Evaluating the Intrusion cost of recommending in Recommender Systems

Felix Hernandez-del-Olmo, Elena Gaudioso, Jesus G. Boticario

Dpto. de Inteligencia Artificial
Universidad Nacional de Educacion a Distancia
C/ Juan del Rosal 16, 28040 Madrid, Spain
e-mail: {felixh,elena,jgb}@dia.uned.es

Abstract. Recommender systems suggest items, guiding the user in a personalized way in a large space of possible options. To accomplish this task, they should try to bother users as less as possible, but each recommendation occupies expensive room in the always small user interface. Unfortunately, current evaluation of recommender systems do not have into account this cost. This work presents some new measures that have into account this intrusion cost while recommending. Some experiments are performed to compare our approach with traditional ones.

1 Introduction

Adaptive recommender systems produce individualized recommendations of items [3]. In fact, the usual way to see this problem is that given a large set of items, a recommender must present a personalized *set of recommended* items to the user.

However, we could reformulate the problem as follows: given a large set of items, a recommender must choose which of them must not be recommended. In fact, every single recommendation presented to the user means a cost for her, because the recommendation must be read, or at least it occupies space that could be occupied by other information, perhaps more interesting.

Unfortunately, current measures for evaluating recommenders do not have into account the intrusion cost of recommending. In this paper we show some new measures which try to overcome this problem.

In addition, in order to be unobtrusive and more grounded while evaluating, we should assess each recommendation without the help of prompts to users nor assumptions about their behavior. In that case, we should consider only *really followed recommendations*. We will see in the next sections that if we have this in mind, traditional measures can hardly be applied.

This paper starts recalling traditional measures to evaluate recommenders. Afterwards, we will present our approach by introducing some new measures. Finally, we will show some experiments by running several recommenders with user models based on machine learning. There, we will see that more accuracy does not always mean "good intrusion cost". In fact, we will see in the final discussion that they are usually conflicting properties.

2 Evaluation in Recommender Systems

This section is devoted to recall traditional evaluation of recommender systems and the problems we claim are related to them. However, lets first highlight some basic considerations.

Firstly, notice that, while recommending, a set of *recommended items* are triggered after choosing another one, which becomes the *currently visited* item, also called *active* item. This *set of recommended items* forms the so called *recommendation*. From now on, we will refer to this last set as "set of recommended items", "recommendation set" or just "recommendation".

In particular, instead of *sets*, there exist a class of recommender systems which consider *lists* of recommended items (i.e. [4]). In these systems, the user *presumably* investigate items in that ordered list starting at the top hoping to find interesting items. As said in section 1, we would prefer to take *presumptions* and *assumptions* out of the evaluation. Accordingly, we consider that all recommendations are always formed by *sets of recommended items*, where each one of those recommended items can be observed (or not) in an undefined order.

Secondly, no many authors get involved deeply in the problem of the size of the recommendation set. Nevertheless, we consider this point a very important feature of every recommendation. For instance, in an extreme case imagine that every recommendation set were formed by all the items, then every recommendation would be useless at all. In fact, a recommendation can be called like that, because it highlights or distinguishes a *few* items from a large number of them. Moreover, it connects to our topic, because if we consider the intrusion cost of each recommended item, we are treating this problem as well.

As a first step, in order to impose a limit on the size of every *recommendation set* and to make easier the work of the recommender, most applications apply some kind of "filter". We will call that filter *recommendation window*. In fact, most authors refer to the concept "available items" as a somewhat fuzzy definition of what we call here recommendation window. For instance, in the collaborative filtering context (see for instance [7]), the "available" items are those which have been previously rated by other users. The last allows to choose between an smaller set of items to be recommended (only the available ones). However, in this paper, we want to confront the recommendation problem from a more general point of view. Therefore, we will not refer to those items as "available" anymore, but as items localized into the *recommendation window*.

To conclude with these basic issues, we may think in the recommendation process as formed by two steps: (i) collecting the items to fit the recommendation window, (ii) deciding which of those (previously collected) items must be part of the *recommendation set*. In fact, the more items into the *recommendation set*, the more intrusion the recommender is provoking to the user. In this paper, we will treat only the latter part: deciding which of those items which belong to the *recommendation window* should form part of the offered *recommendation* in order to keep the recommender *"intrusively efficient"*.

Therefore, we focus on the evaluation of the recommenders activity over the items which fall into the *recommendation window* (available items). As said in section 1, we will consider that a single recommended item is correct or *useful* if, and only if, it is *followed*. Reader must notice that not showed recommendations cannot be followed, which has consequences in the evaluation process (see below). We will see in the next sections how we have confronted this problem in our experiment.
From now on, in order to clarify the discussion which follows and without lost of generality, we will consider alongside this paper that recommendations are composed by a single recommended item (recommendation window size=1). In that way, we will be able to focus on the intrusion provoked by each recommendation without distracting with the (also hard) problem of its size.

	useful (u)	useless ($\neg u$)
recommend (r)	a	b
no recommend ($\neg r$)	c	d

Table 1. Confusion matrix of two classes when considering all recommended items of all recommendations. Diagonal numbers a and d count the *correct* predictions: recommend the item when it is to be followed (useful), do not recommend the item when it is to be not followed (useless). The rest of the numbers b and c count the *incorrect* predictions.

To compare the performance of recommender algorithms, usual measures have into account the number of single recommendations that became useful or useless. Formally, they use the so called *confusion matrix* depicted in table 1. Respectively, we define N as the total number of recommendation possibilities to be offered in a whole session, R as the number of recommendations offered and $\neg R$ the number of times recommender did not recommend, this is:

$$R = a + b; \quad \neg R = c + d; \quad N = R + \neg R = a + b + c + d$$

Now, focusing on the evaluation measures, *information retrieval* is likely the parent of recommender systems field. Therefore, it is not strange evaluating recommenders by measures that come from that field. Most usual of them are [6]:

$$Precision = \frac{a}{a+b}; \quad Recall = \frac{a}{a+c}$$

However, high precision usually means low recall and vice versa: Precision and recall are usually conflicting measures, and that is the reason for another usual measure:

$$F - Measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Examples of recommenders evaluated by those measures appear in [2].

Due to having more and more recommenders in which part of their user models are built by machine learning approaches [8], some measures from that field have appeared:

$$Accuracy = \frac{a+d}{N}; \quad MAE(Mean\ Absolute\ Error) = \frac{b+c}{N}$$

Examples of recommenders evaluated by Accuracy are Syskill&Webert [**?**], or WebWatcher [1]. Recommenders evaluated by MAE are usually collaborative filtering approaches [7].

A big problem of the above traditional measures is that they do not account for the fact that the cost of bad recommendations (not followed) should be considered greater than not recommending at all.

In fact, on the first hand, *Accuracy* (and thus MAE) considers b and c similar quantities. Thus, *Accuracy* increases by the same quantity whenever b or c decreases. However, turning down b is better in terms of intrusion, because though being both of them errors, it is a more intrusive error recommending a bad recommendation, than the error of not recommending a good one. We will see this point in more detail in the experiment shown in the next sections. However, as an advance, lets introduce an example. Imagine a system where 80% of recommendations are to be followed, then, recommending all the time is a very accurate model. Could we say the same about its *intrusion*?. Would it be better a less accurate, but less intrusive, recommender?.

On the other hand, it could be considered that *Recall* (and therefore $F-Measure$) tries to go over the intrusion problem in some way, because it measures the capacity of recommending when recommendations are to be followed (useful=a+c). However, that measure does not count the number of useless recommendations offered to do it. Thus, the mixture between *Recall* and *Precision* which leads us to $F-Measure$, could be a solution. However, $F-Measure$ does not have into account the important number d (do not recommend when useless). In fact, if the relation between a, b, and c are the same in two recommenders, a big d should lead us to consider a recommender *less intrusive than the other*, because *useless* recommendations are not showed.

In addition, reader must notice that, as already mentioned above, splitting up the number $\neg R$ into the numbers c and d is something, when possible, at least really difficult. How can we measure if a recommendation will be followed or not if it has never been shown?. In fact, we find this problem while measuring *Accuracy*, $F-Measure$, or *Recall*, because they need the quantities c and d separately in order to be calculated.

These and other problems while evaluating intrusion of recommendations by traditional measures will be discussed along this paper. To this end, lets introduce in the next section some new measures that consider good recommendations much better than not recommending, but also consider a bad recommendation a little worse than not recommending at all.

3 Measures to account for both the efficiency and the obtrusive cost of recommendations

In the last section we went over the current measures to evaluate recommenders. However, we claimed that none of those measures had into account the effort users made (intrusion) by *interacting* with a previous untrained *interactive recommender*. We treat this problem throughout this section.

To this end, lets first start defining three quantities: r_+, r_-, and r_0. The first quantity (r_+) accounts for the positive fact that a previously recommended item is followed. The second quantity (r_-) accounts for the bothersome that each not followed recommended item has provoked. The third quantity (r_0) accounts for the cost of not recommending. Reader must notice that a trivial property these three quantities must pose is the next: $r_+ \geq r_0 \geq r_-$. In fact, the recommender system have to try recommending in order to get the optimum r_+. However, in the process, the recommender must notice that each bad recommendation is always a little worse (r_-) than not recommending at all (r_0).

Now, formally, we define the next measure:

Definition 1.

$$RG(Recommendation\ Gain) = r_+ \times a + r_- \times b + r_0 \times \neg R \qquad (1)$$

Unfortunately, recommendation gain or RG has the problem that only recommenders which have tried to offer the same number of recommendations (same N, see previous section) can be compared. So as to avoid the last problem, we introduce the next:

Definition 2.

$$ARG(Averaged\ RG) = \frac{r_+ \times a + r_- \times b + r_0 \times \neg R}{N} \qquad (2)$$

Now, in order to get a closer definition, we have assigned the next three values to the last three quantities: $(r_+, r_0, r_-) = (10, 0, -1)$. We have chosen those values because we believe they are near the general objectives of most recommenders. In fact, intuitively we can see that not recommending is the same as doing nothing and, because of that, it has no cost (= zero cost). However, acting has a cost: it means either a little worst than doing nothing (-1), if a bad recommendation is offered; or an order of magnitude better (10) than recommending bad, when a good recommendation is offered. Another way to see previous values is as follows: if we keep $ARG > 0$, it means we can stand a recommender that offers (on average) as much as ten bad single recommendations before offering a good single recommendation, *otherwise* $ARG \leq 0$: it would be better turn off the recommender.

Finally, if we go on considering $r_0 = 0$ as above, ARG range goes from r_- to r_+. But, we could prefer a measure whose range were mostly independent on those quantities and varied mostly from 0 to 1. Therefore, we define what follows:

Definition 3.

$$NARG(Normalized\ ARG) = \frac{r_+ \times a + r_- \times b}{N \times r_+} \tag{3}$$

Next sections are devoted to show how this measure works in a real application. The experiment will show further on that a good performance in traditional measures does not assure good performance in NARG, sometimes they even possess really conflicting properties.

4 The experiment

In order to implement and evaluate our approach, we have chosen a web-based learning environment (based on dotLRN[1]), where a datamining course was built. In fact, we will consider the datamining course plus the recommender as the whole application of our experiment. Lets start describing the details of the course.

For the purposes of the experiment, we left students on themselves. But, firstly, users (students) had to fill a form to be registered into the course. As a result, users who navigate into the course have always an (static) user model filled by themselves. For instance, some of the features of that user model (fields of the form) are: *learning style* (inductive, deductive), *machine learning skills* (1-5), etc.

The interface of the whole course is built by means of four frames: header, index, content, and recommendation. By interacting with the *header* frame, users can rate the current content or leave the course. *Index* frame may be used for jumping to whichever *single* content of the course whenever a student wishes to. Into the index frame, themes are used to organize semantically all single content items, but we have not numbered nor ordered them so as to leave free will when choosing them. *Content* frame, as its name indicates, renders each single content item into its (always scarce) limits. Finally, the rest of the section is devoted to explain the *recommendation frame*.

Leaded by the objectives of the experiment, our recommender is only focused on recommending the best next content item to be visited. Because of the objectives of this paper, we impose a *recommendation window* of size 1. Thus, only one item may be suggested per visited content item. In fact, as said in section 2, in this paper we do not treat the problem of choosing the best *recommended items* to put into the *recommendation window*, but the second problem of deciding which of them are to be shown.

Therefore, the only possible recommended item (the one which is collected into the *recommendation window*) is inferred by predefined rules and the user model previously filled by users (see above). This recommendation consists of a link pointing to the next *suggested* single content item. Afterwards, recommendation frame *always* shows the only one possibility of recommendation, which is followed when clicking into it. In fact, because recommendation window size is 1, recommender only has to

[1] http://www.dotlrn.org

decide between two possibilities: "recommend" or "do not recommend" the next content item. As a result, in this experiment we look for a user model attribute to learn when recommender must recommend, we will call this feature "recommend?".
In order to show clearly the idea of intrusion while recommending, we have implemented a recommender interface in which users can *turn off* or *turn on* it by "minimizing" or "maximizing" the recommendation frame. This occurs because minimizing the frame hides the recommendation at all. Reader must notice here that the act of recommending is *not given for free*, because showing the recommendation needs some space that could be used for another, perhaps more useful, information. For instance, in our experiment, recommender interface always takes space from index and content frames. In fact, we had a not little number of students (14/150) that preferred turn off the recommender, perhaps to have more room to read the contents of the course.
The end of our experiment consisted of evaluating the recommender in terms of intrusion, therefore we made the recommender liable of the state of the recommendation frame in the cases the user chose to have the recommendation frame maximized. In fact, maximizing the recommendation frame has a cost for the user as said above. However, "maximizing" for some *followed recommendation* is worth enough to consider worthless having the recommendation frame minimized for the whole session. Lets clarify this issue in the next section, which is devoted to show how we have faced the machine learning task associated to the user model attribute "recommend?".

5 Applying Machine Learning to the task of guessing when to recommend

The objectives of the experiment consist of comparing the measures exposed previously (sections 2 and 3). Thus, we will examine how those different measures behave when several configurations of recommenders interact upon the *same* previously collected data. To this end, we built different recommenders by varying the machine learning algorithm used for building the user model feature "recommend?". Lets start describing how we constructed the training set.
In order to build the training data, *firstly*, because our recommendation window is of size 1, only one single recommended item is to be offered per current content item. As a result, we have as many recommended items as users visits to content items, considering a different training example each one of those possible visits. Finally, the label of each training instance can be either "+" ("recommend") or "-" ("do not recommend").
Secondly, the set of user model features describing each training example would have to do with the *personal* data that each user has previously filled out (see section 4) and the *usage* data which describe the *content item* the user is currently visiting.
Thirdly, in order to finish the preparation of each example, it must be labeled upon user's interaction on the recommender frame. Therefore, if the user clicks on the recommendation, the training example is labeled

as "+", labeling it as "-" otherwise. As reader may see, these data are clearly *interactive* data collected upon recommendation, and the more interactions, the more training instances. Moreover, right after each user's single visit to a content item, recommender can make use of the new training instance to improve its user model. Thus, we retrain/update the user model attribute "recommend?" after each recommendation. Unfortunately, here we must face the fact that recommender *only* gets a new training instance when it *recommends* (= maximizes the recommendation frame).

What is mentioned above increases the difficulty of the experiment. In fact, we could need a different experiment per configuration of the recommender and set of human beings. In order to solve that problem, we agreed to obtain the data by keeping the recommender always showing its recommendation while interacting with users. We called that phase "collecting previous phase". Therefore, during that previous phase, the recommender could not minimize the recommendation frame, only users could.

Once the *collecting* phase (of several weeks) got finished, we run the machine learning experiments. While running each machine learning experiment, *only* instances the recommender decided to offer (and only when the user decided to "maximize" the frame) were considered as (positive or negative) training instances for updating the user model (the feature "recommend?").

Now, leaving apart the data and focusing on the recommenders, the machine learning algorithms used for testing the approach are three well known and used in the field of user modeling and machine learning [5]: Naive Bayes, Nearest Neighbor, and C4.5. Also, we consider two extreme recommenders: "never recommend" and "always recommend".

In figure 1, we show graphs which describe $Accuracy$, $F-Measure$, and $NARG$ in every moment (visit) of the experiment. It must be noticed, on the first hand, that the most accurate recommenders are at the same time the less efficients in F-Measure and NARG. This fact will be discussed in the next section. On the other hand, it must be noticed the instability of F-Measure along the whole experiment. It can be better appreciated in the last graph where each mean derivate per measure is shown. However, the opposite happens in NARG or Accuracy measures, which after the visit 1000 are kept almost constant for most curves (derivate near 0). Lets conclude and discuss all these results in the next section.

6 Discussion

In this paper, we have introduced some new quantities: RG, ARG and NARG, which have into account the intrusion provoked by the recommender's interactions. Moreover, when we evaluate a system by those measures, (while $r_0 = 0$) we are able to say when a recommender is efficient enough. In fact, a recommender is *intrusively efficient* enough only if, at least, it achieves RG, ARG or NARG quantities greater than zero. Otherwise, we can say that it would be better not having the recommender system turned on. In addition, comparing with traditional measures, NARG presents some more advantages.

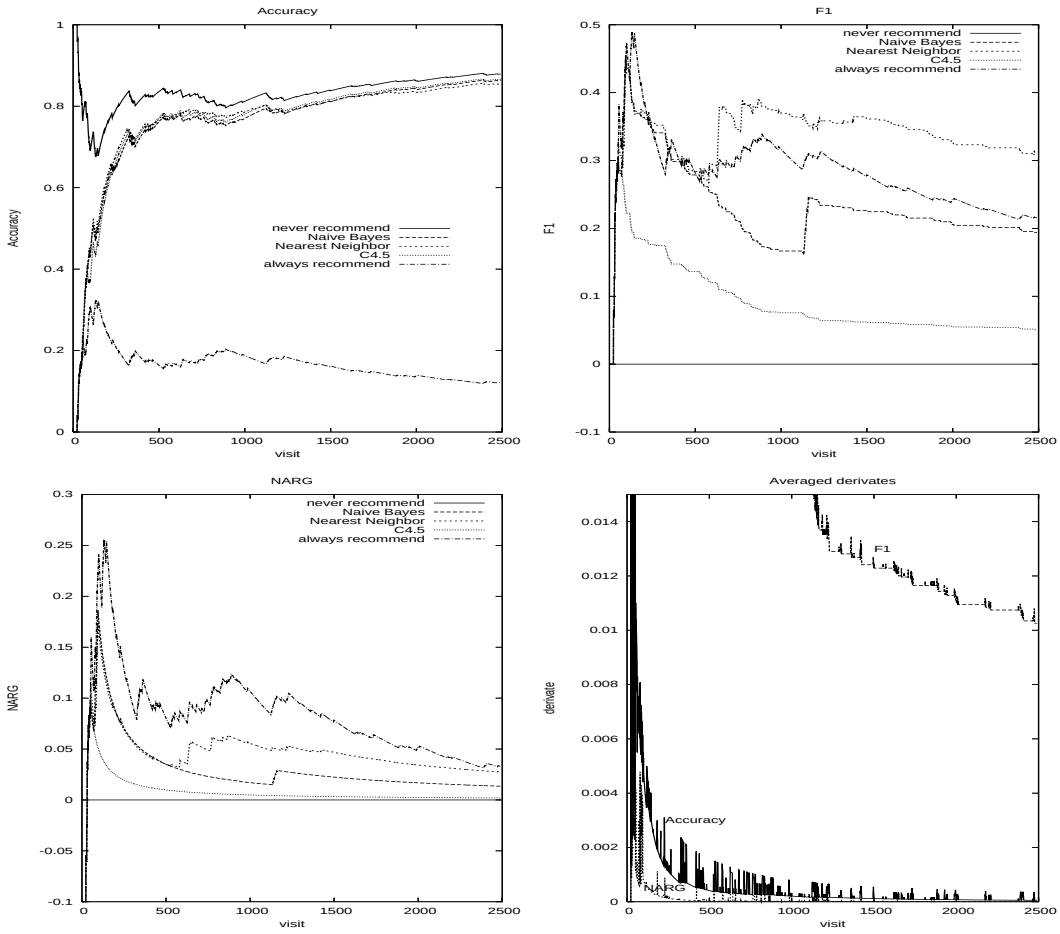

Fig. 1. Left top graph represents Accuracy, right top graph represents F-Measure, and left bottom graph represents NARG. The last (right bottom graph) represents the average of the derivates of the last curves, a different average per measure. Every graph represents its quantity measured after any user's visit. In fact, the x axis represents the number of visits produced along the experiment.

Firstly, as said in section 2, opposite to the most used traditional measures Accuracy and F-Measure, NARG considers the asymmetry of the quantities b and c of table 1. In fact, as first said in section 1 and as mentioned alongside this paper, NARG considers a little better reducing the quantity of bad recommendations (b), than reducing the quantity of good recommendations not offered (c).

Secondly, opposite to *Precision*, *Recall* and F-Measure, NARG and Accuracy have into account the quantity d of table 1. Looking at figure 1,

it must be noticed that Accuracy and NARG are more stable measures than F-Measure when considering the curves of the five recommenders along the whole session. In fact, F-measure behavior may be considered chaotic with rapid changes. We claim that this behavior is due to the fact that d quantity is not present into F-Measure, and every minimum change in a, b, or c provokes a noticeable convulsion into its curves, and thus that high derivate on average.
Thirdly, even considering the d quantity, Accuracy still has a problem: in addition to b and c asymmetry, it does not have into account a and d asymmetry either. In fact, having a large d will keep accuracy high as when having a large a. However, can we say the same about the utility of the recommender?. An example of the latter can be seen in figure 1: even the fact of never recommending is always a very accurate response whenever not many users follow recommendations, as it happens in our experiment. However, in that case a useless recommender (the recommender "never recommend") has the best accuracy (see figure 1). We claim that NARG measure has into account also this fact, as can be seen in the graphs.
Finally, reader must notice that obtaining c and d quantities separately instead of their sum ($c+d$) is hardly possible most of the times. Moreover, we claim it is hardly possible in real working systems. This is due to the fact that once a recommendation is not offered, it is not possible to guess if it would have or would have not been followed by the user (see section 1).

References

1. Robert Armstrong, Dayne Freitag, Thorsten Joachims, and Tom Mitchell. Webwatcher: A learning apprentice for the world wide web. In *AAAI Spring Symposium on Information Gathering*, pages 6–12, 1995.
2. Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proc. 15th International Conf. on Machine Learning*, pages 46–54. Morgan Kaufmann, San Francisco, CA, 1998.
3. Robin Burke. Hybrid recommender systems. *User Modeling and User-Adapted Interaction*, 2002.
4. L. Terveen et al. Phoaks: A system for sharing recommendations. *Communications of the ACM*, 40(3):59–62, 1997.
5. T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
6. G. Salton and M.J. MacGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.
7. U. Shardanand and P. Maes. Social information filtering: Algorithms for automating 'word of mouth'. In *CHI'95: Proceedings of the Conference of Human Factors in Computing Systems*. ACM Press, 1995.
8. G. I. Webb, M. J. Pazzani, and D. Billsus. Machine learning for user modeling. *User Modeling and User-Adapted Interaction*, 11:19–29, 2001.