On Collaborative Filtering Techniques for Live TV and Radio Discovery and Recommendation

Alessandro Basso[2], Marco Milanesio[3,⋆], André Panisson[1], and Giancarlo Ruffo[1]

[1] Dipartimento di Informatica, Università degli Studi di Torino, Turin, Italy
{panisson,ruffo}@di.unito.it
[2] Emporos Systems Corporation, Charlotte, NC, USA
alessandro.basso@gmail.com
[3] Eurecom, Sophia Antipolis, France
marco.milanesio@eurecom.fr

Abstract. In order to integrate properly recording services with other streaming functionalities in a DMR (e.g., AppleTV, PS3) we need a way to put live TV and radio events into friendly catalogs. But recordings are based on parameters to be set by the users, such as timings and channels, and event discovery can be not trivial. Moreover, personalized recommendations strongly depend on the information quality of discovered events.

In this paper, we propose a general collaborative strategy for discovering and recommending live events from recordings with different timings and settings. Then, we present an analysis of collaborative filtering algorithms using data generated by a real digital video and radio recorder.

Keywords: TV and Radio Broadcasts, Recommender Systems, Collaborative Filtering.

1 DMR Context, Motivations and Related Work

Digital Media Receivers (DMRs), such as the AppleTV, or other devices that integrate also DMR functionalities, such as the PS3 and the XBox 360, are rapidly spreading worldwide revolutionizing the way we use our TVs and how we access to streaming content. The major attractiveness of a DMR is in the integration of several functionalities that usually come with different devices: a user can (1) watch (or listen), pause and record live television (or radio); (2) play and store music albums and view related art; (3) view, store, and edit digital pictures; (4) rent or buy new music and movies from catalogs; and so on.

DMR functionalities can be accessed through in-house as well as external services (or *channels*); for instance, the AppleTV allows the user to rent a movie from Apple store and also from Netflix. Moreover, the user can stream a media file stored in another computer connected to the home network, or from other

⋆ Please note that co-authors A. Basso and M. Milanesio contributed to an earlier stage of the work presented in this paper when they were affiliated at the University of Turin as research assistants.

C. Huemer and T. Setzer (Eds.): EC-Web 2011, LNBIP 85, pp. 148–159, 2011.

on-line services like YouTube. However, no matter where the content streams from, the DMR provides an integrated user interface that allows users to browse, search and playback media resources as they were contained in a larger digital library.

This kind of interaction shifts the user's attention from *timing* (e.g., "my favorite TV shows starts at 8:00 p.m.") to *availability* (e.g., "the last movie of my favorite actor is already in my library"). This has implications over recording, because broadcasters schedule timings for their transmissions, and it is up to the user to set parameters accordingly such as the channel, starting and ending times, and so on. It is not surprising that popular applications that offer personalized podcasts, news, TV and radio shows (e.g., Stitcher), usually present lists of available shows to the user, before aggregating media resources together into custom channels. Hence, we need a way to automatically *discover* live TV and radio events and to present them to the user. Probably, this capability is still missing because of the aforementioned timing problem, but also due to the lack of a standard format for structuring the description of events and the unreliability of many Electronic Program Guides (EPGs) (when they are available) [1]. Even if recommendation in the IPTV domain has been studied previously (e.g., [2,3]), there is still room for the discovery of live events to be recorded through custom settings.

After discovery, *recommendation* is a second factor for successful recording services into the DMR context. A recommender system must suggest the user to program the recording of a live event before it occurs. This suggestion must be based on user preferences, and ratings can be used to improve the accuracy of the system.

We are conscious that it is risky to look for general conclusions from a specific case study; for this reason, we decided to remove as many biases as possible. We did not use EPGs and descriptions on timings and content distributed by broadcasters. Moreover, we did not use explicit user ratings. This comes with the observation that feedbacks are not always available, due to user data management strategies (e.g., privacy can be a concern) and unreliability of ratings; in fact, users do not always use explicity feedbacks correctly, due to lazyness or carelessness [12].

Furthermore, given such lack of descriptive information, we cannot use *Content-Based* (*CB*) systems at this stage of the analysis, whereas *Collaborative Filtering* (*CF*) techniques can be easily executed here. We know that CF performances can be improved in practice with the benefits that come with a CB engine, and so we propose a comparative analysis to identify which CF system (and under which assumptions) over performs the others in a real world scenario.

Section 2 gives a brief introduction of the experimental environment. The event discovery procedure is presented in (Section 3). Then, we describe the analyzed recommendation algorithms (Section 4). Finally, the evaluation of the chosen algorithms is presented in Section 5, before drawing conclusions.

2 Discussion on Data Collection

Our analysis is based on real data generated by the *Faucet PVR* system, integrated in a web-based podcasting service named *VCast* (`http://www.vcast.it/`). Faucet allows users to record their favorite (Italian) TV and Radio programs, and to further download them into their devices (e.g., iPod, PC, notebook). User can set up her own programming and see or download their recordings through a simple web interface.

Faucet's users can record their preferred live events in a very traditional way: they can set a list of parameters like the channel, the periodicity, as well as starting and ending times. They are also asked to assign a name to each of their recordings. After the customized event has been recorded, the user can download and reproduce it.

As we said in the introduction, data coming from a general purpose recording system are not immediately usable to identify events such as the transmissions, but assume the form of unstructured information, which have to be properly processed. Intuitively, let T be the set of transmissions during a day and t_i be a specific transmission broadcasted on channel c_{t_i}, starting at time b_{t_i} and ending at time f_{t_i}. Then, in principle, t_i can be directly used in the recommendation engine, as well as $\forall t \in T$. However this is not the case in the real world: if we look at data collected by monitoring the activity of many users, such transmissions are not trivially identifiable, mainly because users set different timings for the same event. This is due to two reasons: (1) users set timings according to clocks that are not in synch each other: this can produce differences in timings in the order of minutes; (2) Users are interested on different parts of the same TV or radio show: in this case, we can have critical differences in timings.

As a final observation, broadcasting is characterized by the *expiration* of some events: we can suggest the user to record only future broadcasts, and even if some shows are serialized, the recording of the single episode should be programmed in advance. This phenomenon is (partially) due to copyright management, since the content provider are not willing to authorize service providers to store previously recorded event for further distribution. Nevertheless, recording of a broadcast is still allowed, because it is seen as a single user activity. As a consequence, we have to deal (also) with volatile content, and this differs very much with the VoD domain, that has been exhaustively explored in the context of recommendation.

The anonymized dataset that we used for our experiments is publicly available at: `http://secnet.di.unito.it/vcast`.

3 Data Processing and Discovery of Events

Even if the DMR environment is perfect for dealing with catalogs of *discrete* events, we cannot prevent the users from setting timing parameters when they want to record live shows. However, we can provide a discovery method that identifies recordings programmed by other users, and that inserts found events in a dynamic catalog: some events can be added when new recordings are observed;

other events are removed when their timings expire. Once we have detected our set of discrete events, we can run our recommender algorithms to create personalized catalogs.

The first step is the identification of the broadcasted transmissions from the amount of unstructured data resulting from the recording process. This is a multi-step procedure that extracts a set of *discrete elements* as the representatives of the broadcasted *events*. Basically, a discrete element is obtained as the result of the aggregation of several different recordings. A preliminary investigation on the extraction of events from recordings is given in [1].

Let $U = \{u_1, u_2, ..., u_k\}$ be the set of distinct users in the Faucet platform. Each user recorded some programs in the past and scheduled some for the future. To schedule a program, a user must choose a channel c among a list of predefined channels C, a periodicity p among the possible periodicities allowed in the digital recorder (for example, daily, weekly, no-repeat), the start and the end of the recording. Besides, the user is required to annotate his/her recording with a (possibly) meaningful title.

Let $R = \{r_1, r_2, ..., r_m\}$ be the set of the recorded programs. Each recording in R is a tuple $r =< u, c, p, tl, b, f >$ set by a user $u \in U$ who recorded on the channel c with periodicity p a program titled tl starting at time b and finishing at time f. Thus, we can assume that there exists a function mapping every user to her recordings.

The set R is first processed by means of **clustering**; then, **aggregation** and **merging** are carried out in sequence on the output of the clustering. The three phases are described in the following.

Clustering: Due to the lack of information about the content of each recording, they are clustered wrt the channel, the periodicity and the difference between timings. Specifically, $\forall r_i, r_j \in R | c_{r_i} = c_{r_j} \wedge p_{r_i} = p_{r_j}$ we have that

$$r_i \uplus r_j \text{ iff } |b_{r_i} - b_{r_j}| < \delta_b \wedge |f_{r_i} - f_{r_j}| < \delta_f,$$

where $\uplus$ is the clustering operator and δ_b, δ_f determine the maximum clustering distance for the start and end times, respectively. The identified clusters contain recordings equal in the channel and periodicity, and similar in the timing. The recording that minimizes the intra-cluster timing distances is chosen as the centroid of the cluster. Each cluster identifies a new event.

Aggregation: As the system produces new recordings continuously, we perform the clustering once an hour obtaining the set of newly generated events. A further step is then required to aggregate the new events with those previously created. Such an operation is performed by comparing each new event with the existing events wrt channel, periodicity and timings; if the timings are similar, we correct the properties of existing events with the values of the newly created ones. The list of users associated to the event is updated accordingly.

Merging: Similar events, i.e. with the same channel and periodicity but timings within a fixed range, are merged into a single event. All features of the new events are computed by means of the values of the merged ones. This operation is required because events can be created in subsequent moments, by aggregating

recordings referring to the same broadcasted transmissions. Due to the high variability of the timings, especially when a new transmission appears, such events slowly and independently converge to more stable timeframes, determining the need of merging them into single events.

As a result of the whole process, we obtain a number of events, each being a tuple defined as $e =< U_e, c, tl, b, f, p >$ where U_e is the list of users who set a recording referring to event e, c is the channel, tl is a title chosen among those given by users using a majority rule, b and f are the the starting and ending times and p is the periodicity. More detail on event detection and title selection can be found in [1].

We observed the behavior of the system in a one year timeframe, i.e., from June 2008 to June 2009, wrt the number of users, events and recordings. As the number of active recordings and events tends to increase over time, the number of users follows a different, less constant, trend. Specifically, we can notice a considerable increase in the number of registered users in the system between November 2008 (< 35.000 users) and March 2009 (> 45.000). In July 2009 we observed an interesting average number of 20.000 users with at least one scheduled recording. The relative success of the service reflected in the number of recordings: we had about until 200K recordings in June 2008 (the service was launched few months ago), and approximately 900K recordings one year after. Analogously, the number of events generated by the aggregations of the recordings grows up: we could detect almost 32K different events in June 2008. The overall number of detected events was about 130k after one year.

4 Recommendation

Two well-known recommendation techniques are considered in this work: (1) the memory based *collaborative filtering* approach named k-Nearest Neighbors (kNN) [9]; (2) the model based approach based on the *SVD transform* [10].

Exploiting the basic idea of the *nearest neighbors* approach, we apply both variants of the kNN algorithm: the user-based one [5], by identifying users interested in similar contents; and the item-based approach [4], by focusing on items shared by two or more users. The *MostPopular* items can be considered as a special case of the user-based kNN approach, where all users are considered as neighbors. In addition, we also analyze the performance of a variant of the SVD technique based on implicit ratings, presented in [6].

User-based kNN. In the *user-based* kNN algorithm, the weight of an element e for a user u can be defined as:

$$\mathrm{w}(u,e) = \sum_{v \in N(u)} \mathrm{r}(v,e) \cdot \mathrm{c}(u,v), \tag{1}$$

$$\text{where} \quad \mathrm{r}(v,e) = \begin{cases} 1 \text{ if} & e \in E_v \\ 0 \text{ if} & e \notin E_v \end{cases}$$

E_v is the set of elements recorded by user v, whilst $N(u)$ is the neighborhood of user u, limited by considering only the top-N neighbors ordered by user

similarity. $c(u, v)$ is calculated using a similarity metric, $S(u, v)$, and we considered several well known measures, such as: the *Jaccard*'s coefficient, the *Dice*'s coefficient, the *Cosine* similarity and the *Matching* similarity [8]. All similarity metrics are calculated using the implicit binary ratings $r(v, e)$. Then, $\forall u$, we can compute the subset $N_u \subseteq U$ of *neighbors* of user u by sorting all users v by similarity with u. Only the k users most similar to u and with $S(u, v) > 0$ will be present on N_u.

If the number of neighbors is limited by the chosen similarity to a number lower than k, we can also consider the 2nd-level neighbors, i.e., for each user v belonging to $N(u)$ we compute $N(v)$. The overall set of 1st-level and 2nd-level users is then used to define the users similar to u, as previously described. It is worth noting that, in case of considering 2nd-level neighbors, the coefficient $c(u, v)$ in eq. (1) has to be computed taking into account the similarity between the considered neighbor and further ones. For example, considering user u, her neighbor v and her 2nd-level neighbor x, we have:

$$c(u, x) = S(u, v) * S(v, x),$$

that is a combination of the similarities computed between the neighbors pairs for the considered user.

MostPopular. The *MostPopular* algorithm can be also defined by means of eq. (1), assuming the number of neighbors unbounded, which implies $N(u) = U,\ \forall u \in U$; and $c(u, v) = 1,\ \forall u, v \in U$.
The weight of an element e to a user u is therefore defined as:

$$w(u, e) = \sum_{v \in U} r(v, e) \tag{2}$$

All elements are sorted in descendant order by weight. The set of neighbors is independent of the user in the *MostPopular* algorithm. As consequence, all users receive the same recommended elements, i.e., the most popular elements.

Item-based kNN. In the *item-based* kNN algorithm, the weight of an element e for a user u is defined as:

$$w(u, e) = \sum_{f \in N(e)} r(u, f) \cdot c(e, f), \tag{3}$$

$N(e)$ is the set of n items most similar to e and recorder by u, and $c(e, f)$ is the neighbor's weight wrt item e.

Differently from the user-based case, using $k = \infty$ in the item-based approach does not lead to the *Most Popular* set of elements. In fact, the algorithm simply takes all items $f \in E_u$ as neighbors of e, making $N(e)$ user-dependent.

The similarity among items, $S(e, f)$, is based on the same measures already mentioned before, yet redefined considering two items e, f and their sets of users U_e, U_f who recorded them. $\forall e \in E$ we can compute the subset $N_e \subseteq E$ of

neighbors of item e. An item f such that $U_e \cap U_f \neq \emptyset$ is thus defined as a neighbor of e. Starting from the neighborhood of e, similarity with e is computed for each pair $< e, f >$ such that $f \in N_e$ using the implicit binary ratings $\mathrm{r}(u, e)$ as defined in (1), and the weights are calculated according to (3).

SVD. The Singular Value Decomposition technique analyzed in this work makes use of implicit feedbacks and implements the method proposed in [6]. Specifically, given the observations of the behavior of user u wrt item i, r_{ui}, we can define the user's preference p_{ui} as equal to the implicit binary rating r_{ui}. Note that r_{ui} is set to 1 when u records item i, 0 otherwise.

After associating each user u with a user-factors vector $x_u \in \mathbb{R}^f$ and each item i with an item-factors vector $y_i \in \mathbb{R}^f$, we can predict the unobserved value by user u for item i through the inner product: $x_u^T y_i$. Factors are computed by minimizing the following function [6]:

$$\min_{x_* y_*} \sum_{u,i} (p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

5 Experimental Results

Our evaluation is based on measuring the accuracy of each recommendation algorithm in predicting the elements that users would program. This is achieved by computing precision and recall on the predicted items. The more accurate is this prediction, the more valuable elements are recommended. It is important to underline that we do not consider any feedback related to the user's interest in the recommended items, but we only focus on the prediction ability of the algorithms analyzed.

To evaluate a recommendation algorithm, we fix an arbitrary time t in the data collection interval, and use the information about the user recordings before time t to predict the elements recorded by each user after time t. The collected data start at January 2008 and end November 2009, thus we choose uniformly distributed values of t varying from June 2008 to June 2009 in order to not have biased results by scarcity of training data or by lack of test data.

Given the set E of events in our framework, we define the following subsets:

- $A(t) \subset E$, define the active events at time t ($b_e > t$);
- $R(u, t) \subset E$, define the events recorded by user u before time t;
- $V(u, t) \subset A(t)$, define the events recorded by user u after time t;
- $Rec(u, t) \subset A(t)$, define the events recommended to user u at time t.

It is important to notice that $A(t)$ is also the set of all elements suitable for recommendation at time t. The aim of our recommendation algorithms is to predict which events are in $V(u, t)$. For that, for each user, the algorithms associate a weight for each element in $A(t)$ that are not present in $R(u, t)$. To recommend items to users, we use the top n recommended elements $Rec(n, u, t) \subset Rec(u, t)$,

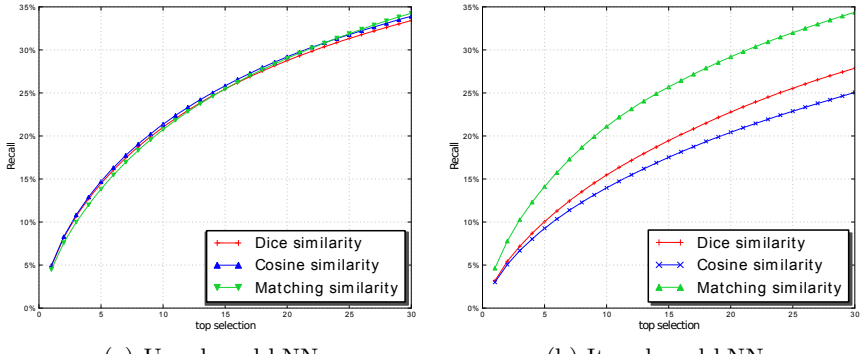
ordered by weight. The *precision* values for the top n recommended elements at time t are computed as the average of $(Rec(n,u,t) \cap V(u,t))/Rec(n,u,t)$ for all users. The same for *recall* values, computed as the average of $(Rec(n,u,t) \cap V(u,t))/V(u,t)$ for all users [10]. Finally, we compute the precision and recall for the top n recommended elements as the average of the precision and recall at different ts.

Our evaluation does not use user's feedbacks regarding his interest in unconsidered items (i.e., not programmed, nor downloaded). Thus in this context, as in [6], recall measures are more suitable than precision measures. In fact, we can assume that e_i is of any interest for user u only if $e_i \in V(u,t)$, otherwise no assumption on user's interests can be made. Anyway, for sake of completeness, we also report the analysis of precision values.

5.1 Evaluation

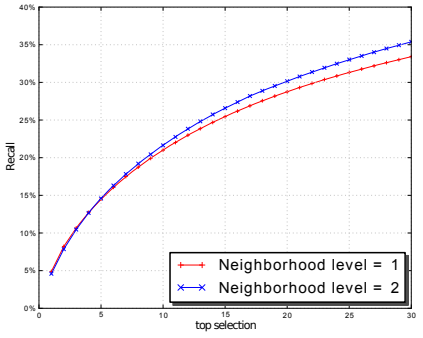
We start our evaluation showing how different similarity functions affect the results of user-based kNN recommendation algorithms. We can observe from Figure 1(a) that, in case of the user-based algorithm, all chosen similarities show nearly the same performances. In all cases, we used a neighborhood of $k = 300$, however the results are similar for other values of k. When it comes to the item-based algorithm, the Matching similarity considerably outperforms the other measures, as displayed in Figure 1(b). Again, both Dice and Jaccard show a very similar behavior, being clearly superior to the Cosine metric when more than 5 elements are recommended. In both Figures 1(a) and 1(b), the Jaccard similarity is not shown being almost identical to the Dice.

In Figure 2(a) we evaluate the consequences of adding second-level neighbors in the neighborhood of user-based kNN recommendation algorithms. We can observe that increasing the number of first level neighbors (when it is lower than k) by adding the second level ones implies a better performance of the algorithms. In this example, we used Dice similarity and $k = 300$, however the results are similar when applying second-level neighbors to other similarities.

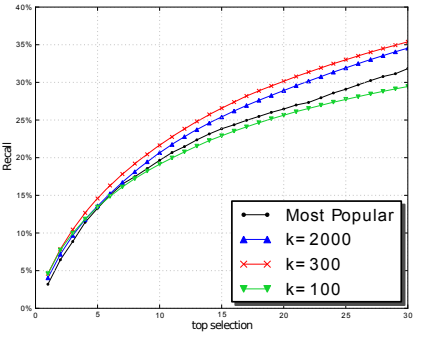


(a) User-based kNN (b) Item-based kNN

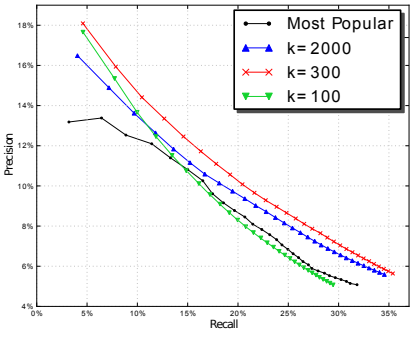
Fig. 1. Comparison between similarity functions in user-based and item-based kNN



(a) Recall values for one-level and two-level neighborhoods for user-based kNN



(b) Recall values for different neighborhood sizes in user-based kNN



(c) Precision vs Recall for different neighborhood sizes in user-based kNN

Fig. 2. Neighborhoods comparison, precision and recall for user-based kNN

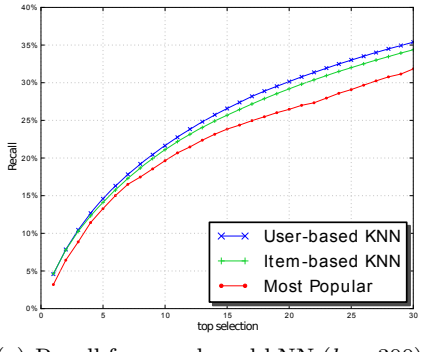
In the next tests, we try to find an optimal value for k in the user-based kNN algorithm. Fig. 2(b) shows the results of kNN user-based for different values of k and the *MostPopular* recommender. We used Dice similarity, but the results are similar with other similarity functions, as previously explained. We can observe that a value $k = 100$ is not sufficient to outperform the *MostPopular* algorithm, due to the lower value of the recall. On the other side, a very high number of neighbors allows to perform better than the *MostPopular*. However, we could notice that for high values of k the algorithm starts to converge to the *MostPopular*, characterized by an unbounded number of neighbors. We found that $k = 300$ is a good compromise between the ability of providing valuable recommendations and the resource consumption in calculating the neighborhood.

To better observe the trend of both recall and precision, Figure 2(c) shows the two values combined. Again, $k = 300$ performs better if we take the top 10 recommended elements, as it also yields to good results in terms of precision. Considering more than 10 recommendations, it would seem appropriate to increase the number of neighbors, as the results for precision and recall

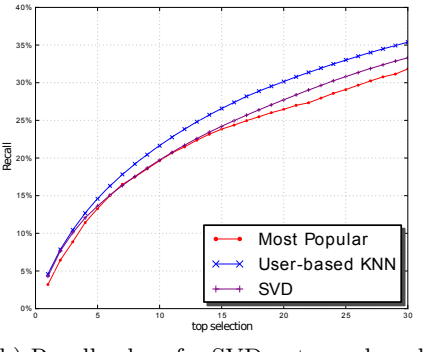
are slightly better. Nevertheless, considering the algorithm performance also in terms of computational requirements, $k = 300$ is still a good choice when we take into account the precision metric.

An interesting comparison among the three kNN algorithms analyzed, i.e., user-based, item-based and *MostPopular*, is depicted in Figure 3(a). We can observe that the latter is clearly outperformed by the other two algorithms in terms of recall, especially when more recommended items are considered. The user-based kNN performs slightly better than the item-based version, although the gap is mostly noticeable when more items are recommended. In general, item-based algorithms tend to perform better because usually the number of items is considerably lower than the users [9]. Such a property does not hold in our domain, hence making the user-based version superior in terms of recall, as we initially expected.

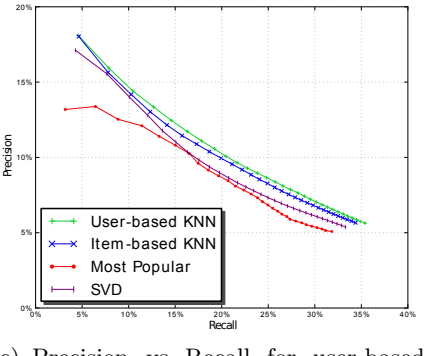
A final experiment was made in order to compare the performance of the SVD approach to the kNN. The implementation of the SVD algorithms described in Section 4 is tested with different parameters, with the purpose of identifying the more suitable ones in our context. In particular, we try different sizes for



(a) Recall for user-based kNN ($k = 300$), item-based kNN and MostPopular



(b) Recall values for SVD wrt user-based kNN ($k = 300$) and MostPopular



(c) Precision vs Recall for user-based kNN ($k = 300$), SVD and MostPopular

Fig. 3. Precision and recall for the analyzed algorithms

user-factors and item-factors vectors, values for the λ parameter and number of training steps. Results are depicted in Figure 3(b). The best prediction is obtained with 100 features, $\lambda = 500$ and 15 training steps. However, the performance of the SVD approach in the analyzed context is worse if compared to a neighborhood model such as kNN. Similarly, results related to the precision (Figure 3(c)) show an analogous performance of the kNN algorithms wrt SVD, with the *Most Popular* being considerably less precise than others.

It could appear surprising that the prediction performance of the SVD recommender is worse than other techniques, as this algorithm normally performs better in several other contexts [10,7,6]. We believe that the motivations for such an unusual behavior reside in the dataset characteristics. In particular, a reason might be identified in the so called *cold start problem*, whose effects involve users, items and communities [11]. In our context, the cold start problem is particularly noticeable with items and is due to the lack of relevant feedbacks when a new event first appears in the system. Such an issue is made worse by the fact that items to recommend are generally new ones, i.e. those events having a starting time in the future. This property holds for no-repeat events as well as for repetitive ones (the starting time is updated according to their periodicity). So, events whose starting time has passed are no longer eligible for recommendation.

The fact that recommendations are affected by the cold start problem is one key factor that may influence SVD performance, as this algorithm needs support of user's preferences to perform well. On the contrary, a neighborhood-based approach such as kNN appears to better deal with newly introduced items, as also reported in [2].

6 Conclusion and Future Work

We proposed a methodology to detect live TV and radio events from a set of independently programmed recordings. Assuming that such events can be browsed, searched and played back as other digital resources as they are included in a large digital library, it emerges the importance of suggesting recordings to user. Thus, we experimented with data of a real digital recording service to compare collaborative filtering techniques. Our findings showed that neighborhood based strategies, such as kNN, can return in good prediction accuracy and, if correctly tuned, they can outperform SVD-based techniques as well as *most popular* strategies, which dangerously leverage the phenomenon of many users concentrated on very few relevant events.

The evaluation of a content-based recommender system in this domain is planned. This was not possibile at this stage of the work because of the difficulty of getting descriptions about recorded events with earlier versions of the analysed DMR system.

Acknowledgements. This work has been partially produced within the "SALC" (Service à la carte) project, supported by Finpiemonte, ("Progetto Polo ICT"). Of course, we are grateful to InRete and Giorgio Bernardi that provided us the access to the *VCast* digital recording system data.

References

1. Basso, A., Milanesio, M., Ruffo, G.: Events discovery for personal video recorders. In: EuroITV 2009: Proceedings of the 7th European Interactive TV Conference, pp. 171–174. ACM, New York (2009)
2. Cremonesi, P., Turrin, R.: Analysis of cold-start recommendations in IPTV systems. In: RecSys 2009: Proc. of the 3rd ACM conf. on Recommender Systems, pp. 233–236. ACM, New York (2009)
3. Cremonesi, P., Turrin, R., Bambini, R.: A Recommender System for an IPTV Service Provider: a Real Large-Scale Production Environment. In: Kantor, P., Ricci, F., Rokach, L., Shapira, B. (eds.) Recommender Systems Handbook, vol. ch.30, pp. 200–220. Springer, Heidelberg (2009)
4. Deshpande, M., Karypis, G.: Item-based top-n recommendation algorithms. ACM Trans. Inf. Syst. 22(1), 143–177 (2004)
5. Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In: SIGIR 1999: Proc. of the 22nd Annual Intern. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 230–237. ACM, New York (1999)
6. Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: ICDM 2008: : Proc. of the 2008 Eighth IEEE Intl. Conf. on Data Mining, pp. 263–272. IEEE Computer Society, Washington, DC (2008)
7. Koren, Y.: Collaborative filtering with temporal dynamics. Commun. CACM 53(4), 89–97 (2010)
8. Markines, B., Cattuto, C., Menczer, F., Benz, D., Hotho, A., Stumme, G.: Evaluating similarity measures for emergent semantics of social tagging. In: WWW 2009: Proc. of the 18th Int. Conf. on World Wide Web, pp. 641–650. ACM, New York (2009)
9. Sarwar, B., Karypis, G., Konstan, J., Reidl, J.: Item-based collaborative filtering recommendation algorithms. In: WWW 2001: Proc. of the 10th Int. Conf. on World Wide Web, pp. 285–295. ACM, New York (2001)
10. Sarwar, B.M., Karypis, G., Konstan, J.A., Riedl, J.T.: Application of dimensionality reduction in recommender system - a case study. In: ACM WebKDD Workshop (2000)
11. Schafer, J.B., Konstan, J., Riedi, J.: Recommender systems in e-commerce. In: EC 1999: Proceedings of the 1st ACM Conference on Electronic Commerce, pp. 158–166. ACM Press, New York (1999)
12. Smyth, B., Wilson, D.: Explicit vs. implicit profiling a case-study in electronic programme guides. In: Proc. of the 18 th In. Joint Conf. on Artificial Intelligence (IJCAI 2003), pp. 9–15 (2003)