing, decision-making support, recommender systems and personalization, and adaptive mobile systems. He received his PhD in computer science from the University of Trento. Contact him at the Faculty of Computer Science, Free Univ. of Bozen-Bolzano, Piazza Domenicani 3, I-39100 Bolzano, Italy; quang.nhatnguyen@unibz.it.