Knowledge Discovery from User Preferences in Conversational Recommendation

Maria Salamó, James Reilly, Lorraine McGinty, and Barry Smyth

Smart Media Institute, University College Dublin,
Belfield, Dublin 4, Ireland
{maria, james.d.reilly, lorraine.mcginty, barry.smyth}@ucd.ie

Abstract. Knowledge discovery for personalizing the product recommendation task is a major focus of research in the area of conversational recommender systems to increase efficiency and effectiveness. Conversational recommender systems guide users through a product space, alternatively making product suggestions and eliciting user feedback. Critiquing is a common and powerful form of feedback, where a user can express her feature preferences by applying a series of directional critiques over recommendations, instead of providing specific value preferences. For example, a user might ask for a '*less expensive*' vacation in a travel recommender; thus '*less expensive*' is a critique over the *price* feature. The expectation is that on each cycle, the system discovers more about the user's *soft* product preferences from minimal information input. In this paper we describe three different strategies for knowledge discovery from user preferences that improve recommendation efficiency in a conversational system using critiquing. Moreover, we will demonstrate that while the strategies work well separately, their combined effort has the potential to considerably increase recommendation efficiency even further.

1 Introduction

Recommender systems apply knowledge discovery techniques to decide which recommendations are the most suitable for each user during a live customer interaction. In this paper, we focus on conversational case-based recommenders [1] which help users navigate product spaces by combining ideas and technologies from information retrieval, artificial intelligence and user modelling. As part of their cyclic recommendation process, conversational systems aim to retrieve products that respect user preferences by requiring users to provide minimal feedback in each cycle. It is expected that over the course of a recommendation session that the recommender learns more about user preferences and therefore it can assist in the discovery of recommendation knowledge which prioritises products that best satisfy these preferences [2,3]. Advantages of the approach include: (1) users have more control over the navigation process [4]; and (2) users are guided to target products faster than standard browsing and alternative recommendation approaches [5,6].

A. Jorge et al. (Eds.): PKDD 2005, LNAI 3721, pp. 228–239, 2005.

Recommender systems can be distinguished by the type of feedback that they support; examples include *value elicitation*, *ratings-based feedback* and *preference-based feedback* [7]. In this paper we are especially interested in a form of user feedback called *critiquing* [8], where a user indicates a directional feature preference in relation to the current recommendation. For example, in a travel/vacation recommender, a user might indicate that she is interested in a vacation that is *longer* than the currently recommended option; in this instance, *longer* is a critique over the *duration* feature.

Within the recommender systems literature the basic idea of critiquing can be traced back to the seminal work of Burke *et al.* [4]. For example, Entrée is the quintessential recommender system that employs critiquing (also sometimes referred to as *tweaking*) in the restaurant domain. Entrée allows users to critique restaurant features such as *price, style, atmosphere* etc. Importantly, critiquing is a good example of a minimal user feedback approach where the user does not need to provide a lot of specific preference information, while at the same time it helps the recommender to narrow its search focus quite significantly [8]. As recommender systems become more commonplace, there has been renewed interest in critiquing, with the major focus of research in increasing the efficiency of recommendation dialogues [9]. Furthermore, recent research has highlighted the importance of investigating techniques for automating the discovery of implicit preference knowledge while requiring minimal information input from the user [3].

In this paper we describe three strategies for knowledge discovery from user preferences that improves the performance of a critique-based recommender. Specifically, we build upon work previously described by [10], where the idea is to consider a user's critiquing history, as well as the current critique when making new recommendations. This approach leads to significant improvements in recommendation efficiency. We continue in this paper by considering the history of critiques as a *user model* which determines the user preferences in a session. We present a case discovery strategy, a feature discovery strategy and a query discovery strategy which have the potential to focus rapidly on satisfactory product cases. Finally, we show that by combining all three strategies, we can further improve overall recommendation efficiency.

2 Background

This section describes the "incremental critiquing" [10] approach, which offers major benefits in recommendation efficiency over the basic critiquing approach as described by [11]. We consider the incremental critiquing approach as the basis for our knowledge discovery strategies because it also considers a user's critiquing history as a basic starting point.

The incremental critiquing implementation assume a conversational recommender system in the style of Entrée [11]. Each recommendation session starts with an initial user query resulting in the retrieval of a case with the highest *quality*. The user will have the opportunity to accept this case, thereby ending

```
q: query, CB: CaseBase, cq: critique, c_r : current recommendation, U : User model

1.   define Incremental_Critiquing(q, CB)
2.   cq:= null
3.   U:= null
4.   begin
5.     do
6.       c_r ← ItemRecommend(q, CB, cq, U)
7.       cq ← UserReview(c_r, CB)
8.       q  ← QueryRevise(q, c_r)
9.       U  ← UpdateModel(U, cq, c_r)
10.    until UserAccepts(c_r)
11.  end

12.  define ItemRecommend(q, CB, cq, U)
13.    CB' ← {c ∈ CB | Satisfies(c, cq)}
14.    CB' ← sort cases in CB' in decreasing Quality
15.    c_r ← most quality case in CB'
16.  return c_r

17.  define UserReview(c_r , CB)
18.    cq ← user critique for some f ∈ c_r
19.    CB ← CB - c_r
20.  return cq

21.  define QueryRevise(q, c_r)
22.     q ← c_r
23.  return q

24.  define UpdateModel(U, cq, c_r)
25.     U ← U - contradict(U, cq, c_r)
26.     U ← U - refine(U, cq, c_r)
27.     U ← U + (<cq, c_r>)
28.  return U
```

Fig. 1. The incremental critiquing algorithm

the recommendation session, or to critique it as a means to influence the next cycle. A simplified version of the incremental critiquing algorithm is given in Figure 1.

The incremental critiquing algorithm consists of 4 key steps: (1) a new case c_r is *recommended* to the user based on the current query and the previous critiques; (2) the user *reviews* the recommendation and applies a directional feature critique, *cq*; (3) the query, q is *revised* for the next cycle; (4) the user model, U is updated by adding the last critique *cq* and pruning all the critiques that are inconsistent with it. The recommendation process terminates either when the user is presented with a suitable case, or when they give up.

Importantly, the recommendation process is influenced by the user model of previous critiques, U, that is incrementally updated on each cycle. Incremental critiquing modifies the basic critiquing algorithm. Instead of ordering the filtered cases on the basis of their similarity to the recommend case, it also computes a *compatibility* score (see Equation 1) for each candidate case. The compatibility score is essentially the percentage of critiques in the user model that this case satisfies. Then, the compatibility score and the candidate's (c') similarity to the current recommendation (c_r) are combined in order to obtain an overall *quality* score (see Equation 2, by default $\beta = 0.75$). The quality score is used to rank the filtered cases prior to the next recommendation cycle (see line 14 in Figure 1) and the case with the highest quality is then chosen as the new recommendation.

$$Compatibility(c', U) = \frac{\sum_{\forall_i} satisfies(U_i, c')}{|U|} \tag{1}$$

$$Quality(c', c_r, U) = \beta \cdot Compatibility(c', U) + (1 - \beta) \cdot Similarity(c', c_r) \tag{2}$$

Algorithm 1 maintains a critique-based user model which is composed of those critiques that have been chosen by the user so far. One of the key points

focused on the incremental critiquing approach is the maintenance of the user model which prevents the existence of critiques that may be inconsistent with earlier critiques. The user model is maintained using two possible actions: (1) pruning previous critiques that are inconsistent with the current critique; (2) removing all existing critiques, for which the new critique is a refinement.

3 Knowledge Discovery Strategies

This section presents three strategies for knowledge discovery from user preferences that have the potential to improve product recommendations. The first strategy discovers those cases that better satisfy the user preferences. The second strategy deals with feature dimensionality, taking into account user preferences to discover the relative importance of each feature for computing similarity. Finally, this paper presents a query discovery strategy that facilitates larger jumps through the product space based on the current user critique. All of them share a common foundation, they exploit the user's history of critiques to discover recommendation knowledge from the user preferences, in order to personalize and focus in more rapidly on satisfactory product cases.

3.1 Discovering Satisfactory Cases: Highest Compatibility Selection

A key problem with the standard incremental critiquing [10] approach is that there are no guarantees that the recommendations it returns will completely satisfy a user's preferences. This is largely due to the underlying case selection strategy which averages the compatibility (with past critiques) and similarity (with the current preference case). What we propose is a new strategy for product recommendation, *Highest Compatibility Selection* (HCS), that allows the recommender to select the most *suitable* cases; i.e., those cases that are most compatible with user preferences. This maximum compatibility strategy can be easily introduced into the incremental critiquing algorithm.

Figure 2 demonstrates that the only procedure which is affected in the incremental critiquing algorithm is the *ItemRecommend* step. As before, the list of remaining cases is filtered out using the current critique *cq*. In addition two new steps are added. First, the recommender computes the compatibility score, as detailed in Equation 3. It is important to note that the compatibility function has also been modified, as explained below in Equation 3. Instead of averaging the compatibility and the similarity, as is done with incremental critiquing, our second step assembles the cases with the highest compatibility from the list of remaining cases. Importantly, in our approach, only the remaining cases with highest compatibility value, CB'', influence the product recommendation. Put differently, the strategy prioritises those cases that satisfy the largest number of critiques made by the user over time.

The compatibility function. We have considered case discovery to be an optimization problem in which we are trying to recommend cases that maximally

```
q: query, CB: CaseBase, cq: critique, c_r : current recommendation, U: User Model

1.define ItemRecommend(q, CB, cq, U)
2.  CB' ← {c ∈ CB | Satisfies(c, cq)}
3.  CB' ← sort cases in CB' in decreasing compatibility score
4.  CB'' ← selects those cases in CB' with highest compatibility
5.  CB'' ← sort cases in CB'' in decreasing order of their sim to q
6.  c_r ← most similar case in CB''
7.return c_r
```

Fig. 2. Adapting the incremental critiquing algorithm *ItemRecommend* procedure to improve focus on recommendation by using Highest Compatibility Selection strategy

satisfy the user preferences. For this reason, we evaluate the remaining cases as if they were a set of states in a *Reinforcement Learning Problem* (RLP) [12], which consists of maximising the sum of future rewards in a set of states. Reinforcement Learning theory is usually based on *Finite Markov Decision Processes* (FMDP).

Each case is treated as a state whose compatibility score is updated at each cycle using a Monte-Carlo value function (see Equation 3). This function evaluates the *goodness* of each state — for us the possible states are the complete set of remaining cases we want to enhance — according to the critiques the user has selected.

$$Compatibility(c', U_f) = \begin{cases} comp(c') + \alpha \times (1 - comp(c')) & \textit{if c' satisfies } U_f \\ comp(c') + \alpha \times (0 - comp(c')) & \textit{if c' dissatisfies } U_f \end{cases} \quad (3)$$

Our goal is to maximally satisfy all the user preferences. Thus, we are looking for a set of maximally compatible cases (i.e., those cases which have the highest compatibility *(comp)* value considering all the user preferences (U) or past critiques). At the beginning of each session each candidate case, *c'*, has a default compatibility value (i.e., *comp(c')= 0.5*). This value is updated over cycles taking into account the satisfaction or not of the current critique. The α parameter in Equation 3 is the learning rate which is usually set up to 0.1 or 0.2 values; a larger value leads to a larger gap between cases in early stages. In our case, the learning rate is not important since we are looking for levels of satisfaction. In other words, we are not trying to obtain a set of states that arrive as quickly as possible to a 1.0 value, as usually is done in RLP.

It is important to note that Equation 3 updates the compatibility value stored by each case according to the last user critique (U_f) as opposed to computing all the set of critiques like the incremental approach (see Equation 1). The $Compatibility(c', U_f)$ value computed in the current cycle will be the ($comp(c')$) in the next cycle.

3.2 Discovering Important Features: Local User Preference Weighting

The previous strategy highlights the case dimensionality problem. In other words, it is focused on discovering cases that maximally satisfy user preferences.

Now, we present a strategy that concentrates on the feature dimensionality. We propose a *local user preference weighting* (LW) strategy that discovers the relative importance of each feature in each case as a weighting value for computing the similarity, taking into account user preferences.

Our LW strategy for the discovery of feature knowledge is basically motivated by the previous knowledge discovery strategy. As we have explained in Section 3.1, the discovery of case knowledge is based on maximising user preferences, which means we are looking for the most compatible cases. These cases are quite similar on their critiqued features and their differences mainly belong to those features that have not yet been critiqued. So, the aim of LW strategy is to prioritise the similarity of those features that have not yet been critiqued.

$$\begin{array}{l} \textit{for each feature } f \textit{ in case } c' \textit{ compute:} \\ weight(c'_f) = 1 - \left(\frac{\#critiques\ in\ U\ that\ satisfy\ feature_f\ in\ case\ c'}{\#total\ critiques\ feature_f\ in\ U} \times 0.5 \right) \end{array} \tag{4}$$

We generate a feature weight vector for each case, as shown in Equation 4. A feature that has not been critiqued will assume a weight value of 1.0 and a decrement will be applied when a critique is satisfied by the case. As such, the feature weight will be proportional to the number of times a critique on this feature is satisfied by the case. However, as it can be seen in Equation 4 the weights never decrease to a 0 value. For example, in a travel vacation recommender with a user model that contains two critiques [price, $>$, 1000] and [price, $>$, 1500], a case with two features {*duration*, *price*} whose price is 2000 will have as price weight a 0.5 value because it satisfies both critiques whereas the duration weight will be 1.0 because there is no critique on this feature. It is important to recap that the key idea here is to prioritise the similarity of those features that have not yet been critiqued in a given session.

Our proposal is to discover the best product to recommend by exploiting the similarity of those features that best differentiate the highest compatible cases. To achieve this, a candidate's (c') similarity to the recommended case (c_r) is computed at each cycle in the *incremental* recommender system as shown by Equation 5.

$$Similarity(c', c_r) = \sum_{\forall_f} weight(c'_f) \times similarity(c'_f, c_{r_f}) \tag{5}$$

The similarity between the candidate case (c') and the recommended case (c_r) for each feature f is combined with the weight for this feature. The weight is computed previously using Equation 4.

3.3 Discovering Query Knowledge: Binary Search

The incremental critiquing approach is susceptible to feature-critique repetitions that offer only a minor change in the relevant feature value from each cycle to the next. We propose that this is largely due to the linear search policy it uses to navigate through the value-space for the critiqued feature. The result

is that the recommendation system takes very *short steps* through the space of possible alternatives. In this section we describe how the incremental critiquing algorithm can be easily altered to facilitate *larger jumps* through the value space for knowledge discovery of a given feature by taking a more efficient *binary search* (BS) approach.

```
q: query, CB: CaseBase, cq: critique, c_r : current recommendation
1.   define QueryRevise(q, cq, CB, c_r)
2.   begin
3.       q   ← c_r
4.       CB' ← {c ∈ CB | Satisfies(c, cq)}
5.       CB' ← eliminate cases that conflict with prior critiques
6.       f_cq  ← set value in q for critiqued feature f ∈ c_r by Eq. 6
7.     return q
8.   end
```

Fig. 3. Illustrating the binary search procedure for query discovery

Figure 3 demonstrates how the incremental algorithm can be easily extended to support our proposed approach. The only procedure which is affected is the *QueryRevise* step of the incremental critiquing algorithm. The new query is updated with all of the features from the current recommendation, c_r. In addition two new steps are added. First, the recommender system gathers all of the available cases that satisfy current feature critique (see line 4 of Figure 3). The second step involves determining the value-change the critiqued feature will take on in the revised query, q, used for retrieval of the next recommendation. Importantly, in our approach all the remaining cases, CB', influence the final value. There are many approaches that could be used to compute this value. In this paper we examine the possibility of computing the median (see equation 6) for all cases in CB'.

The recommender system collects all of the alternative value possibilities for the critiqued feature from the cases covered by CB'. For instance, if the critiqued feature were [price, $<$, 2000] the recommender would gather all value options that were less than 2000 from the set of remaining cases (e.g., 1800, 1650, 1600, 1570, 1460, 1350, etc.). Equation 6 assigns a value for the critiqued feature $f_{cq} \in q$ by calculating the average feature value over all the relevant cases.

$$f_{cq} = \begin{cases} CB'_{n+1/2}(f \text{ of } cq) & \text{if odd } \#cases \\ \frac{CB'_{n+1/2}(f \text{ of } cq) + CB'_{(n+1/2)+1}(f \text{ of } cq)}{2} & \text{if even } \#cases \end{cases} \quad (6)$$

For Equation 6 it is assumed that the remaining case options, CB' are first sorted in ascending order. Here $CB'_i(f \text{ in } cq)$ is the feature value critiqued by cq in the i^{th} case. The median value corresponds to a cumulative percentage of 50% (i.e., 50% of the values are below the median and 50% of the values are

above the median). We place the critiqued features in ascending value order and find the middle value if the number of cases is odd or find the middle pair and compute the mean value between them if we have an even number of cases.

One important point, that also needs to be considered, is previous critiques on the same feature. For example, suppose that a user has asked in a previous cycle for a *less expensive* vacation than a 2500 recommendation and, in the current cycle, the user says that she prefers a *more expensive* than a 1000 vacation. In such situation in the current cycle, all the cases including those that exceed a 2500 vacation will satisfy the current critique *more expensive* than 1000. If we compute the median value to jump larger in the search space, we also include those cases rejected previously by the user. To avoid these situations, we use the history of critiques applied by the user in order to cut correctly off the search space. The previous critiques stored in the user model are treated as a set of *soft constraints* [13] that allow us to control the number of remaining cases that will be used to compute the median value.

So, following the earlier example, we only consider computing the median of those cases that are *more expensive* than 1000 and *less expensive* than 2500. As detailed in line 5 of Figure 3, before computing the median, we check for the existence of previously applied critiques that contest the inclusion of cases in CB', and eliminate these cases from further consideration. Put differently, we use prior critiques to decide what cases should be covered by CB', and to ultimately set the value selection bounds for f_{cq}.

The key motivation behind our *binary search* extension to incremental critiquing was to reduce critique repetition sequences, and improve recommendation efficiency by discovering satisfactory products for users more rapidly. In short, this binary search style approach enables the recommender to focus its search on those candidate cases that: (1) satisfy the current critique; (2) fulfill previously applied critiques; and (3) are similar to the current case but further away from it, and thus have the capability of navigating the search space of options quickly.

4 Evaluation

In this paper so far we have argued that the incremental form of critiquing is limited by its tendency to recommend cases that do not maximally satisfy the user preferences. We propose three strategies that aid knowledge discovery in a quest to improve retrieval accuracy and recommendation efficiency. This section describes the related evaluation methodology that we used and the results that ensued.

4.1 Setup

The evaluation was performed using the standard Travel dataset (available from *http://ww.ai-cbr.org*) which consists of 1024 vacation cases. Each case is described in terms of 9 features including *price*, *duration*, etc. The dataset was chosen because it contains numerical and nominal features and it also provides a wide search space.

We evaluate the highest compatibility selection (HCS), the local user preference weighting (LW), the binary search (BS) and also all strategies combined in our recommender (ALL) over incremental critiquing (incremental).

4.2 Methodology

We would like to have carried out an online evaluation with live-users, but unfortunately this was not possible. As an alternative we opted for an offline evaluation similar to the one described by [14]. Accordingly, each case (which are called the 'base') in the case-base is temporarily removed and used in two ways. First, it serves as a basis for a set of queries by taking random subsets of its features. We focus on subsets of 1, 3 and 5 features to allow us to distinguish between *hard*, *moderate* and *easy* queries respectively. Second, we select the case that is most similar to the original base. These cases are the recommendation targets for the experiments. Thus, the base represents the ideal query for a user, the generated query is the initial query provided by the 'user', and the target is the best available case for the user. Each generated query is a test problem for the recommender, and in each recommendation cycle the 'user' picks a critique that is compatible with the known target case; that is, a critique that when applied to the remaining cases, results in the target case being left in the filtered set of cases. Each leave-one-out pass through the case-base is repeated 10 times and the recommendation sessions terminate when the target case is returned.

Related real user studies [15] have highlighted discrepancies between the original [14] artificial user model construction and real user behaviour with respect to critiqued application and repetition. We use a modified artificial user model that is informed by our real-user studies. The new model is designed to respond to recommendations in a manner that is more consistent with the responses recorded from real-users. In particular, our artificial user model repeats critique selections during recommendation sessions until its target feature values are met. For example, suppose our artificial user is looking for a 3-week vacation and they are presented a 3-day city-break. They are likely to ask for a *longer* vacation by critiquing the *duration* feature. In this evaluation, the artificial user will continue to critique a feature until it's preferences constraint is satisfied.

4.3 Recommendation Efficiency

We analyse the recommendation efficiency — by which we mean average recommendation session length — when comparing the new strategies to incremental critiquing. Figure 4(A) presents a graph comparing the average session length of the incremental critiquing approach to the combination of all the strategies (ALL) for 3 different initial query lengths. The three strategies combined consistently reduce average session length when compared to the incremental critiquing approach, demonstrating the potential to improve recommendation efficiency. For example, for the hard queries the incremental recommender results in an average on session of 12.46 cycles while the combined recommender results in an average of 11.47 cycles.

Figure 4(B) shows the benefit of each strategy (HCS, LW, and BS) separately and the combined strategies (ALL) in our recommender when compared to the incremental critiquing. We find that all strategies separately result in a relative session length reduction of between 2.65% and under 7.5%, with some variation in the relative benefit due to the HCS, LW and BS approaches. The lowest benefit is for the highest compatibility selection (HCS) approach, which ranges between 2.65% and 3.81%, because it does the same process as the incremental critiquing approach with two little modifications that consists of using a different compatibility measure and a different strategy for discovering the set of cases available for recommendation. Similarly a 3% to 4% benefit is found using the binary search (BS) strategy. On the other hand, the local weighting approach (LW) gives the highest benefit, ranging from 4.5% to 6.73%, when applied alone. These results show that the strategy to promote uncritiqued features is able to discover and detect differences between cases that are maximally compatible to the user critiques.

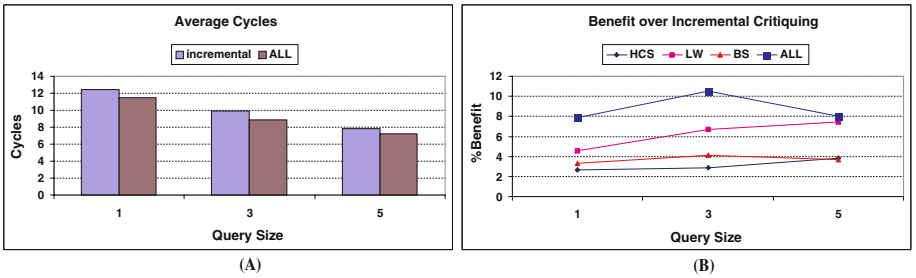

Fig. 4. Average session length and benefit over incremental critiquing

On the other hand, the combined strategies in our recommender result in a reduction in session length that ranges from nearly 8% to 10.5%. Combining all of the strategies further enhances recommendation performance, resulting in the discovery of better recommendations for all queries (hard, moderate and easy). It seems that the recommenders ability to learn user preferences is greater when combining information from these three distinct knowledge discovery resources. An important point to note is that all results show a lower benefit for easy queries. This is to be expected perhaps since the easy queries naturally result in shorter sessions and thus there are fewer opportunities to find good lower and upper critique bounds to focus the search space properly in the BS strategy, and hence fewer opportunities for the benefit to be felt.

It is worth noting the benefit of the proposed strategies over the basic critiquing algorithm, see Figure 5. We have selected incremental critiquing as a benchmark because it improves on the recommendation efficiency of the basic critiquing algorithm by over 82%. Nevertheless, our combination of approaches has the potential to deliver further reductions in session length (from 83.5% to

upper 84%) even with short sessions where the BS approach does not have much of an opportunity to affect the recommendations.

To summarise, a significant efficiency benefit is enjoyed by HCS, LW and BS strategies, when compared to the incremental critiquing approach. The main contribution of this paper is that the proposed strategies assist in the discovery of useful recommendation knowledge, allowing the system to prioritise products that best satisfy the user. We have demonstrated that this approach is highly effective, even in situations where only a minimal knowledge of user preferences is available (e.g., critiquing approach). Furthermore, the results of the combined strategies show a significant increase in recommendation efficiency when compared to incremental critiquing and also to the basic critiquing approach proposed by [11].

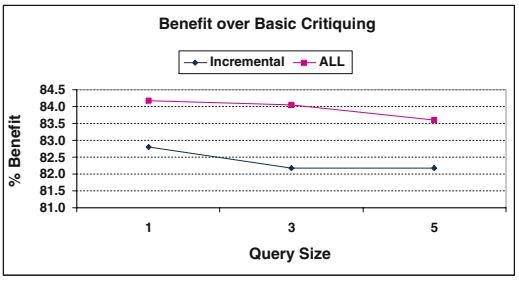

Fig. 5. Incremental and ALL benefit over basic critiquing approach

5 Conclusions

The discovery of implicit user preference knowledge is necessary to decide which product recommendations are the most suitable for each user during a live customer interaction. In this paper we have proposed three discovery strategies that aim to improve recommendation efficiency. First of all, we have presented a case prioritization strategy that maximises the user preferences over time. Secondly, we have presented a user preference weighting strategy that prioritises features locally to each case. Finally, we have presented a query strategy that has the capability of navigating the search space quickly.

Our experiments indicate that the three proposals have the potential to deliver worthwhile efficiency benefits. Reductions in the average length of recommendation sessions were noted in all of the proposals, both separately and combined, when compared to the incremental and basic critiquing setups. Importantly, the proposed strategies are sufficiently general to be applicable across a wide range of recommendation scenarios. In particular, those that assume a complex product-space where recommendation sessions are likely to be protracted, and/or domains where only minimal user feedback is likely to be available.

References

1. D.W. Aha, L.A. Breslow, and H. Muñoz-Avila. Conversational Case-Based Reasoning. *Applied Intelligence*, 14:9–32, 2000.
2. D. McSherry. Increasing Dialogue Efficiency in Case-Based Reasoning without Loss of Solution Quality. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 121–126. Morgan-Kaufmann, 2003.
3. D. McSherry and C. Stretch. Automating the Discovery of Recommendation Knowledge. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, page forthcoming. Morgan-Kaufmann, 2005.
4. R. Burke, K. Hammond, and B.C. Young. The FindMe Approach to Assisted Browsing. *Journal of IEEE Expert*, 12(4):32–40, 1997.
5. L. McGinty and B. Smyth. Comparison-Based Recommendation. In Susan Craw, editor, *Proceedings of the 6th European Conference on Case-Based Reasoning*, pages 575–589. Springer, 2002. Aberdeen, Scotland.
6. H. Shimazu. ExpertClerk: A Conversational Case-Based Reasoning Tool for Developing Salesclerk Agents in E-Commerce Webshops. *Artificial Intelligence Review*, 18(3-4):223–244, 2002.
7. B. Smyth and L. McGinty. An Analysis of Feedback Strategies in Conversational Recommender Systems. In P. Cunningham, editor, *Proceedings of the 14th National Conference on Artificial Intelligence and Cognitive Science*, 2003. Dublin, Ireland.
8. L. McGinty and B. Smyth. Tweaking Critiquing. In *Proceedings of the Workshop on Personalization and Web Techniques at the International Joint Conference on Artificial Intelligence*. Morgan-Kaufmann, 2003.
9. R. Burke. Interactive Critiquing for Catalog Navigation in E-Commerce. *Artificial Intelligence Review*, 18(3-4):245–267, 2002.
10. J. Reilly, K. McCarthy, L. McGinty, and B. Smyth. Incremental Critiquing. In M. Bramer, F. Coenen, and T. Allen, editors, *Research and Development in Intelligent Systems XXI. Proceedings of AI-2004*, pages 101–114. Springer, 2004. Cambridge, UK.
11. R. Burke, K. Hammond, and B. Young. Knowledge-Based Navigation of Complex Information Spaces. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 462–468. AAAI Press/MIT Press, 1996. Portland, OR.
12. M.E. Harmon and S.S. Harmon. Reinforcement learning: A tutorial, 1996.
13. M. Stolze. Soft Navigation in Electronic Product Catalogs. *International Journal on Digital Libraries*, 3(1):60–66, 2000.
14. B. Smyth and L. McGinty. The Power of Suggestion. In *Proceedings of the International Joint Conference on Artificial Intelligence*. Morgan-Kaufmann, 2003.
15. K. McCarthy, L. McGinty, B. Smyth, and J. Reilly. On the Evaluation of Dynamic Critiquing: A Large-Scale User Study. In *Proceedings Twentieth National Conference on Artificial Intelligence*, pages 535–540. AAAI Press / The MIT Press, 2005.