An on-line evaluation framework for recommender systems

C. Hayes[1], P. Massa[2], P. Avesani[2], and P. Cunningham[1]

[1] Trinity College Dublin, Ireland
{cunningham,cchayes}@cs.tcd.ie
[2] ITC-IRST, Via Sommarive 18 - Loc. Pantè, I-38050 Povo, Trento, Italy
{avesani,massa}@irst.itc.it

Abstract. Several techniques are currently used to evaluate recommender systems. These techniques involve off-line analysis using evaluation methods from machine learning and information retrieval. We argue that while off-line analysis is useful, user satisfaction with a recommendation strategy can only be measured in an on-line context. We propose a new evaluation framework which involves a paired test of *two* recommender systems which *simultaneously* compete to give the best recommendations to the *same* user at the *same* time. The user interface and the interaction model for each system is the same. The framework enables you to specify an API so that different recommendation strategies may take part in such a competition. The API defines issues such as access to data, the interaction model and the means of gathering positive feedback from the user. In this way it is possible to obtain a relative measure of user satisfaction with the two systems.

1 Introduction

Acting upon recommendations from other people is a normal part of life. We do it when we eat at restaurant on the advice of a friend, or we see a movie having read the review in the newspaper of our choice. In each case our decision to act upon a recommendation is based on essentially three premises: first, we trust the recommender; second, we assume that the recommender has sufficient knowledge of our tastes or of the tastes of people like us; third, we assume that the recommender has knowledge of the alternatives available.

By using recommendations we can take a shortcut to the things we like without having to try many things we dislike or without having to acquire all the knowledge to make an informed decision. Unsurprisingly, systems that automate this facility have become popular on the Internet. Irrespective of the techniques used, the success of the automated recommender is still reliant upon the trust of the user, having sufficient knowledge of the user's requirements, and having knowledge of the range of items available.

Increasingly, there has been a demand for objective evaluation criteria for these types of systems. This stems from a difficulty in evaluating which recommender is better than another, and in judging which criteria to use when

making this evaluation. There are many aspects of a recommender system we could analyse [12] - for instance the ease with which it can be used is certainly an important factor for success. This has much to do with HCI factors such as presentation and the interaction model employed. The most common evaluation approaches are performed off-line using techniques from machine learning and information retrieval such as cross validation and measures of recall/precision.

In this paper we present an evaluative framework for recommender systems based on the idea of system utility - a comparative measure of how one recommendation strategy performs against another. We draw attention to the fact that evaluation has to measure whether real people are willing to act based on the advice of the system. Therefore it is necessary to evaluate recommendation strategies as part of live, fully realised applications. In sections 2 and 3 we introduce related evaluation methods used by the machine learning and information retrieval community and we propose that, while these techniques may provide useful insights, only an on-line evaluation methodology can truly gauge user satisfaction with a recommendation strategy. Section 4 introduces a new evaluation methodology to compare the utility of different recommendation strategies running in a fully realised application. Unlike the off-line analysis, this methodology plays one recommendation strategy against the other in an on-line setting and measures the *relative* degree of success of each strategy according to how the user utilises the recommendations of either system. This framework doesn't measure absolute user satisfaction but only relative user satisfaction with one system over another. We examine how we might maintain user trust during the evaluation period, and how we might offer each recommendation engine equal opportunity for success. Finally, in section 5 we discuss some advantages and some open issues related to the deployment of the proposed methodology.

2 Existing Approaches

Konstan and Riedl [9] suggest that existing approaches to evaluating recommender systems can be divided into two categories:

Off-line evaluation: where the performance of a recommender mechanism is evaluated on existing datasets.

On-line evaluation: where performance is evaluated on users of a running recommender system.

Konstan and Riedl argue that on-line evaluation is problematic because of the need to field a fully engineered system and build up a community of users. Consequently they favour off-line evaluation, not because it is better but because it is easier to do.

Off-line evaluation uses existing datasets such as the publicly available EachMovie dataset or on data gathered during the operation of a recommender prototype [8]. In off-line evaluation, recommendation can be seen as information retrieval, i.e. the selection of the subset of assets that are relevant to the user. From this perspective the metrics for evaluation are the well known measures of precision and recall [3]. Precision is the proportion of retrieved assets that

are relevant and recall is the proportion of relevant assets actually retrieved. Basu and Hirsh [4], in evaluating their recommender system, defined the upper-quartile of the movies rated by the user as the relevant set. So the precision is the percentage of upper-quartile movies in the set returned.

The alternative to viewing recommendation as an information retrieval problem is to view it as a classification or regression problem. In a situation where users have rated assets, the recommendation problem may be cast as the prediction of these ratings - a regression problem. Alternatively, it may be viewed as a classification problem - the classification being whether an asset will be liked or disliked. The measure of accuracy may be: 0/1 error for classification, absolute or root-mean-square (RMS) error for regression or a measure of the correlation of predicted ratings with actual ratings (e.g. Pearson's correlation coefficient). This allows for the type of evaluation that is common in Machine Learning where the data is partitioned into training and test data - using the training data to produce predictions for the test data. These estimates may be improved by using k-Fold Cross Validation or by using a Leave-One-Out evaluation, i.e. a rating is predicted by using all the data except the rating itself [5].

An interesting on-line approach has been taken by Swearingen and Sinha [11] whereby recommendations from recommender engines are compared to recommendations from friends. In the next section, we will outline the drawbacks in only using off-line analysis. We will suggest that the datasets used for this evaluation ignore the context in which recommendations are made. Users may give assets quite different ratings depending on their interests or information needs at the time. Elaborating the work of Swearingen and Sinha we will propose a methodology to compare the relative strengths of two recommendation strategies, both automated.

3 Evaluating Contexts

Recommender systems generally operate as augmentative systems within larger application systems. Intuitively, their purpose is to help the user exploit the resources available within the larger application domain. Therefore, when employing a recommender system, the goal is to translate continued user satisfaction into continued use of the system resources. It is the continuing use of these resources based on the advice of the recommender which requires analysis when conducting an evaluation of a recommender system.

How does a recommendation strategy maintain user satisfaction? Generally, a recommendation engine will present a list of n resources based on the user's feedback or a user profile. The simplest way to do this is present an ordered list of recommendations based on a score calculated by an algorithm. This score might reflect the similarity of a resource to user profile/query or a score predicted by a collaborative filtering algorithm. However, a recommendation engine may choose other strategies to present a relevant list of resources to the user.

Smyth and McClave [10] show that a recommendation set that contains too many similar items may be highly redundant whereas a small, diverse set would

offer the user more choice. Swearingen and Sinha's research [11] would suggest that including "trust generating items" in a recommendation set is perceived as being 'useful' by the user. In the area of music recommendation, Hayes et al. [7] introduce a notion of context sensitivity that boosts recommendations that are pertinent to the users' current listening habits.

In each of these cases there is a concern for maintaining the trust of the user by providing items that are useful within the user's context. We define context as the discourse which informs the users' current behaviour in the system - their current requirements, their motivation, their previous experience, their preferences and the knowledge available to them.

3.1 Problems with existing approaches

In Section two we described an evaluation of a recommender system from a machine learning perspective which treated recommendation as a classification or a regression task. There are a number of problems with an off-line approach like this.

Classification and regression are well established machine learning techniques and can be applied with many of the datasets available from the UCI machine learning repository. Typically, a measure of generalisation error indicates whether a classification/regression model is performing well. However, there is no evidence to suggest that people are sensitive to minor improvements in generalisation error as far as algorithmic performance is concerned. In fact standard social research methodologies include metrics such as the test-retest correlation to account for the variance that occurs when people are given the same test on two different occasions. Hill et al. [8] observed a reliability correlation of of 0.83 on ratings taken six weeks a part in the Bellcore project. This situation is complicated by the fact that users may not always give explicit feedback on recommendable items. More noise is introduced where the system must infer an implicit score based on the user's usage of the system. This factor is exacerbated where there is latency between the time a recommendation is made and when it is ultimately 'consumed'.

The recommendation task, however, is not as well defined as the classification/regression task. It would be a mistake to assume that domain specific datasets such as those from the EachMovie and MovieLens projects can be used to test every Recommendation System. These datasets contain a date-time field which allows the use-data to be ordered. However, current evaluation practices do not make use of this information. For classification/regression purposes each data point is treated as a set of un-ordered feature-value pairs. Using the un-ordered dataset one algorithm is judged to have better generalisation error than another. While this may provide us with insight into the success or failure of a recommendation strategy, it gives us no knowledge as to whether lower generalisation error translated into continued use of the system. Nor do we have any indication whether resources were used as a result of a recommendation. Indeed, the drawbacks of off-line evaluation become quite apparent when we consider

how we might use this date-time information. Let us consider an evaluation scenario where we simulate the usage patterns of each user in the dataset. Using the temporal constraints of the dataset we make predictions for each simulated user at discrete intervals of time. At each time slice the recommender engine may only use the data in the system up to that instant. After each prediction we evaluate whether the user actually goes on to use the items we predict. If the user doesn't use that item the prediction of our algorithm shouldn't necessarily be considered a false positive since the test data was collected under the advice of another algorithm. We have very little information to indicate whether our prediction would have been successful using a data set biased by another recommendation strategy. This example enforces the view that an on-line analysis captures use-context. Context is something which is affected by the recommender algorithm itself. We suggest that you cannot successfully evaluate one algorithm using the data from another algorithm, even in a simulated test where temporal constraints are observed. Since the dataset is tied to the methods and domain in which it was collected, it is inappropriate that two datasets become the test data for evaluating every recommendation strategy.

However, in order to accurately measure the effectiveness of a recommender system in an off-line situation we would need to capture an unfeasibly large amount of state information about the user's interaction with the system. In the next section we propose a new evaluation framework which involves an on-line paired test of *two* recommender systems in which we are able to define a relative measure of user satisfaction without having to capture the implicit notion of context.

4 Methodology

In this section we outline a methodology for evaluating two competing recommendation strategies. Our evaluation environment consists of a real on-line application used by a community of users, with a well defined recommendation task using a specific user interface. The application is serviced by two competing recommendation engines. In order to be able to gauge a relative measure of user satisfaction with the two strategies, it is necessary to deploy the overall application and to log the user interactions with respect to the recommendation engines. By comparing usage of the recommendations, it will be possible to say which strategy performed better than the other. In order to isolate the recommendation strategies we keep other aspects which influence user satisfaction (interface, interaction model) the same. Using the system, the user is unaware of the source of recommendations. The proposed methodology can be seen as a competition between two different approaches to solving the same problem (in this case, winning user satisfaction) in which the winner is defined by the how the user makes use of recommendations. We will define a framework for the rules of the competition. This framework will deal with issues such as how to define access to the resources for making recommendations and how to define a winner.

The architecture of the framework is very similar to a standard recommender system's architecture. The big difference is that, instead of a recommender engine, there is a wrapper component for the two recommendation engines (figure 1) that receives recommendation requests and call the relevant methods of the respective engines. The two result sets are then presented according to the policy of the presentation assembler component. The interface of the wrapper is the same as that of the recommender engine, so there is no need to modify part of the already deployed system.

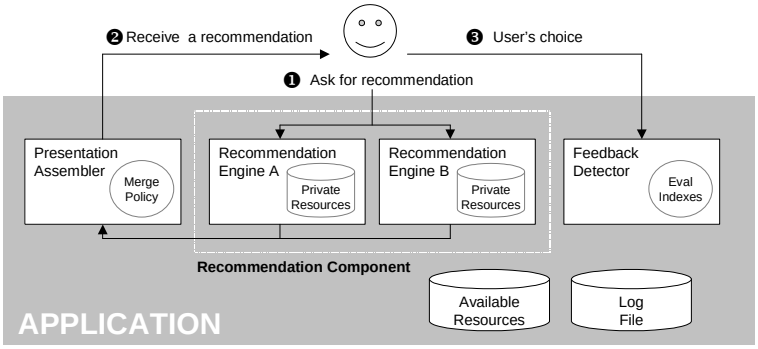

Fig. 1. The impact on a recommender system architecture.

In order to manage a specific competition between two alternative solutions the following elements have to be clearly specified:

- **Available resources**: an API defining what resources competing algorithms can access in order to provide recommendations;
- **Recommender methods**: an API defining the methods a competing recommender must implement;
- **Presentation policy**: defines how the recommendations provided by the two algorithms will be presented to the user;
- **Evaluative feedback**: defines how user actions will be considered evidence of preference of one algorithm over the other;
- **Comparing metric**: defines how to analyse the evaluative feedback in order to determine a winner.

Let us now briefly describe more in detail the elements described above.

Available resources: In making a recommender competition, the most important thing is that both algorithms run under the same conditions and have access to the same basic resources. However, different strategies may rely on different data. For example, collaborative filtering algorithms rely on user-rating data while content-based approaches rely on resources with good semantic description. There are two possibilities in specifying the available resources: we can define an API with a standard language or we can declare in advance a

source of knowledge (like IMDB *(http://www.imdb.com)* for movies). The language to specify the API is free and there is a trade-off between two different needs: minimising the impact on a deployed architecture using the specific language in which it was developed (for example Java) or minimising the constraints for the solution provider configuring a recommendation engine as a stand alone component using protocols like CORBA or SOAP.

The *Recommender methods* element specifies the API defining the methods (*boot* and *recommend*) a recommender must implement (i.e.: its interface) in order to let the wrapper invoke them. The *boot* has usually no parameters and serves to construct the internal model for the algorithm.

The inputs and outputs of the *recommend* method really depend on the specific kind of recommender system: for example, in a book seller's site, the input can be the userId and the output can be an unsorted set of books. In a CD burner application (such as CoCoA [1] [1]) the input can be a partial compilation of songs and the output a set of past compilations made by users. In a personalised radio system (SmartRadio [6], RadioCoCoA [2]) the input can be the userId (with his listening history) and the past 10 listened songs and the output an ordered list of recommended songs.

Presentation Policy defines the way the output of the *recommend* method of the two competing strategies is presented to the user. An open issue related to this is whether the user will know that he is being presented with the combined output of two different recommender engines. We consider three possible presentation policies:

- Merged Set: this is a synthesis of one single result set. It can be achieved by simply interleaving items from each result set. Since the item presented first in a result set is considered to have priority, access to this position is alternated between each recommender engine. Duplicated items are presented once by the presentation assembler.
- Contrasting Set: presentation of two clearly separated result sets. In this case, the two result sets are kept clearly separated and the user knows that they come from two different algorithms.
- Cascading Set: alternate presentation of one result set after the other. On a request for recommendations, a result set from one recommender is presented. On the next request, a result set from the other recommender is presented.

Of course, depending on the domain and the specific application, a particular policy may be not appropriate. For example, the merged set is not a good solution where one recommendation strategy emphasises diversity presenting an heterogeneous selection of items to the user. Merging the results would destroy this effect, and any advantage this strategy would hope to gain.

The contrasting set demands more effort from the user in that he needs to be aware that he is receiving two result sets from two different strategies. However,

[1] http://cocoa.itc.it

it is suited for domains where the output is highly structured or has dependencies between items. An example would be a holiday package recommender which contains accommodation and itinerary information, or a web radio recommender where mixing two different proposed radio programmes would violate the integrity of each.

The cascading set policy presents some other issues such as whether the user should know that he is receiving recommendations from two different recommender algorithms alternatively. The user may perceive the differences in alternate rounds of recommendations and, if he is not aware of the system logic, may begin to mistrust the stability of the overall system.

Evaluation feedback is an important issue. This framework depends on being able to derive, from user actions, a preference for one algorithm over another. The simplest situation occurs where the user clicks on preferred recommendations and it is therefore possible to infer which recommender strategy is better. However, as we discussed in section 3, the user is often not able to express an opinion on new items because he does not know them. A partial solution might be to provide a preview of the items in the recommendations, something Swearingen and Sinha found was appealing to users of recommendation systems. Ultimately the user is only able to evaluate the item after he has experienced it (listened to a song, read an article, watched a movie ...).

There are other means of inferring feedback. For example, measuring time reading a web page or listening to a song. Adding an item to a shopping cart could certainly be considered positive feedback. It should be recognised however that inferring preferences in this way will always be noisier than receiving explicit information from the user. In every case, we extract from the user's actions a judgement of relative user satisfaction with respect to the two strategies.

Comparison metric defines how to combine user's evaluative feedback in order to say "and the winner is ...".

The simplest way is just to count the number of rounds won by the competing systems. However, certain algorithms, such as collaborative filtering, may only start to perform well after sufficient data has been collected. Therefore, we need to analyse the performance curve of each system rather than a cumulative score.

5 Discussion

It is clear that we will need to deploy a real application in order to have a deeper evaluation of the proposed framework. In the meanwhile we introduce some critical factors that only an empirical analysis will make clearer.

First of all, the main advantage introduced by pairwise comparision is that the evaluation does not suffer due to changes in the user community or due to changing conditions (such as the number of items to recommend, the number of new items, ...). At the same time we aren't required to explicitly detect which factors potentially affect the notion of context. This is especially true when the temporal dimension plays a key role in the recommendation process. While

date-time information is recorded in datasets like Eachmovie our earlier example demonstrated that it is not straightforward to make use of this information in an off-line evaluation. This is not a question of data storage but of a lack of evaluation methodology.

One of the drawbacks of our framework is that we need a fully realised system with users. Whether we employ real users or volunteers depends on the state of deployment of our application. If we have an already running system we can use our registered users. If we have yet to deploy a system we have no choice but to use volunteer users. Both approaches have problems. With a 'real' system we have to be careful not to alienate our users by significantly lowering the level of service they already enjoy. A badly performing algorithm may impact upon the trust of users in the overall system. One approach might be to employ a form of reinforcement learning such as the 'n-armed bandit' technique to privilege the recommendations of the better performing algorithm. However, this approach assumes that both 'bandit' algorithms have constant performance over time. This is not the case for algorithms such as collaborative filtering which only begin to perform well after sufficient data is collected. This bootstrap problem is obviously pertinent where we are initialising a 'beta' system using volunteer users. At the beginning there will be very little preference data in the system, and many recommendation algorithms will perform badly as a result. We should also acknowledge that there may be a difference in mentality between either type of user. The 'real' user may be less tolerant of recommender error. The ultimate test is whether the real user stays with the service being provided. Volunteer users are likely to be more conscious of their roles as testers and thus use the system during the trial period.

One aspect of the framework which has been neglected concerns the recommendation process. A tacit assumption underlying our proposal is that the recommendation step can be accomplished by a one-shot operation. However, the process of receiving recommendations may have also have a seperate requirements elicitation process. When evaluating two competing strategies we need to ensure that both use similar types of interaction models. i.e. 'One shot' recommenders need to be compared with 'one shot' recommenders, and conversational recommenders compared with conversational recommenders (sharing the same API). In the latter case, systems with shorter dialogue are judged to be better in standard off-line evaluation. However, it is entirely possible that a strategy with a longer dialogue achieves greater user satisfaction in an on-line evaluation. Perhaps its success may be due to ordering the questions more intuitively, or giving brief explanations at each step.

Finally, we introduce the possibility of extending our framework so as to provide an API for the integration of third party recommender systems. While we easily envisage the deployment and evaluation of two competing algorithms developed 'in-house', the integration and testing of third party software raises some difficult technological issues.

6 Conclusions and future directions

In this paper we argued that generalisation error is very often not a representative measure of user satisfaction with a recommender system. We propose an on-line methodology of pairwise comparison to complement the off-line analysis. We have defined an evaluation framework taking into account advantages and possible drawbacks. We have drawn attention to the relationship with existing approaches in order to highlight the motivations underlying this work.

Our next step is to test the proposed framework and to show with an example how the method works in practice. The short term goal is to collect some empirical evidence that enables a direct comparison between an off-line and an on-line evaluation. A further goal is concerned with the definition and management of a call for competition for prototype recommendation systems, such as SmartRadio or CoCoA. The objective is to assess how sustainable the proposed framework is at a technological level.

References

1. S. Aguzzoli, P. Avesani, and P. Massa. Compositional recommender systems using case-based reasoning approach. In *ACM SIGIR 2001 Workshop on Recommender Systems*, New Orleans, Lousiana, 2001.
2. P. Avesani, P. Massa, M. Nori, and A. Susi. Collaborative radio community. In *Proceedings of Adaptive Hypermedia*, Malaga, Spain, 2002. Springer Verlag.
3. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.
4. C. Basu and H. Hirsh. Learning user models for recommendation. In *UM' 99 Workshop on Machine Learning for User Modelling*, 1999.
5. J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, Madison, WI, July 1998. Morgan Kaufmann.
6. C. Hayes and P. Cunningham. Smart radio: Building music radio on the fly. In *Expert Systems 2000, Cambridge, UK*, 2000.
7. C. Hayes, P. Cunningham, P. Clerkin, and M. Grimaldi. Programme driven radio. In *Proceedings of the 15th European Conference on Artificial Intelligence*, Lyons, France, April 2002. IOS Press.
8. W. Hill, L. Stead, M. Rosenstein, and G. Furnas. Recommending and evaluating choices in a virtual community of use. In *CHI' 95*, 1995.
9. J. A. Konstan and J. Riedl. Research resources for recommender systems. In *CHI' 99 Workshop Interacting with Recommender Systems*, 1999.
10. B. Smyth and P. McClave. Similarity vs. diversity. In *Proceedings of the 4th International Conference on Case-Based Reasoning*, Vancouver, Canada, 2001. Springer Verlag.
11. K. Swearingen and R. Sinha. Beyond algorithms: An HCI perspective on recommender systems. In *ACM SIGIR 2001 Workshop on Recommender Systems*, New Orleans, Lousiana, 2001.
12. Stephan Weibelzahl. Framework for the evaluation of adaptive cbr systems. In *Proceedings of the 9th German Workshop on Case-Based Reasoning*, pages 254–263, Baden-Baden, Germany, 2001.