Comparison-Based Recommendation*

Lorraine Mc Ginty[1] and Barry Smyth[1,2]

[1] Smart Media Institute, University College Dublin, Dublin 4, Ireland.
{Lorraine.McGinty@ucd.ie}
[2] ChangingWorlds Ltd., South County Business Park, Dublin 18, Ireland.
{Barry.Smyth@ChangingWorlds.com}

Abstract. Recommender systems combine user profiling and filtering techniques to provide more pro-active and personal information retrieval systems, and have been gaining in popularity as a way of overcoming the ubiquitous information overload problem. Many recommender systems operate as interactive systems that seek feedback from the end-user as part of the recommendation process to revise the user's query. In this paper we examine different forms of feedback that have been used in the past and focus on a low-cost preference-based feedback model, which to date has been very much under utilised. In particular we describe and evaluate a novel comparison-based recommendation framework which is designed to utilise preference-based feedback. Specifically, we present results that highlight the benefits of a number of new query revision strategies and evidence to suggest that the popular *more-like-this* strategy may be flawed.

1 Introduction

Recommender systems represent the integration of technologies such as information filtering (predominantly collaborative and case-based), user profiling, machine learning, and adaptive user interfaces in an effort to better help users to find the information that they are looking for (for example [2, 7, 9]). For instance, a typical recommender system in the PC domain interacts with a user by retrieving PC descriptions from its product catalogue that are in line with the target PC description (query) specified by the user. Traditionally a similarity-based retrieval strategy has been adopted here, although more sophisticated approaches have been proposed (see for example [4]). Ideally, the accuracy of the recommendations produced improve on each user interaction with the system, until such time that the user finds the exact PC they are looking for.

A key feature that separates recommender systems from more conventional information retrieval technologies, such as search engines, is their conversational character. Search engines typically focus on *single-shot* retrieval - the user provides a query and the search engine responds with a list of results. In contrast,

* The support of the Informatics Research Initiative of Enterprise Ireland is gratefully acknowledged

recommender systems will often elicit feedback from the user during the recommendation process upon the presentation of initial or partial recommendations. This feedback is used to elaborate the user's query and so guide the next recommendation cycle.

Recently researchers have explored different ways of capturing feedback from users; such as asking users to specify individual feature values as search criteria (*navigation by asking* [11]), or inviting them to rate or grade recommendations as relevant or not relevant (*navigation by proposing* [11]). Feedback strategies often differ in terms of the level of effort required from the user. For example, requesting the user to provide a feature value requires a high-degree of user effort because the user not only has to consider which feature value she will provide, but must also have some understanding of the recommendation space and the value-space of the feature in question. In contrast, asking the user to rate or grade a recommendation demands significantly less effort. In this paper we are interested in the different ways that feedback can be captured from a user's interaction with a recommender system as part of the recommendation process. We describe and evaluate a novel comparison-based recommendation framework which uses preference-based feedback. Specifically, we highlight the benefits of a number of new query revision strategies and propose that the popular *more-like-this* strategy may be flawed.

2 Background

Feedback strategies can be classified in terms of the type of feedback that they aim to capture (feature-level vs. case-level feedback) and in terms of the cost to the user (low-cost vs. high-cost). Four different feedback strategies have been incorporated into recommender systems to a lesser or greater extent (see Figure 1). *Value elicitation* and *tweaking* are feature-level techniques in the sense that the user is asked to provide information about the features of a recommended case; in the former the must provide a specific value for a specific feature (eg. show me *PCs* whose *type* is *laptop*), while the latter requests that they simply express a directional preference for a particular feature value (eg. show me *PCs* that are cheaper that the current recommendation, or *PCs* that have increased memory). In contrast, *ratings-based* and *preference-based* feedback methods operate at the case-level. In the former the user is asked to rate or grade the recommended cases according to their suitability, while in the latter the user is simply asked to select one of the current set of recommendations that is closest to their requirements (or indeed farthest from their requirements).

2.1 Value Elicitation

Probably the most direct approach to eliciting user feedback during the recommendation process is to simply ask the user to provide a specific feature value (see for example [1]). For example in the PC domain the user may be requested

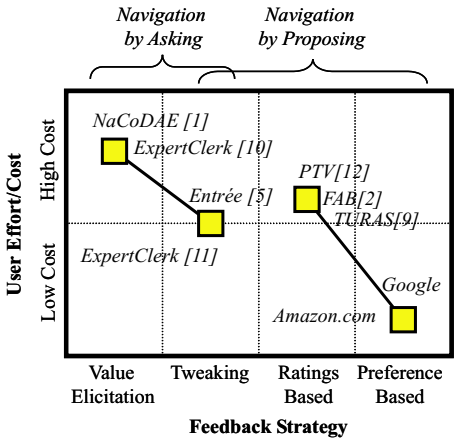

Fig. 1. A comparison of four feedback strategies used by recommender systems.

to indicate the *price* of the PC that they are looking for, or their preferred *manufacturer*, or whether they are looking for a *laptop* or a *desktop* model. The form of feedback provides the recommender with a significant degree of new information because each new feature value allows the recommender to eliminate a potentially large number of irrelevant cases that do not share this feature value (or a feature value that is similar); for example, recognising that the user is interested in buying a *laptop*, eliminates all *desktop* models from further consideration.

However, the value that this technique brings must be balanced against its cost. Research indicates that users do not like being asked direct and specialised questions during the recommendation process [6, 8, 11]. Furthermore, unless the user has a good understanding of the recommendation domain she may not be in a position to answer a direct feature question reliably. For example, how many ordinary users would be able to indicate *memory* preferences when buying a new PC? Forcing users to answer specific questions early on in the recommendation process may result in unreliable answers and ultimately eliminate relevant cases from recommendation. Finally, a key challenge with this approach is recognising which feature should be queried and when, since features can have different discriminating power at different times in the recommendation process.

The above problems notwithstanding, direct feature elicitation is probably the most common type of feedback used in recommender systems today and a variety of strategies have been forwarded to control the elicitation order. For example, Cunningham & Smyth describe an information theoretic approach to identify the next most discriminating feature [13]. They show that their incremental CBR approach out-performs the more traditional pure inductive learning approach across a number of problem domains. Similarly, Shimazu [11], in his work, calculates the information gain of asking possible questions, using an algorithm very similar to $ID3$ [10], that prioritises features that promise maximal information gain.

2.2 Tweaking

The FIND-ME group of recommender systems [5] is interesting for two main reasons. Firstly, they advance the *navigation by browsing* or *navigation by proposing* mode of operation, whereby a user is presented with a group of cases during each recommendation cycle and the next cycle is based on the feedback provided. But secondly, and perhaps more importantly for this work, is that the user is asked to express feedback in the form of a *tweak* or set of *tweaks*. In short the user implicitly selects one of the presented cases and expresses directional preferences for certain features. For instance, in the Entree restaurant recommender (see [5]), the user is presented with the single best case during each recommendation cycle and can introduce tweaks over features such as *price* (find me cheaper or more expensive restaurants) or *style* (find me a more casual or less casual restaurant), for example.

In a sense tweaking is a combination of navigation by asking and navigation by proposing - sequences of cases are proposed and the user is asked for specific information about particular features. The cost to the user is lower than value elicitation however because the user is not expected to provide specific feature values, of which they may have little or no knowledge. This strategy also combines elements of preference-based feedback (see below) since the user must explicitly identify a *preferred case* over which the tweaks should be applied.

2.3 Ratings-Based

Recently, ratings-based feedback has become a popular form of feedback in recommender systems. The basic idea is to ask the user to rate individual recommendation proposals; as such the user is providing case-level feedback. For example, PTV is a recommender system operating in the TV listings domain [12]; it compiles personalized TV guides for users based on their learned TV preferences and as such is involved in recommending TV programmes to users. PTV encourages users to rate its programme recommendations (using a 5-point scale from strong-negative to strong-positive) as a means of fine tuning subsequent recommendation cycles.

Generally speaking ratings-based feedback is a relatively low-cost form of feedback since the user need only express a qualitative (or perhaps quantitative) indication of interest at the level of an individual case. Having said this, the level of effort naturally increases if the user needs to rate many or all of the recommended items - although in most systems, such as PTV, the user can chose to rate as few or as many of the items as she desires.

2.4 Preference-Based

Perhaps the lowest cost form of feedback is for the user to express a simple positive or negative preference for one of the recommended cases. Strangely enough this strategy has been relatively under-utilized by recommender systems to date. Maybe this is because the information-content associated with this simple form of

feedback is assumed to be very low, and hence it is expected to provide very little advantage for subsequent recommendation cycles. Nevertheless, it has become a popular form of feedback in many search engines. Google for example allows users to request pages that are similar to a specific search result - in essence the user is expressing a preference for a specific page and requesting more like it during the next recommendation cycle. Similarly *Amazon.com* provides a *more like this* feature for its search results with equivalent functionality.

As mentioned above, this form of feedback can be viewed as a subset of tweaking in the sense that the user is just asked to express an individual positive or negative preference without needing to indicate specific feature tweaks. Similarly it has a lower cost than ratings-based feedback because the user is not expected to distinguish between different levels of preference.

This form of feedback is the core focus of our current work. We are interested in it primarily for two reasons. First its low-cost makes it particularly well suited for use in recommender systems. And secondly, we propose that the assumption that it provides a recommender with little information, and thus does not add value to current recommender systems, is flawed. In the the next section we will introduce the comparison-based recommendation approach, which is designed to better exploit preference-based feedback by transforming the user's preference into explicit query adaptations.

3 Comparison-Based Recommendation

Comparison-based recommendation is a navigation by proposing type of recommendation process. The basic algorithm (fully described in Figure 2) consists of 3 main steps: (1) new items are *recommended* to the user based on the current query; (2) the user *reviews* the recommendations and selects a preference case (positive or negative) as *feedback*; (3) information about the difference between the selected item and the remaining alternatives is used to *revise* the query for the next cycle. The recommendation process terminates either when the user is presented with a suitable item or when they give up.

3.1 Item Recommendation

As with traditional recommender systems the goal of the recommendation step is to select a set of k items that closely match the current user query. However, one of the key features of our comparison-based technique is its ability to make query elaborations on the basis of differences between these recommended items. Thus, it is important to ensure, where possible, that the recommendations presented to the user are diverse in addition to being similar to the target query. As such the comparison-based recommendation technique can be used with either a traditional similarity-based recommendation process (whereby items are selected based on the their similar to the current query) or a diversity-preserving technique, such as that presented by [3, 14]. For reasons of clarity however, we will assume a standard similarity-based recommendation process in this work.

```
1.    define Comparison-Based-Recommend(Q, CB, k)
2.    begin
3.      Do
4.       R    ItemRecommend(Q, CB, k)
5.       c_p   UserReview(R, CB)
6.       Q    QueryRevise(Q, c_p, R)
7.      until UserAccepts(c_p)
8.    end
-----------------------------------------------------------------
9.    define ItemRecommend(Q, CB, k)
10.   begin
11.     CB'    sort cases in CB in decreasing order of their sim to Q
12.     R    top k cases in CB'
13.     return R
14.   end
-----------------------------------------------------------------
15.   define UserReview(R, CB)
16.   begin
17.     c_p   user selects best case from R
18.     CB    CB - R
19.   return c_p
-----------------------------------------------------------------
20.   define QueryRevise(Q, c_p, R)
21.   begin
22.     R'    R - {c_p}
23.     For each f_i   c_p
24.       Q    update(Q, f_i, R')
25.     end For
26.     return Q
27.   end
```

Fig. 2. Outline of the comparison-based recommendation algorithm.

3.2 User Feedback

The review process is conceptually simple. The end user is asked to select that case, from among the recommended items, which best matches the needs of the user; this is a form of *positive* feedback. Alternatively the user can provide *negative* feedback in the form of a case which best corresponds to items that the user is not interested in. The power of comparison-based recommendation is its ability to predict how this feedback can be used to elaborate the current query in order to better reflect the user's implicit needs. Once again, for the sake of clarity, in this paper we will focus on positive feedback.

3.3 Query Revision

The key to the success of comparison-based recommendation is the way in which the current query is modified to reflect the user's preference feedback. During this step, the aim is to update the current query based on what can be learned from the user's feedback. In the case of positive feedback (where the user selects a preferred case, c_p) the ultimate objective is to update query features with features from the preferred case that best reflect the user's implicit preferences. The following paragraphs outline the update strategies of interest in this work and their associated update rules.

MLT - More Like This The simplest form of query revision is to take each feature of the preferred case, $f_i \in c_p$, as a new query feature (see MLT Rule given

below). This corresponds to the *more like this* strategy used by some search engines. The advantage that it has is that a partial user query can be instantly extended to include additional features (from the preferred case) in the hope of focusing more effectively on the right set of cases during the next recommendation cycle. However, the potential danger is that not every feature of the preferred case may be actively preferred by the user; for instance, the user may have preferred a PC because of its low *price* and not because of its low *processor speed*. Because of this we can expect the MLT update strategy to suffer from a form of overfitting to user feedback and as a result we may find that the user is guided to irrelevant parts of the recommendation space based on their early preferences. This can result in long recommendation cycles and the need for the user to examine more cases than necessary. Nevertheless the strategy is a useful benchmark against which to judge more principled alternatives.

***MLT RULE*: update**(Q, f_i, R): $Q.f_i := f_i$

pMLT - Partial More Like This Consider the scenario shown in Figure 3. The preferred case is for a laptop with a 15 inch display. According to the MLT strategy, this means that the user is likely to be interested in more PCs with *15 inch* displays and as such this feature value should be added to the query. However, in this situation we notice that the user has also rejected a case with a *15 inch* display, suggesting that perhaps the *display size* is not the reason for the user's preference. The pMLT strategy captures this idea by only transferring a feature value from the preference case if none of the rejected cases have the same feature value. For example, we can transfer the preference for *Compaqs* to the new query because none of the rejected cases are *Compaqs*. Similarly, the query can be updated with a preference for a *40GB hard-disk* (see pMLT Rule below).

***pMLT RULE*: update**(Q, f_i, R): $Q.f_i := f_i$ if $(\neg \exists c_j \in R' : c_j.f_i = f_i)$

wMLT - Weighted More Like This How reliable is the decision to include *Compaq* in the new query? Should the preference for *15 inch displays* be ignored completely simply because it exists in one of the rejected cases? How does the user's preference for *Compaq* relate to their preference for *Pentiums*? The wMLT strategy attempts to weight the new query features based on the degree of confidence that we can attribute to them as preferences for the user. The basic idea is that the more alternatives that have been presented to the user, the more confident we can be about learning their preference. For example, we can be confident in their *Compaq* preference because there was a wide range of alternatives for the *manufacturer* feature. Compare this to the *processor type* feature, where only two alternatives were presented, *Celeron* and *Pentium*; we can not be sure whether the user is genuinely interested in *Pentiums* or whether they are simply more *interested* in *Pentiums* than *Celerons*.

Features	C1	C2	C3	C4
Manufacturer	Dell	Sony	Compaq	Fujitsu
Memory	256	128	128	256
Monitor	15	14	15	12
Type	Laptop	Laptop	Laptop	Laptop
Processor	Celeron	Pentium	Pentium	Celeron
Speed	700	600	700	650
Disk	20	20	40	20
Price	1300	999	1200	1000

Fig. 3. A typical recommendation scenario where a user is presented with a number of alternatives (PC descriptions) and are asked to choose between them.

The wMLT strategy transfers all preferred features, ($f_i \in c_p$), to the new query but weights them according to the formula shown in Equation 1 which gives preference to diverse features among the recommended set R (see wMLT Rule below). For example, the preference for *Compaqs* gets a weighting of 1 because all 3 of the rejected cases have unique *manufacturer* values that are different from *Compaq*. In contrast, the preference for *Pentiums* gets a weight of 1/3 since the 3 rejected cases only account for one alternative to *Pentiums*.

***wMLT RULE*: update**(Q, f_i, R): $Q.f_i :=< f_i, weight >$

$$weight(f_i, R) = \frac{\#\ of\ alternatives\ to\ f_i\ in\ R}{|R|} \tag{1}$$

LLT - Less Like This In addition to learning about features that the user is likely to prefer it may also be possible to learn features that the user tends to dislike. The LLT strategy attempts to do this by treating the query as a set of negative preferences and ordering cases during future cycles by prioritising dissimilarity to the negative query.

The LLT strategy is a simple one: if the rejected cases all have the same feature-value combination, which is different from the preferred case then this combination can be added to a negative query (see LLT rule below). For example in Figure 3, all of the rejected cases have *20 Gb disks* whereas the preferred case has a *40 Gb disk*. The LLT strategy assumes that the user is not looking for a *20 Gb disk* and updates the negative query accordingly.

***LLT RULE*: update**(Q, f_i, R): $Q_{neg}.f_i := f_i'$ if $(\forall c_j \in R', c_j.f_i = f_i') \wedge (f_i' \neq f_i)$

MLT+LLT - More Like This + Less Like This Our final strategy combines the MLT and LLT strategies to learn both positive and negative preferences in a

single cycle according to the MLT and LLT rules. In practical terms this means that the query has two components, a positive part and a negative part. To compute the degree of relevance between such a query and a case we simply compute the similarity between the positive query component and the case and subtract the similarity between the negative query component and the case (see MLT+LLT Rule below). Other approaches could be used here but for reasons of simplicity and clarity the above has been chosen.

$$\textbf{\textit{MLT+LLT RULE}: update}(Q, f_i, R)\text{: } Q_{pos}.f_i := f_i \text{ if } (\neg\exists c_j \in R' : c_j.f_i = f_i) \bigwedge Q_{neg}.f_i := f_i' \text{ if } (\forall c_j \in R', c_j.f_i = f_i') \wedge (f_i' \neq f_i)$$

4 Experimental Evaluation

So far we have presented a novel recommendation framework called comparison-based recommendation which attempts to optimise the use of preference feedback as a means of revising the user's query within a navigation by proposing recommendation approach. Furthermore, we have presented a variety of possible strategies for revising the query. In this section we evaluate the comparison-based recommendation technique and these individual revision strategies with respect to a benchmark recommendation system.

4.1 Setup

The following experiments were carried out using a case-base from the PC domain. This case-base contains 120 unique cases, each describing a unique PC in terms of features such as *manufacturer, PC type, processor type, processor speed, memory, disk size, display size, and price.* The cases themselves are available for download from URL *www.cs.ucd.ie/students/lmcginty/PCdataset.zip.*

In total we evaluate 5 different versions of our recommendation technique, each employing a different revision strategy and labeled according to this strategy (MLT, pMLT, wMLT, LLT, MLT+LLT). In addition, we chose to implement a simple similarity-based recommender (Sim) as a benchmark against which to judge the performance of each of the comparison-based techniques.

In each of the following experiments we use a leave-one-out evaluation methodology varying values of k (the number of cases returned during each recommendation cycle) from 2 to 5, and q (the number of initial features per query) from 1 to 5. In brief, each case of the case-base is temporarily removed and used in two ways. First each case serves as the basis for a set of random queries of varying sizes (ranging from 1 to 5 features per query). Second, we select that case in the case-base which is most similar to the current target case. These *best cases* serve as the recommendation targets during the experiments. In other words, the original target case is taken as the ideal query for a user, a generated query serves as the initial query that the user provides to the recommender, and the *best case* is the best available case for the user based on their ideal query. In

total a set of 1200 separate queries are generated and all reported results are averaged as appropriate.

4.2 Recommendation Efficiency

Ultimately, a key success criterion for any navigation by proposing recommender is the number of individual cases that the user is expected to examine before they settle on a *best case*. This metric has a direct analogy in the case of the similarity-based benchmark.

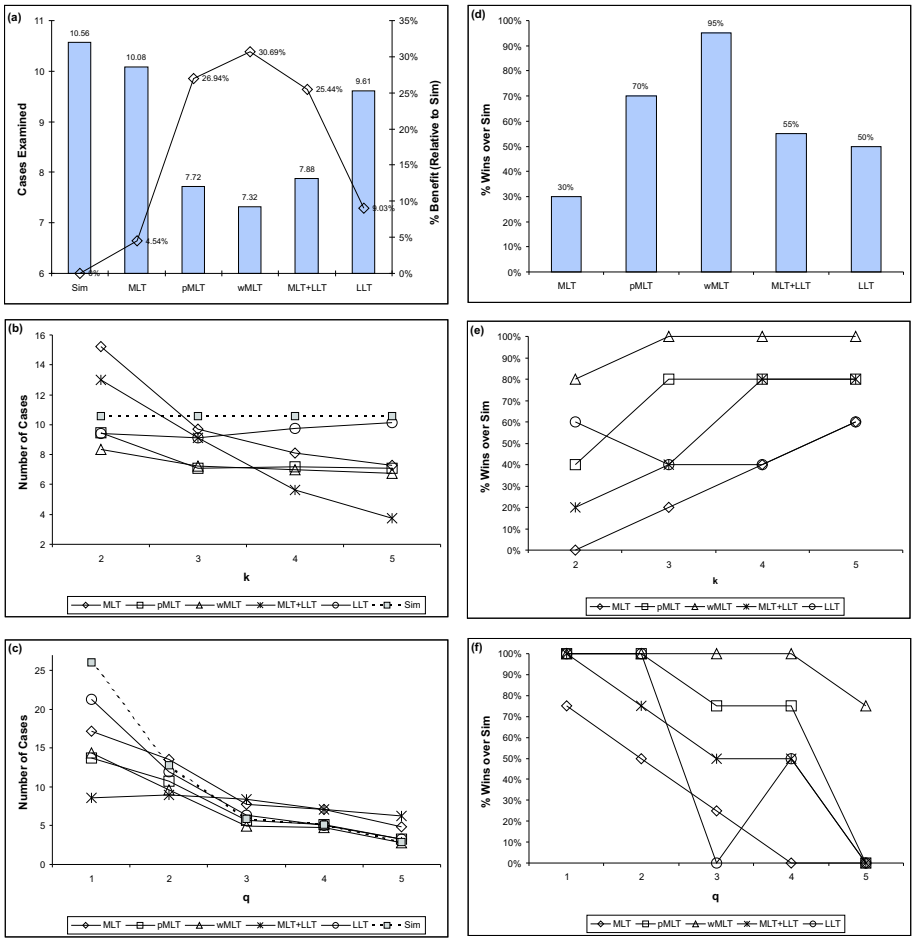

Fig. 4. Summary evaluation results looking at recommendation efficiency (a-c), and winners and losers(d-f).

Method In this experiment we run leave-one-out trials for each combination of k (2 to 5) and q (1 to 5) - a total of 20 separate trials, each one over 1200 queries - and note the average number of cases that the user must view before the best case is located. In the case of the similarity-based benchmark this is the average position of the best case in the similarity ordered list produced from the initial queries. In the case of the comparison-based recommenders it is the total number of cases presented in each recommendation cycle prior to the final cycle plus the position of the best case in the final cycle. For example, for a $k = 3$ trial where 3 recommendation cycles are needed and the best case is the second case in the final cycle, then the user has needed to look at a total of $(3 * 2) + 2 = 8$ cases.

Results The results are shown in Figure 4(a-c). The first graph plots the average number of cases examined for each technique over all trials as a bar chart, and also plots the relative benefit associated with each technique, compared to the benchmark (Sim), as a line graph. Figures 4(b&c) then show a breakdown of these results averaged over different values of k and q respectively.

Summary Analysis Figure 4(a) indicates that overall the comparison-based recommenders are working well in relation to the benchmark even though they operate on the basis of minimal feedback from the user during each recommendation cycle. The best case for a query is positioned at 10.56 on average for the Sim benchmark, compared to 7.32 for the wMLT method. In other words, the wMLT method requires the user to look at 30% fewer cases than Sim, on average. Similar, albeit not quite as pronounced, benefits are available to the pMLT and MLT+LLT methods, with reductions of approximately 27% and 25%, respectively. The MLT and LLT methods on their own perform less well, as they require the user to examine an average of 10.08 and 9.61 cases per session (benefits of 4.54% and 9.03%, respectively).

Varying k Figure 4(b) indicates that as k increases the average number of cases examined by the user decreases (of course the Sim plot remains flat here as it does not depend on k). However, we also see that while the MLT and MLT+LLT techniques outperform Sim on average, they actually lose to Sim when $k = 2$; MLT and MLT+LLT rack up scores of 15.2 and 12.99 respectively compared to Sim's score of 10.56. It seems that for low values of k the recommendation window is not large enough for MLT to generate sensible revisions for the query. Note also that the LLT method on its own performs well at the $k = 2$ level indicating that the poor performance of MLT+LLT at $k = 2$ is largely due to MLT and not to its LLT component. In general, the results suggest that the simple MLT strategy is not an effective revision strategy as all techniques bar LLT perform better for all values of k. Interestingly the combination of MLT and LLT performs much better at higher values of k to eventually win outright for $k > 3$. Overall the pMLT and wMLT techniques perform best of all for the

different values of k, with the more sophisticated wMLT method performing slightly better than pMLT for all but $k = 3$. This suggests that the weighting scheme employed by wMLT does have a positive effect.

Varying q Figure 4(c) shows the average results (over the different k values) for different values of q, the initial query size. This time the Sim method is affected, with significant reductions in the number of cases that must be examined as q increases, as expected. Once again the MLT method and MLT+LLT methods perform poorly here, losing to Sim for values of $q > 2$ in the case of MLT and $q > 3$ in the case of MLT+LLT; although it is interesting that MLT+LLT is the outright winner for $q = 1$ and $q = 2$. The best overall performers are pMLT and wMLT, which perform well for $q = 1$ and $q = 2$ (they are only beaten by MLT+LLT), winning outright at $q > 2$. And notably, wMLT outperforms pMLT indicating the additional value conferred by its weighting technique.

Discussion The results so far indicate that the simple (and naive) MLT technique does not perform well when compared even to the simple Sim benchmark. As predicted, MLT tends to overfit to the current preference case and this serves to steer the recommender into parts of the case-base that do not contain the target case. The more sophisticated pMLT and wMLT methods offer significant improvements across all of the recommendation trials as they successfully select features from the preference case for transfer to the new query. The influence of negative preference features is less clear, with some benefits available for lower values of k and higher values of q. In particular the interaction between MLT and LLT seems to be inconsistent and warrants further analysis.

4.3 Winners & Losers

The previous experiment examined average case results in terms of the numbers of cases that need to be examined by a user for each recommendation technique. However it is important to understand that these average case results do not tell us whether one technique is consistently performing better than another (especially the benchmark). In fact, a low average could be the result of a single or small number of good trials for an otherwise poorly performing method. For this reason, in the following experiment we look at the number of trials where an individual technique wins, compared to the benchmark (and outright).

Method The trial results are processed to compute the number of times that each recommendation technique wins over the benchmark; that is the number of trials where a technique results in fewer cases being examined than the Sim benchmark.

Results The results are shown in Figure 4(d-f) as a summary over all trials (Figure 4(d)) and individual break-downs for differing values of k and q (Figures 4(e) and (f) respectively).

Summary Analysis Figure 4(d) indicates that the wMLT method is the best performer overall, beating Sim 95% of the time, that is, in 19 out of 20 of the trials; the trial where wMLT looses to Sim is with $k = 2$ and $q = 5$ and the difference is only 0.02. pMLT ranks second overall with a win rate of 70% (14 out of 20 trials). Once again we find that MLT performs poorly, and even though it beats Sim in terms of the average number of cases examined overall (10.56 vs. 10.08) it wins over Sim in only 30% of the trails (in 6 out of the 20 trials); in other words its improved average is derived from good results on a few trials. Once again we find that the LLT method has some promise, beating Sim 50% of the time. Interestingly, the combination of MLT and LLT performs only marginally better than LLT on its own, indicating the the MLT technique is having a minor positive impact overall, and even though MLT+LLT performed well on average in terms of number of cases examined by the user; once again this good average result is derived from a few good trials and is masking poor individual performances compared to Sim.

Varying k The results when k is varied are summarised by Figure 4(e). Once again we find wMLT performing best overall, beating Sim 100% of the time for $k > 2$. Similar, albeit less impressive, performance is seen for pMLT as it beats Sim for 40% of $k = 2$ trials (2 out of 5 of these trials) and for 80% of the $k > 2$ trials. Interestingly the performance for LLT is seen to degrade for $k > 2$ reflecting the average increase in the number of cases examined for LLT witnessed in the previous experiment. MLT is the worst performer, never beating Sim at $k = 2$ but improving to beat Sim on one extra trial for each increment of k (these wins come for low values of q).

Varying q The results when q is varied are shown in Figure 4(f). In general the performance of each of the comparison-based techniques, relative to Sim, degrades with increasing values of q; this is to be expected as the Sim technique will perform well when queries become more and more complete. However it is worth pointing out that the wMLT technique maintains its 100% win rate for all values of q except $q = 5$ where it looses once as indicated above. Moreover, wMLT is the only technique to beat Sim at all at the $q = 5$ level, so the fact that it wins in 3 out of the 4 trials at this q value is impressive by comparison. The MLT and MLT+LLT techniques degrade rapidly with increasing values of q as shown while pMLT performs better until $q = 5$ where it too succumbs to the superiority of Sim. The LLT strategy is somewhat erratic in this analysis, in general performing poorly, but beating Sim anomalously in 2 out the 4 trials at $q = 4$.

Discussion The results in this section add more detail to the average case analysis of the previous experiment, and indicate that some of the good average-case performances are the result of a small number of good trials, and are not indicative of good performance overall, compared to the benchmark. This is especially true for the MLT, LLT, and MLT+LLT techniques. We find that the

good average-case performances of pMLT and wMLT do represent genuine improvements over Sim across the board. The good performance of pMLT indicates that its ability to select features for the new query works well, and the winning performance of wMLT indicates that its weighting techniques adds considerable further value. In fact the success of wMLT is highlighted in Figure 5(a & b) which charts the percentage of trials that wMLT beats each other individual technique and the percentage of trials that wMLT wins outright. Figure 5(a) shows that wMLT displays as impressive a performance when compared to other update strategies as it does when compared to Sim (see Figure 4(d)), consistently beating LLT, MLT, and pMLT, and beating MLT+LLT in 60% of the trials. Impressively, Figure 5(b) highlights that wMLT wins outright (beating every other technique) in 50% of the trials (10 out of 20 trials).

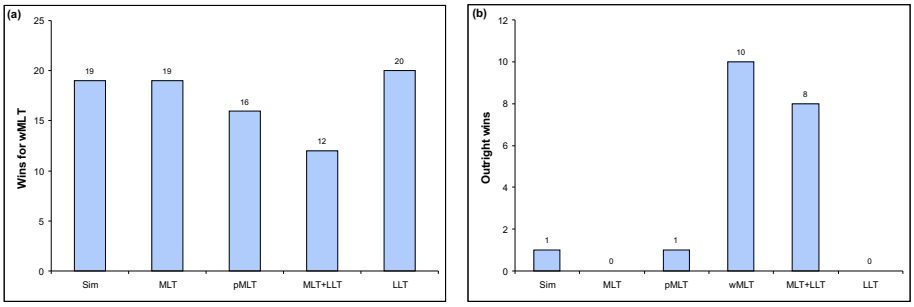

Fig. 5. Comparison of the performance (win-rate) of each of the feedback strategies examined compared to wMLT (a), and relative to each other (b).

5 Conclusions

Our focus in this paper has been the various feedback methods that have been, or can be, employed by recommender systems as a means of incrementally refining the recommendation process. We have attempted to classify the different types of feedback that have been explored to date along granularity and cost dimensions. We have further highlighted the apparent lack of interest in the low-cost preference-based feedback method, suggesting that this apathy is rooted in the assumption that this technique is unlikely to gather sufficient information to effectively guide the recommendation process.

In response we have outlined the comparison-based recommendation framework which is designed to fully exploit preference-based feedback. A family of preference-guided query revision strategies has been fully described and evaluated. We found promising results for the pMLT and the wMLT methods which are based on principled ways of selecting and weighting features in the revised query as a result of preference feedback. Furthermore we have provided evidence that the simplest form of preference feedback (MLT - *more like this*), which has been widely adopted by the search-engine community, is likely to perform poorly

in recommender systems. Future work will focus on extending our evaluation into additional recommendation domains and into looking at new revision strategies based on alternative weighting models.

References

1. D.W. Aha, L.A. Breslow, and H. Muoz-Avila. Conversational case-based reasoning. *Applied Intelligence*, 14:9–32, 2000.
2. M. Balabanovic and Y. Shoham. FAB: Content-Based Collaborative Recommender. *Communications of the ACM*, 40(3):66–72, 1997.
3. K. Bradley and B. Smyth. Improving Recommendation Diversity. In D. O'Donoghue, editor, *Proceedings of the Twelfth National Conference in Artificial Intelligence and Cognitive Science (AICS-01)*, pages 75–84, 2001. Maynooth, Ireland.
4. D. Bridge. Product Recommendation Systems: A New Direction. In D. Aha and I. Watson, editors, *Workshop on CBR in Electronic Commerce at The International Conference on Case-Based Reasoning (ICCBR-01)*, 2001. Vancouver, Canada.
5. R. Burke, K. Hammond, and B.C. Young. The FindMe Approach to Assisted Browsing. *Journal of IEEE Expert*, 12(4):32–40, 1997.
6. M. Doyle and P. Cunningham. A Dynamic Approach to Reducing Dialog in On-Line Decision Guides. In E. Blanzieri and L. Portinale, editors, *Proceedings of the Fifth European Workshop on Case-Based Reasoning, EWCBR-2000*, pages 49–60. Springer, 2000. Trento, Italy.
7. M. Goker and C. Thompson. Personalized Conversational Case-based Recommendation. In E. Blanzieri and L. Portinale, editors, *Advances in Case-Based Reasoning: Proceedings of the Fifth European Workshop on Case-based Reasoning*, pages 99–111. Springer-Verlag, 2000.
8. A. Kohlmaier, S. Schmitt, and R. Bergmann. Evaluation of a Similarity-based Approach to Customer-adaptive Electronic Sales Dialogs. In S. Weibelzahl, D. Chin, and G. Weber, editors, *Empirical Evaluation of Adaptive Systems. Proceedings of the workshop held at the 8th International Conference on User Modelling*, pages 40–50, 2001. Sonthofen, Germany.
9. L. McGinty and B. Smyth. Collaborative Case-Based Reasoning:Applications in Personalised Route Planning. In D. Aha and I. Watson, editors, *Proceedings of the International Conference on Case-Based Reasoning (ICCBR-01)*, pages 362–376. Springer-Verlag, 2001. Vancouver, Canada.
10. J.R. Quinlan. Induction of decision trees. *Journal of Machine Learning*, 1:81–106, 1986.
11. H. Shimazu. ExpertClerk : Navigating Shoppers' Buying Process with the Combination of Asking and Proposing. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, pages Volume 2, pages 1443–1448. Morgan Kaufmann, 2001. Seattle, Washington.
12. B. Smyth and P. Cotter. A Personalized TV Listings Service for the Digital TV Age. *Journal of Knowledge-Based Systems*, 13(2-3):53–59, 2000.
13. B. Smyth and P. Cunningham. A Comparison of Incremental Case-Based Reasoning and Inductive Learning. In *Proceedings of the Second European Workshop on Case-Based Reasoning, EWCBR-94*. Springer, 1994. Chantilly, France.
14. B. Smyth and P. McClave. Similarity v's Diversity. In D. Aha and I. Watson, editors, *Proceedings of the International Conference on Case-Based Reasoning*, pages 347–361. Springer, 2001.