Events Discovery for Personal Video Recorders

Alessandro Basso
Università degli Studi di Torino
Dipartimento di Informatica
Torino, Italy
basso@di.unito.it

Marco Milanesio
Università degli Studi di Torino
Dipartimento di Informatica
Torino, Italy
milanesio@di.unito.it

Giancarlo Ruffo
Università degli Studi di Torino
Dipartimento di Informatica
Torino, Italy
ruffo@di.unito.it

ABSTRACT
Recommendation Systems can be one of the killer applications for a new generation of PVRs. Unfortunately, these systems do not apply directly to this domain because of the unreliability of EPGs and registrations set on temporal basis instead of discrete objects. We propose a method for discretizing temporal and volatile events and the preliminary version of a recommender system that can be executed on such data.

Categories and Subject Descriptors
H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—Information filtering; J.4 [Computer Applications]: Social And Behavioral Sciences

General Terms
Algorithms, Experimentation, Human Factors.

1. INTRODUCTION AND MOTIVATION
The convergence between traditional media and the Internet resulted in the available information and digital contents to be outnumbered. Nevertheless, the discrepancy between the available services, through the Web and through an interactive TV, is still wide. Moreover, today’s high bandwidth availability and diffusion made the joint between the Internet and the Television possible as a sign of our times (e.g., IP-TV, VideoOnDemand, YouTube). Such a fact leads to a huge amount of available contents, increased further by the coming of a new generation of Personal Video Recorders (PVRs) [1], unbouding the requirement of being in front of the media device for viewing a specific program.

Just like old video recorders, PVRs record and play TV and Radio programs, setting the user in control of the stream in terms of channel selection and recording time intervals. Most of the available PVRs are part of the subscription with a service provider (e.g., Virgin Sky Media in the United Kingdom or Fastweb in Italy), and they can be dedicated hardware or a proprietary system (users can only register their channels in an armored format).

Most of these tools, however, lack in user’s personalization, as well as the possibility of automatically creating and managing a kind of playlist based on user’s interests, which leads to more passive user’s behavior. Moreover, existing Electronic Program Guides (EPGs) lack in specific information on the content (aside, of course, from the title and the channel), just like newspapers TV guides, which purely consist in an often unreliable list of programs.

Recommender Systems [4] are a great tool for implementing user personalization. In fact, a generic recommender engine is able to assist the user in finding interesting information. Usually, it takes the profile of user’s interests as input, returning a personalized set of suggestions on given objects. Unfortunately, the adoption of such systems is not straightforward in our PVR domain, due to two important reasons: (1) we are not dealing with discrete and permanent objects, but with events characterized by a temporal validity; (2) we cannot rely on existing EPGs, and users independently define what and when to record.

Our approach is based on the analysis of real data generated by the Faucet PVR system (www.vcast.it/faucetpvr), integrated in a web-based podcasting service named VCast (www.vcast.it). Faucet allows users to record favorite TV and Radio (italian) programs, that can be further downloaded into their devices (e.g., iPod, PC, notebook).

The goal of this paper is to present a novel data analysis, events discretization and users aggregation framework, specifically tailored to the PVR-services domain. Such a framework is the corner stone for a further development of a user-centric personal recommendation engine. First of all, we introduce our approach to discover discrete events from chaotic and independent recordings (Section 3). Then, a preliminary version of the recommendation algorithm is given. Finally, some evaluation on the overall process is presented in Section 4, before drawing conclusions and proposing future research questions (Section 5).

2. BACKGROUND
Recommendation engines are usually categorized as content based (CB) or collaborative filtering (CF) systems [6]. CB methods recommend an item basing on its description and a profile of the user’s interests. They are not applicable in our domain, due to the lack of description of the involved items. CF methods, instead, bring together the opinions and/or the behaviors of large interconnected communities on the web, supporting filtering of substantial quantities of
data, and triggering suggestions based only on similarities between users.

We adopted a CF approach using a variation of the User-Based Nearest Neighbor algorithm, and Affinity Networks in order to model user’s similarities. These networks have been found very efficient in other dynamic domains with incomplete or unreliable knowledge (e.g., P2P file sharing networks [5], and in ad-hoc mobile scenarios [7]).

As underlined in [3], which proposes a recommendation framework for building personalized EPGs, one of the main drawbacks of the CF approach is the sparsity problem, i.e., users tend to give few or no ratings; hence the expected item overlap between two random users is usually very low. Our approach is less affected by this problem, as we only focus on implicit ratings derived from the user’s behavior rather than on explicit ones. A further important difference is the lack of a library of item descriptions, used in [3] to query against. We do not rely on the recording content, but only on users’ behavioral similarities.

Authors in [8] show that many CF techniques used to build a personalized EPG return a limited number of recommended items, whilst their approach is to return a prioritized list of all TV programs. However, we believe that the implicit user profiling is enough to return to each user a personalized (i.e., not all) recommendation list of items.

3. DISCOVERY METHOD

Faucet is a distributed system that converts broadcasted audio/video content to a proper (suited for different kinds of portable multimedia devices) podcast format. This approach tries to simulate the normal criteria of a VCR regarding: (a) the choice of the preferred content, (b) the enforcing of the private aspect for each recording by attaching to it rights management information in different formats and (c) the management of several personal podcast channels, in order to guarantee the privacy of each user.

We worked on SQL dumps from the database of the set recordings, updated every hour. Each dump rewrites the previous one, adding the newly added recordings and deleting the expired ones. A tuple in the recording database contains various information. Among the obvious ones (e.g., title, channel, start time), a useful attribute is the periodicity of the recording. A user can set its recording to be daily, weekly, only once, monday to friday and so on. We can therefore identify two main classes of programmings: those which occur only once (no-repeat), and those which are repeated over time (either daily or weekly). First of all we had to define what is being broadcasted, on the basis of users’ usage of the system, in which only the channels are fixed, while all the other attributes are freely inserted by the user. We then defined a discrete event, by means of an aggregation process over computed clusters, and finally we computed the relationships between discrete events and users.

3.1 Discretization

In order to define a discrete event, we have to cluster all the different recordings on the basis of the respective values of the single attributes. One of the main issues in this phase is the total lack of information on the content (e.g., sport, news, action-movie). Nonetheless, we are not interested in this information, because what really discriminates between two programs (on the same channel) is the timing, plus the periodicity of the recording. So, after a detailed analysis on the available datasets we put the focus of the clustering algorithm on the similarity between intervals of two distinct recordings. If two recordings, on the same channel, present similar start time and end time, then we cluster them together.

The clustering algorithm proceeds as follows. It takes as input the set of the programmed recordings, for each different channel and each different periodicity (i.e., all the possible channel-periodicity combinations). The first element of each set is taken as the temporary pivot. Each recording is thus associated to the cluster if its timing falls into a fixed $\epsilon$-neighborhood. At the end, the recording which minimizes the timing differences within the cluster is elected as the final pivot of the cluster. It’s important to underline that the clustering is made only on top of timing, because no content information is available, since we cannot rely on users’ inserted information on titles. Furthermore, neither the size of the cluster, nor the final number of clusters is fixed a-priori.

The aggregation procedure strictly follows the clustering, for mitigating the lack of information on contents. We aggregate the single clusters defining a discrete event. Each element $R_i$ in the cluster is a programmed recording, i.e., a tuple in the dataset, composed by a user $u_i$ who recorded on the channel $c$ a program titled $t_i$ from time $b_i$ to time $e_i$: $R_i = \langle u_i, c, t_i, b_i, e_i \rangle$. It’s important to notice that the channel is set. In fact it is one of the few attributes that can not be described by the end user, but rather is given as a list of predefined channels; thus each element in a single cluster has exactly the same channel attribute. This does not happen for the title, which is completely up to the user without any restriction. After the aggregation, we have a single discrete element $E_d = \langle \{u_i\}, c, \{t_i\}, b, e \rangle$, where $\{u_i\}$ is the list of users in the cluster, $\{t_i\}$ is the set of titles used in the cluster, $b$ and $e$ are the average timing computed from the cluster.

The final creation of a discrete element is obviously constrained by the eventual past insertion of a very similar discrete event. Our algorithm thus checks the database for a discrete element with similar attributes (concerning periodicity and timing). The constraints for timing are very strict (the $\epsilon$-neighborhood is smaller than 5 minutes). If such an element is found in the database, then the users of the new discrete element, along with the titles used, are associated to the already inserted element, and the timing information are gathered together the previous, while in case such an element is not found, the new discrete element can be inserted. In both cases, the most common title among those associated to the element is extracted and is set as the most likely title of the discrete element.

Two main problems were faced in the clustering and aggregation procedure. The first, as said before, is the fact that the end user has an almost complete control on the system, thus she can set any timing and any title she wants. The channel is the only field that is given. This leads to a wide variety of titles and timing for recording the same program, but, while for the timing this was solved with the introduction of various thresholds, for the string representing the title it was necessary to compute the “most popular” title of the cluster. The second problem is the periodicity of the recordings. As long as five options are available in VCast (i.e., daily, weekly, mon-fri, mon-sat, no-repeat), it was necessary to develop five different aggregating functions for managing the different behaviour of the recordings.
3.2 Main Features

As a first product of the aggregation algorithm, the daily hits are computed wrt the periodicity of the associated recordings. For each periodicity, we compute the most programmed discrete events, and we propose them to the users. The main goal of the aggregation procedure is, however, to build a valuable dataset for the setting up of a recommendation engine. Given a user with some interests (i.e., some discrete elements associated to her), we built a recommendation engine based on the collaborative filtering approach. The basic idea on which this goal is layered on is: what programs have been recorded and programmed in the future by users who share some interests?

To define such a similarity notion between users, we introduce an affinity function $Aff(u,v)$, that returns a similarity evaluation for any pair of users $(u,v)$. Affinity is estimated on top of past and future recording activities (i.e., resources in common) and similar recording behaviors. Instead of considering top-K nearest neighbors (that usually is unbiased for users with few or too many similars), a threshold metric is adopted. Hence, our Affinity Network is represented by a graph where nodes are users and there exists an edge between $u$ and $v$ iff $Aff(u,v) > \theta$, where $\theta$ is a given connectivity threshold.

For sake of simplicity, let’s follow an example, having in mind that this preliminary version of the recommendation algorithm is mainly devised with the purpose of validating the discretization and aggregation procedure.

Example. A user $u$ has recorded or scheduled a set of programs. Let $R_u = \{R_{u1}, R_{u2}, ..., R_{un}\}$ be this set. $E_u = \{e_{u1}, e_{u2}, ..., e_{un}\}$ is then the set of discrete elements which user $u$ is associated to (i.e., user $u$ made recordings clustered in the discrete elements in $E_u$). Each discrete element is associated to a user by means of a function $f$ that counts the number of recordings associated to that specific discrete element. Let $F_u$ be the similarities to $u$:

$$F_u = \{v : E_v \cap E_u \neq \emptyset \land Aff(u,v) > \theta\}.$$

Starting from the sets of the similars to $u$, the recommendation engine proposes to user $u$ all the discrete elements in the set:

$$Recom_u = \{v \in F_u, e_v \in E_v \land e_v \notin E_u \land f(v,e_v) > \delta\}$$

The set $Recom_u$ is the set of the discrete elements that are recommended to $u$: it contains all the elements of some strong (i.e., $> \delta$) interest for the similars to $u$ which $u$ has never seen, recorded or programmed for the future. Remember that we do not have any information about the proposed content, but the collaborative filtering approach allows us to group users on the basis of their personal interests, even without this knowledge. Of course this process can be improved with a feedback mechanism that allows users to vote for the suggested programs, specifying if it was satisfactory or not.

In Figure 1, the functionalities of the proposed framework are pictured. The end user (a) inserts into VCast her programming. These recordings (b) are saved in the central database of the podcasting system. The RD-PVR framework, querying the database, computes the discrete elements and associates each user to her own subset of discrete elements (c). The VCast service is then able to send back to the end user the computed information (d).

4. PRELIMINARY ANALYSIS

The explained approach has been implemented and is processing dumps since January 2008. Each dump is composed by nearly 26000 recording entries. Given two successive dumps, we compute only the difference between them. This difference is given by the newly inserted recordings and by the modification of some of the parameters in the previously inserted recordings, that is, a new program or a change in the timing, the title or the periodicity. This difference is very changeable, depending on the particular period. Since almost one year of dumps, we had the situation depicted in Table 1, wrt the new and modified recordings in each dump and the new and modified discrete elements got after the processing.

<table>
<thead>
<tr>
<th>recordings</th>
<th>min</th>
<th>max</th>
<th>new</th>
<th>modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>average</td>
<td>176</td>
<td>1624</td>
<td>27</td>
<td>5582</td>
</tr>
<tr>
<td>discrete</td>
<td>min</td>
<td>0</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>130</td>
<td>1326</td>
<td></td>
</tr>
<tr>
<td></td>
<td>average</td>
<td>20</td>
<td>191</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Values of new or modified recordings and discrete elements.

On a total of 15483 distinct users, at the time of writing (March 2009) we collected the following data from the system: the number of discrete elements a user is associated with, the number of similar users and the number of discrete elements recommended to a single user. The average, minimum and maximum values for each of the above mentioned categories is given in Table 2.

<table>
<thead>
<tr>
<th>elements</th>
<th>similars</th>
<th>recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>max</td>
<td>349</td>
<td>64</td>
</tr>
<tr>
<td>average</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 2: Values for associated elements, similar users and recommended events.

With reference to Table 2, the average case is interesting: with few similar users the system is able to recommend a number of elements almost equal to the elements of interest for the user. Of particular interests are also the conditions in which the recommendation engine returns no elements. This happens for mainly two reasons: (a) few or no friends, which is generally caused by having too few elements, or for having
very specific interests, or (b) too many elements, since it is very likely that the user has no similars with different elements.

<table>
<thead>
<tr>
<th></th>
<th>no-repeat</th>
<th>weekly</th>
<th>daily</th>
<th>mon-ri</th>
<th>mon-sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>discrete</td>
<td>2.5</td>
<td>1</td>
<td>0.5</td>
<td>0.22</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 3: Average numbers of discrete and recommended elements per user.

As can be seen in Table 3, the average number of no-repeat events is rather high wrt the other classes. In fact, the no-repeat events are nearly 85% of total discrete elements. Nonetheless, it is important to notice the much lower average number of no-repeat recommended elements; in fact, as the high majority of similars to a particular user is likely to have in common with her no-repeat events, it is difficult to identify non shared elements suitable for recommendation.

A side-effect of the aggregation procedure is the definition of each periodicity top hits. For each periodicity, the set of the most programmed recordings (but, in fact, these are discrete elements) is computed and suggested to the users of the VCast service. This result has to be considered as a first “raw” version of recommendation.

4.1 Title Precision

One of the main goals of this preliminary analysis is the evaluation of the correctness of the titles chosen by the aggregation methodology, explained in Section 3.1, for the discrete elements. To achieve this, we performed several checks to validate the chosen titles against those retrieved from a real programming guide. The basic idea behind this session of tests is to compute all matches between two different lists: (a) the list of all discrete elements representing the VCast recordings for a specific day (i.e., D), and (b) the broadcast programming of that day (i.e., G). We then focused both on the channel and the starting time of programs in D and we attempted to find a match in G, aside from an error of 10 minutes for the timing. As a hit (a candidate) is found, we tested the string distance between our assigned title and the real one. The implemented function for checking the distance between two strings exploits the Levenshtein algorithm [2], plus a number of improvements for correcting the returned value. The results are given in Table 4 and consider one single month (i.e., February 2009).

<table>
<thead>
<tr>
<th>candidates</th>
<th>exact</th>
<th>non-exact</th>
<th>false-pos</th>
<th>false-neg</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>379</td>
<td>79</td>
<td>300</td>
<td>0</td>
</tr>
<tr>
<td>%</td>
<td>100</td>
<td>21</td>
<td>79</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4: Average results for February 2009.

On a total number of 11859 discrete elements that could fit for the considered period, we could identify an average of 379 possible candidates each day. In G we do not have any information on the periodicity of the programming; however, we can infer this from the periodicity attribute of the discrete elements, which contributes to define a more complete program information wrt to the considered EPG. For example, for the discrete elements marked as weekly, we have to select only those which fall in the exact day under consideration. Regarding the starting time, we defined an acceptability threshold of 10 minutes (before and after).

It is important to notice that: (a) 21% of the computed discrete elements has an exact match in the programming guide; (b) the percentage of false positives occurrences (i.e., wrong titles that are matched as valid) is 0, whilst the false negatives (i.e., correct titles matched as invalid) represent about 6% of the unmatched titles. False negatives are due to an excessive distance value between titles, even if the meaning of the given title is related to the broadcasted content. Notice also that, being more permissive in the interpretation of the user given title leads to an increased false negative rate, which can be estimated as around 10% of unmatched EPG titles.

5. CONCLUSIONS AND FUTURE WORKS

The data analysis is not exhaustive yet, because it covers only a limited number of days, where the number of distinct recordings and their adherence to the real broadcasts can vary in a strong way. Among non-exact elements, notice that there exist a moderate subset of titles (10%) whose text is completely different from the real one, although the meaning of the user’s given title is clear and has some strong “semantic” relation with the broadcasted content. The recommendation engine is now under heavy upgrading. We are introducing a more sophisticated way to compute the similarity between users and also to order in a better way the recommended podcasts. Our aim is to investigate a way of exploiting the developed analysis system in order to automatically build up a user-interests-based EPG, on top of a collaborative and social experience.

6. REFERENCES