Feature Selection Methods for Conversational Recommender Systems

Nader Mirzadeh
ITC-irst
via Solteri 38
Trento, Italy
mirzadeh@itc.it

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

Mukesh Bansal
TIGEM
via P. Castellino 111
Napoli, Italy
bansal@tigem.it

Abstract

This paper focuses on question selection methods for conversational recommender systems. We consider a scenario, where given an initial user query, the recommender system may ask the user to provide additional features describing the searched products. The objective is to generate questions/features that a user would likely reply, and if replied, would effectively reduce the result size of the initial query. Classical entropy-based feature selection methods are effective in term of result size reduction, but they select questions uncorrelated with user needs and therefore unlikely to be replied. We propose two feature-selection methods that combine feature entropy with an appropriate measure of feature relevance. We evaluated these methods in a set of simulated interactions where a probabilistic model of user behavior is exploited. The results show that these methods outperform entropy-based feature selection.

1. Introduction

Business to consumer web sites have proliferated, and nowadays almost every kind of product or service can be bought on-line. In these web sites, recommender systems have been introduced to help users to select products when large catalogues are available and the user has not enough knowledge to pick up autonomously the best option [9]. In a previous paper we have introduced Trip@dvice [7], a travel planning recommendation methodology that integrates *interactive query management* (IQM) and case-based reasoning (CBR).

In this paper we focus on the attribute selection method used in the IQM subsystem and its evaluation. We present two new feature-selection methods: one is based on the estimation of the probability that a feature/question would really be used/replied by the user, and the second one that combines that estimation with the feature entropy. We have evaluated these methods in a similar way as in [4], i.e., under a pragmatic simulation of user-system interactions. Our simulation takes advantage from the observation of real user-system interactions in a recommender system for travel planning, from which a frequency model of feature usage is derived. This is exploited to model the user acceptance to provide a value to a proposed feature/question. The result of this evaluation shows that feature entropy is effective in result size reduction, but it identifies features that would unlikely be used by a real user. In addition, we show the new proposed methods outperform the classical entropy-based feature selection in a more realistic simulation of user-system interaction.

The rest of the paper is organized as follows. The next Section describes briefly the recommendation methodology, the role of feature selection in product recommendation, and summarizes the results of an earlier evaluation, which involved real users, from which have we derived feature frequency usage data. Section 3 introduces the new proposed utility- and a popularity-based feature-selection methods. Sections 4 and 5 compare four feature selection methods with two different evaluation strategies, i.e. without and with a probabilistic user acceptance model for features/questions. Finally, Section 6 summarizes the conclusions.

2. The Recommendation Process

Case-based reasoning (CBR) is a problem solving methodology that tries to solve a problem at hand by using the solutions of the past similar problems [1]. We have applied CBR to rank (recommend) those products that meet the explicit user's preferences contained in a user query, by deriving implicit preferences contained in previously stored user-system interactions (cases).

Figure 1 sketches the architecture of the proposed recommendation methodology. When the user is searching for desired item(s) described in an electronic catalogue, he is supposed to specify a query in terms of constraints over the features describing the product. The Graphical User Inter-

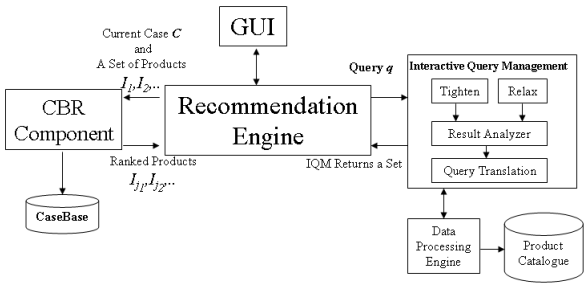

Figure 1. Recommendation architecture

face (GUI) collects these explicit user's preferences, and converts them into a query. The RecommendationEngine dispatches that query to the *interactive query management* (IQM) module that computes the result set. If it is empty, then IQM exploits the Relax sub-module to determine a set of relaxed sub-queries that would return some result by relaxing some constraints [5]. On the other hand, if the result set contains too many items (i.e., above a predefined and application depended threshold), then the *Tighten* sub-module is called to suggest some features that the user can use to further specify (tighten) the initial query. In this paper we focus on the Tighten module and its feature-selection method.

If the result set contains a reasonable (small) set of items, then these are passed to the Case Base Reasoning component, where each item is scored. An item gets a higher score if it is more similar (or equal) to those selected by other users in similar recommendation sessions, where the similarity of two sessions is computed relying on the whole session description, and by applying a collaborative-by-content approach [6].

This methodology has been implemented in a recommender system called *NutKing* (http://itr.itc.it), which recommends travel products/items that the regional tourism organization of Trentino (Italy) promotes [7]. When a query to a product catalogue (e.g. accommodations) returns too many items, NutKing suggests three features and asks the user to provide a desired value (or values range) for one or more of those features (see Figure 2)

We conducted an empirical evaluation of NutKing, which involved more than 40 users [7]. In this experiment feature selection based on entropy maximization was applied in the Tighten module. The following results were obtained:

- On the average 4.4 features were constrained in the initial user query;
- 13.4 queries were issued by a user in a recommendation session;
- In 2.1 among these 13.4 queries (15.7%) the system suggested to tighten the query;

Figure 2. A screen shot from NutKing

- In 0.6 among these 2.1 queries (28.6%) the user accepted to tighten the query using one of the three suggested features.

This shows that only a small number of users accepted tightening suggestions (28.6%). We hypothesized that the cause was a mediocre feature selection method. For instance, the system may suggest to specify "pets welcome", because (roughly) half of the accommodations in the result set allows guests to bring their pets and half not, hence producing a large entropy score for that feature. But a few guests do have pets, and hence are interested in replying to that question.

3. Feature-Selection Methods

In this section, we shall introduce two feature selection methods that cope with the interaction problems mentioned earlier. These methods compute a score for each feature and then select the feature that gets the highest score. The score computed by the first method measures the frequency of usage of the feature, and it is called "popularity". The second methods scores features with a utility function.

In the following discussion we assume that products/items are described with a feature vector $(x_1, \ldots, x_n)$, where x_i is the value of the i-th feature f_i. Feature types are Boolean, symbolic or numeric. The *Popularity* of feature f_i is given by

$$\hat{p}_i = \frac{\text{\# of queries constraining } f_i}{\text{total number of queries}} \quad (1)$$

where we assume to have a sample of queries defined by a community of user.

Utility theory is a tool for modelling rational decision making [8]. In this setting, the utility function maps world states $S_1, \ldots, S_M$ to real numbers, $U(S_j)$, the utility of state S_j. A state is the outcome of an action $(A_1, \ldots, A_N)$ available to a rational agent. The expected utility of an action A_i given the evidence E is computed as

$$EU(A_i|E) = \sum_{j=1}^{M} U(S_j)p(S_j|A_i, E) \quad (2)$$

where $p(S_j|A_i, E)$ is the probability of reaching state S_j after action A_i with the background evidence E.

In our approach action A_i is performed by the system and models the suggestion of feature f_i to the user to further constraint the current query. If the item space has n features, then we have $S_1, \ldots, S_{2n}$ outcome states:

$$\begin{array}{lcl} S_{2i-1} & = & \text{user accepts } A_i \\ S_{2i} & = & \text{user rejects } A_i \end{array}$$

When a system suggestion is accepted, the user specifies a preferred value for that feature, and the current query is tightened. In this context, given action A_i only states S_{2i-1} and S_{2i} can be reached, hence $p(S_j|A_i, E) = 0$ if $j \neq 2i-1$ or $j \neq 2i$. Moreover $p(S_{2i-1}|A_i, E) = 1 - p(S_{2i}|A_i, E)$. Here E is the evidence about the already constrained features at a given interaction.

We define the utility of state S_{2i-1} as the entropy of feature f_i in the current result set. The rationale is that a feature with larger entropy is more likely to reduced the result set if it is constrained. Whereas the utility of state S_{2i} is 0, since the system suggestion is discarded by the user.

$$\begin{array}{lcl} U(S_{2i-1}) & = & H^i_{R_q} \\ U(S_{2i}) & = & 0 \end{array} \quad (3)$$

where

$$H^i_{R_q} = -\sum_{v \in X_i} p^i_v \log(p^i_v) \quad (4)$$

is the entropy of feature f_i computed on the result set R_q of query q. In this formula X_i is the domain of f_i, and p^i_v is the estimated probability of observing the value v in R_q. This is given by:

$$p^i_v = \frac{\text{\# of items in } R_q \text{ s.t. } f_i = v}{|R_q|}$$

In order to compute the expected utility of action A_i, we must compute the probability $p(S_{2i-1}|A_i, E)$. To estimate it, we mined the log file of the users' queries submitted to NutKing during the empirical evaluation mentioned in the previous Section. Unfortunately, as we noted, the system suggestions had been used few times (low acceptance rate) to build a reliable estimation of such probability. Hence, we conjecture that $p(S_{2i-1}|A_i, E)$ is proportional to the a-priori probability that f_i is used in a user query, i.e., is proportional to the Popularity of feature f_i, defined in Eq. 1:

$$p(S_{2i-1}|A_i, E) = \gamma \hat{p}_i \quad (5)$$

where γ is a positive constant. The rational of this estimate is that if a feature is frequently used/constrained in the users' queries, then it is more likely to be accepted by the user when it is suggested by the system to further constrain the current query.

From the above discussion, using Eq. 2, 3 and 5, the expected utility of suggesting feature f_i becomes

$$EU(A_i|E) = H^i_{R_q} \gamma \hat{p}_i \quad (6)$$

We observe that when sorting a set of features according to their expected utilities, the γ constant does not have any influence since $EU(A_i) \geq EU(A_j)$ iff $H^i_{R_q}\hat{p}_i \geq H^j_{R_q}\hat{p}_j$.

4. Feature Selection Evaluation

In this section we compare the performance of four feature selection methods. Each method first sorts the remaining features, i.e., those not included in the current query, according to a score function and then selects the required number of features. The score functions for a feature, say f_i are: Entropy of f_i in the current result set (Eq. 4); Popularity of f_i (Eq. 1); Utility of f_i (Eq. 6); and Random, as base line comparison, which assigns to f_i a random score.

The above methods are compared by using a generic evaluation procedure that simulates user-system sessions (Figure 3). Each session is composed by some interactions. In each interaction the system suggests some features and the simulated user accepts some of them to further constrain the initial query. The values provided by the user for the selected features are those of a test item, thus a simulation basically models a user that is trying to select the test item.

The evaluation procedure takes as input: a set of randomly selected items (test set), the feature selection method (i.e. one among Entropy, Popularity, Random, and Utility), the number of constraints in the initial query, the number of feature(s) to be suggested by the feature selection method at each interaction, the acceptance probability of each feature. For each simulated interaction we record: the number of items in the result set and the popularity of the feature that has been constrained.

At line 2, for each test item, an initial query is generated, which randomly constrains one feature among the 4 most popular features, to the value found in x. Then, if $n_c > 1$, the other constraints are generated by choosing randomly $n_c - 1$ features among those not constrained yet, and assigning the corresponding values found in x. After this initial step, the loop starting at line 5 simulates the user-system

interactions. If the result size of the query is above the given threshold (condition at line 6), the feature-selection method λ is called (new interaction) to sort all the remaining features (i.e., those not constrained yet) and retrieve the n_f best features.

The *for* loop at lines 8-18 simulates the user's interaction with the system. As we discussed in Section 2, three features are suggested to further tighten the current query and hence the simulation tries to constrain the first feature among the top-3 features in the sorted list. If one of the suggested features is "accepted" (condition at line 14), then it is constrained and added to the current query at line 15. More precisely, a feature is *accepted* (i.e., by the simulated user) with probability p_{i_j} provided its value in test item x is neither false (for boolean features) nor null (not known), hence a meaningful constraint can be generated.

If none of these three features is accepted and the sorted list contains *more* features, then next three best features are considered, and the interaction counter (condition at line 10) is increased. Actually, this will not happen in our real recommender system and the interaction will terminate. We opted to model this failure condition as a new interaction to produce results that can be compared with similar experiments conducted by other researchers [3, 4, 10]. In the next Section we shall configure the evaluation procedure in such a way that the interaction terminates if none of the three questions are accepted.

Feature-selection method for recommender systems have usually been evaluated by measuring the interaction length produced in simulated interactions [2, 4, 10]. The typical simulation proceeds, as we described above, by first selecting a target product (e.g. a hotel), and then searching for those products in the catalogue that are most similar to the target with respect to a subset of all the product features. If the target product belongs to the first k retrieved items (stop condition), then the simulated interactions terminates. Otherwise a new feature is selected, its value is assigned as that in the target product, and a new retrieval is performed, matching also this additional feature-value. The stop condition may vary, for instance a simulation can stop when at least one among the 10 most similar products to the target is found in the first k items in the retrieved set. All of these procedures simulate scenarios where a user expresses preferences, one feature after another, to narrow down the result set until it contains an item considered as a good solution to the recommendation problem. The final number of interactions is computed and better feature selection methods are considered those that produce shorter sessions.

As we have already mentioned, unlike the above approaches that use similarity-based retrieval, we rather use logical based filtering, i.e., the items in the result set satisfy the query constraints, and our stop condition is that the result set size is less than 50.

S: A set of items.
λ: feature-selection method to be evaluated
n_c: number of constraints in initial query
$\mathbf{p} = (p_1, \ldots, p_n)$: user acceptance probability of features
n_f: number of features suggested by method λ

```
FeatureSelectionEval(S, n_c, p, λ, n_f)
1  for each x ∈ S do
2     q ← InitialQuery(x, n_c); % make the initial query
3     stop ← false;
4     interaction ← 0;
5     while (not stop) do
6        if Count(q)> 50 then
7           sf ← λ(q, n_f);
8           for j := 1 to |sf| do
9              if (j modulo 3 = 0)
10                interaction ← interaction + 1;
11             end:if
12             f_ij ← get the j^th feature from the sorted list sf;
13             p_ij ← get the acceptance probability of f_ij from p
14             if accept(f_ij, p_ij, x_ij) then
15                q ← q ∪ {constrain f_i according to its value in x};
16                j ← |sf|; % to break the loop
17             end:if
18          end:for
19          if q has not been updated then
20             stop ← true; % report failure
21       end:if
22       else % if the result size threshold is reached
23          stop ← true;
24    end:while
25 end:for
```

Figure 3. The evaluation procedure for testing a feature selection method

In the experiments, we randomly selected 500 accommodations from a lodging catalogue. This catalogue contains 3400 real accommodations promoted by the local tourist organization in Trentino. Accommodations are described by 15 features, 2 numeric and 13 Boolean.

We have run the evaluation procedure (Figure 3) on 500 items (the set S) randomly selected from the accommodation catalogue, initially constraining only one feature ($n_c = 1$), ($n_f = 15$), and simulating a user that always accepts the feature suggested, i.e., $\mathbf{p} = (1, \ldots, 1)$, which follows a more traditional approach assuming a posed question is always replied by the user [3, 10].

Figure 4 shows the average result size of the queries corresponding to the 500 items selected, by different methods at each interaction, and the average popularity of the feature used at each interaction. This results confirm that the entropy method outperforms the others in terms of reduction of the result size, but does not select popular features, especially in the first interactions (until interaction 6).

In this simulation, after the third interaction, the Popularity method performs even worse than the Random one. This happens because the features selected from the fourth to the sixth interactions are correlated with those selected

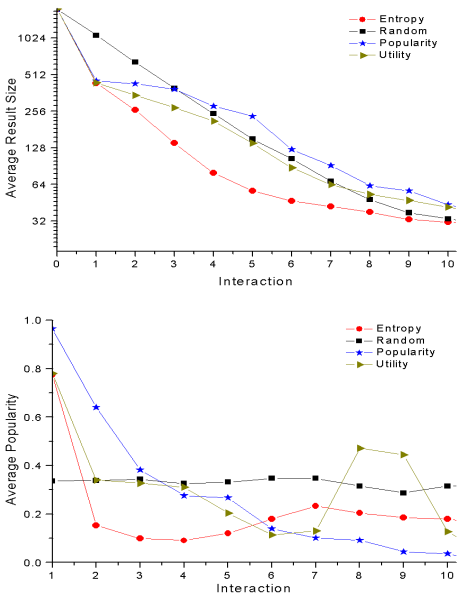

Figure 4. Average result size (a), and average popularity of used features at each interaction (b)

in the initial interactions, and the added constraints are satisfied by most of the items already selected. For instance, the $parking$ feature is the fourth in popularity, and 91% of accommodation items have the parking facility. The Utility method reduces the result size similarly to the Random method from the fourth interaction onwards. We finally observe that the rise in average popularity at steps 8 and 9 for the Utility method is due to the fact that there are features with low utility because the entropy factor ($U(S_{2i-1}) = H^i_{R_q}$ in Equation 2) is low but the popularity of the feature is not.

The conclusion of this evaluation is that the classical Entropy method is still the best when the user is supposed to always accepts tightening with the recommended feature, i.e., the probabilities p_i in the FeatureSelectionEval procedure are set to 1. In the following Section we shall relax this assumption an hypothesis that the user will accept a selected feature with probability smaller than 1.

5. Probabilistic Acceptance Model

In this section we evaluate again the feature selection methods but with a different simulation of user/system interaction. In fact, in the NutKing empirical evaluation (see Section 2) we observed that users frequently stop using the tightening functionality even when the result set is still large (more than 50 items currently selected), not replying to the system features/questions. Hence in this Section we analyze simulations that terminate either when the result size of a query reaches 50 (as before), or if none of the three suggested features is "accepted."

As discussed in Section 3, the probability that a user will accept a feature suggestion can be estimated as $p_i = \gamma \hat{p}_i$ for all $i = 1, \ldots, n$. Now a precise value for γ must be found in order to produce p_i values that are close to the real observations, i.e., close to the real acceptance rate we measured in the empirical evaluation conducted with human subjects (Section 2).

It is worth to mention that $\hat{p}_i$, i.e., the popularity of feature f_i, is the a-priori probability that a user will constrain the feature f_i in the initial query, and it is likely to be an over-estimate of the acceptance probability of using f_i in a successive interaction.

In the experiments with real users, the Entropy method was used in the Tighten module, and we observed that only 28,6% of the times the suggestions were used/accepted. This rather low acceptance rate can have many explanations: the graphical user interface may be too complicated and the user could not understand the suggestions, or the suggestions do not fall into the user's interests and are ignored. This is irrelevant to our evaluation, provided that the simulated acceptance rate is close to that observed in real interactions.

In order to estimate γ and obtain acceptance rates ($p_i = \gamma \hat{p}_i$) close to the observed ones, we run simulations with different values for γ, and we found that with $\gamma = 0.5$ the overall acceptance rate of the Entropy feature selection method comes close to 28.6%. Then, we have used this acceptance model, i.e., $p_i = 0.5\hat{p}_i$, for all the methods, implicitly assuming that a feature is accepted or not with a probability that depends only on the feature itself and not on the other features suggested by a particular method (in previous interactions or in the same interaction).

We run the evaluation procedure again with $n_c = 1$ (one constraint in any initial query), and $n_f = 3$ meaning only three features are suggested at each interaction. This means that if none of these is accepted, then the interaction is stopped. Finally, as we said above, the acceptance model is now not trivial as before ($p_i = 1$), but $\mathbf{p} = (0.5\hat{p}_1, \ldots, 0.5\hat{p}_n\}$.

Figure 5 shows again the average result size of queries at each interaction. It is important to note that now a simulated session can terminate at an arbitrary interaction.

In this new evaluation, the Random method, as expected, performs worse than all the other methods at every interaction. The Entropy method does not improve (reduce) the result size much after the second interaction. This happens because the suggested features, even if in principle are effective in reducing the result set, they have lower probabilities to be accepted, than those suggested by the Popularity and Utility methods. Hence the recommendation session tends to terminate without using any suggested feature, and

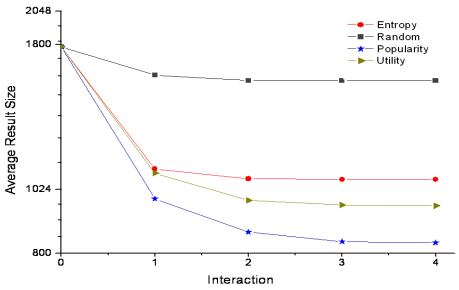

Figure 5. Average result size at each interaction with a probabilistic acceptance model

the result set is not reduced any further.

On the other hand, the Utility, and more clearly the Popularity method, which performed poorly in the previous simulations, now outperform the Entropy method. This is because they select features that may not be so effective in the reduction of the result set, but have higher probability of being used, hence the recommendation session proceeds with a further interaction. A final observation is related to the computational cost of these methods. Entropy here is computed on the result set of a query which is expensive, whereas the Popularity method is a constant time inexpensive method.

We finally observe that these results still hold in other simulations, where we increased the number of constraints in the initial query, i.e, when $n_c > 1$. We do not show this results for lack of space.

6. Conclusions

In conclusion, our work originated from the observation that entropy-based feature selection methods suggest features not necessarily interesting for the user. Hence, we have here introduced two new feature selection methods. The first assigns a utility value to a feature according to its information content and the probability of its usage by the user (popularity), and the second one that uses only the feature popularity. We have compared these methods with an entropy-based approach in two types of simulated interactions. The first follows a more traditional approach, and assumes the user always replies to a question/feature if he knows the value. In this case the entropy-based method outperforms both utility and popularity ones. In the second, we assigned different acceptance probabilities to the features/questions, such that they are proportional to features' popularity, and with an overall acceptance rate that is close to the real one measured in a user study. The results of this second evaluation show that the new proposed methods perform better in reducing the result size because they select questions/features that will more likely be replied/used, and therefore they keep the interaction alive. The proposed methods are very simple to implement but they require some background knowledge of user behavior, such as feature popularity, and feature probabilistic dependency. In the future we plan to test more extensively these methods on additional data sets and to introduce a user model that can better inform the selection method about the features that more likely the user will exploit.

References

[1] A. Aamodt and E. Plaza. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.

[2] D. Aha and L. Breslow. Refining conversational case libraries. In *Case-Based Reasoning Research and Development, Proceedings of the 2nd International Conference on Case-Based Reasoning (ICCBR-97)*, pages 267–278. Springer, 1997.

[3] M. Doyle and P. Cunningham. A dynamic approach to reducing dialog in on-line decision guides. In *Advances in case-based reasoning: 5th European workshop, EWCBR-2000, Trento, Italy, September 6–9, 2000: proceedings*. Springer, 2000.

[4] A. Kohlmaier, S. Schmitt, and R. Bergmann. A similarity-based approach to attribute selection in user-adaptive sales dialogs. In *Proceedings of the 4th International Conference on Case-Based Reasoning*, volume 2080 of *LNAI*, pages 306–320, Vancouver, Canada, 2001. Springer.

[5] N. Mirzadeh, F. Ricci, and M. Bansal. Supporting user query relaxation in a recommender system. In *E-Commerce and Web Technologies, Proceedings of the 5th International Conference, EC-Web 2004*, LNCS 3182, pages 31–40, Zaragoza, Spain, August/Semptember 2004. Springer.

[6] M. J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13:393–408, 1999.

[7] F. Ricci, A. Venturini, D. Cavada, N. Mirzadeh, D. Blaas, and M. Nones. Product recommendation with interactive query management and twofold similarity. In A. Aamodt, D. Bridge, and K. Ashley, editors, *ICCBR 2003, the 5th International Conference on Case-Based Reasoning*, pages 479–493, Trondheim, Norway, June 23-26 2003.

[8] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.

[9] J. B. Schafer, J. A. Konstan, and J. Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1/2):115–153, 2001.

[10] H. Shimazu. ExpertClerk: Navigating shoppers buying process with the combination of asking and proposing. In B. Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, pages 1443–1448, Seattle, Washington, USA, August 4-10 2001. Morgan Kaufmann.