Proceedings of the Second International Conference on Machine Learning and Cybernetics, Xi'an, 2-5 November 2003

ADAPTIVE PROGRAM FILTERING UNDER VECTOR SPACE MODEL AND RELEVANCE FEEDBACK

ZHI-WEN YU, XING-SHE ZHOU, JIAN-HUA GU, XIAO-JUN WU

Department of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an, 710072, P.R.China
E-MAIL: yuzhiwen77@sina.com

Abstract:

The overabundance of DTV (Digital Television) programs precipitates a need for smart "filters" to help people obtain programs that they really like. In this paper, we propose an adaptive program filtering system, which is designed to assist users by adapting to their personal preferences. We firstly provide architecture of the program filtering system. Secondly, we present the user profile and program feature representation model and similarity measurement using vector space model. Thirdly, we describe the user profile learning algorithm based on relevance feedback. For user profile learning, we first put forward a primary learning algorithm. With several issues in further consideration, we then present the improved learning algorithm, which is more reasonable and comprehensive than the primary one. Finally, we present the performance evaluation on the prototype of the system.

Keywords:

Program; Filtering; User profile; Vector space model; Relevance feedback

1 Introduction

Digital television (DTV) and the rapid growth of communication technologies have created an overabundance of programs from which each consumer can choose. This precipitates a need for smart "filters" to help people obtain personalized programs.

To meet this new requirements, the TV-Anytime Forum, a global association of organizations that seeks to develop specifications to enable audio-visual and other services based on mass-market high volume digital storage, has defined specifications that will enable applications to exploit local persistent storage in consumer electronics platforms [1]. A typical simple TV-Anytime system can be viewed as containing three major elements: a service provider delivering the TV-Anytime service, a transport provider that carries the service and a piece of equipment in the home that stores the content and plays it back at the consumer's request [2].

In the TV-Anytime context, the central element is a new generation of consumer electronic equipment called Personal Digital Recorder (PDR). PDR is a kind of personal digital media storage device that is widely expected to become an extremely popular consumer electronic device for DTV in the near future.

The rapid development of PDR enables consumers to store large quantities of rich multimedia content for their personal use. TV-Anytime in turn provides new opportunities for content and service providers to offer large amounts of personalized multimedia for the benefit of consumers. In a word, TV-Anytime and PDR make new solutions providing *adaptive* and *personalized* services possible. Therefore, this paper proposes an adaptive program filtering system in the TV-Anytime environment, which is designed to assist users by adapting to their personal preferences.

The rest of this paper is organized as follows. Section 2 provides the architecture of the filtering system. Section 3 presents the analysis and design of VSM based feature representation and similarity measurement method. Section 4 presents user profile learning algorithm based on relevance feedback. Section 5 describes performance evaluation on the prototype of the system. Finally, Section 6 summarizes the conclusions of this paper.

2 System Design

Figure 1 shows the architecture of our adaptive program filtering system. There are three main elements: filter engine, profile learning, and user profile. The filter engine filters the incoming live program broadcast to PDR, only recording the program that the system thinks the user would like. The profile learning is responsible to update the user's profile according to his/her viewing history. User profile is the program preferences or interests of individual user.

When a program arrives, the filter engine will determine whether to record on the basis of preference information from the user profile. The filtering scheme is presented in Section 3. The user profile can be learned

automatically by the system through machine learning algorithm presented in Section 4.

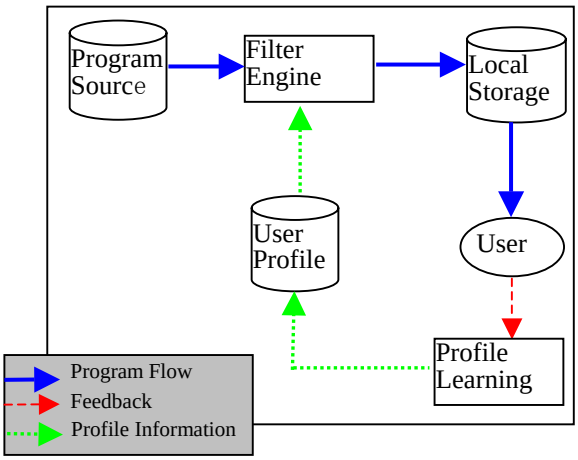


Figure 1. System conceptual diagram

3 Feature Representation and Similarity Measurement Using Vector Space Model

3.1 Feature Representation

We take VSM [3] as the feature extraction and object information presentation method. In the VSM, we identify an object by a set of terms. Weights are assigned to terms as important indications. If m distinct terms are available for content identification, the content C can be conceptually represented as a m-dimensional vector, C=($w_1,...w_m$), where w_i is the weight assigned to the i-th term.

To compute the vector representation of an object, usually these steps are followed [4]. First the individual words occurring in the object are identified. Words that belong to the *stop list*, which is a list of high-frequency words with low content discriminating power, like "a", are deleted. Then a stemming routine is applied to reduce each remaining word to word-stem form, that is, the remaining words are reduced to their stem by removing prefixes and suffixes. For instance the words "computer", "computers", "computing" and "computability" could all be reduced to "comput". For each remaining word stem (a term), a weight is assigned in an attempt to represent how "important" that term is. This is used for decreasing redundancy.

In the user's profile, there may be a lot of terms, which indicate the user's interests. The terms have weights and orders respectively. We can define each term as a 3-tuple (term, weight, order). Hence a profile can be represented as a vector of these 3-tuples, if there are m distinct terms in the profile, and then it will be represented as a vector like this:

$$P = ((t_1, w_1, 1), (t_2, w_2, 2), ..., (t_m, w_m, m)) \quad (1)$$

$$w_i \geq w_{i+1} \qquad (1 \leq i \leq m) \quad (2)$$

where t_i is a term, w_i is the weight of term t_i, i is the order of t_i in the profile. The weights and orders describe the relative importance of the terms in the profile. Assuming there is a user who has 7 terms (with weight and order) in his/her profile. The user profile is represented as:

P=((Romance, 4.25, 1) (Vivien Leigh, 3.16, 2) (Titanic, 2.41, 3), (Hero, 1.80, 4), (Love, 1.44, 5), (Church, 1.28, 6), (Marriage, 1.12, 7))

For computational reasons, we can take the top n highest weighted terms to represent the user's preference. So the user's profile can be conceptually represented as the following vector:

$$P = (w_1, ..., w_n) \quad (3)$$

where w_i is the weight of term t_i in the profile. Supposing $n = 6$, then the profile example can be represented as:

$$P = (4.25, 3.16, 2.41, 1.80, 1.44, 1.28)$$

Similarly, content can be also represented as a vector with n items, which are the same as those in the profile vector, that is term t_i in the profile vector and content vector are the same:

$$C = (u_1, ..., u_n) \quad (4)$$

where u_i is the weight assigned to term t_i. Since terms are not all equally important for program representation, for instance, terms in Actors field may be more important than those in the Keywords field, important factors are assigned to the terms in proportion to their presumed importance for program identification. The weight u_i is assigned complying with this rule: if term t_i is included in the Title, Genre or Actor field of the program's metadata, then u_i=2, if t_i is included in the Keywords field, then u_i=1, otherwise u_i=0. For example, in metadata of the film: "Gone with the Wind", the Genre is "Romance", the Actor includes "Vivien Leigh", and the Keywords includes "Love". So the film can be represented as:

$$C = (2, 2, 0, 0, 1, 0)$$

3.2 Similarity Measurement

In the typical vector space representation, we can measure the degree of similarity between a program-profile

pair on the weights of the corresponding terms. A distance metric that measures the proximity of vectors to each other is defined over the space. After research, we decide to use the classical cosine similarity measure to compute the similarity between program and profile representations. Given the program $C = (u_1,...,u_n)$ and the profile $P = (w_1,...,w_n)$, the cosine similarity can be calculated as follows:

$$sim(C,P) = \frac{C \times P}{\| C \| \times \| P \|} = \frac{\sum_{i=1}^{n} u_i w_i}{\sqrt{\sum_{i-1}^{n} u_i^2 \sum_{i=1}^{n} w_i^2}} \quad (5)$$

Consider the program $C = (2, 2, 0, 0, 1, 0)$ and the profile $P = (4.25, 3.16, 2.41, 1.80, 1.44, 1.28)$, their cosine similarity can be calculated as:

$$sim(C,P) = \frac{4.25\times 2 + 3.16\times 2 + 1.44\times 1}{\sqrt{(4.25^2 + 3.16^2 + 2.41^2 + 1.80^2 + 1.44^2 + 1.28^2)(2^2 + 2^2 + 1^2)}} = 0.85$$

In the context of filter engine, if the calculated similarity is above the preset threshold θ (such as 0.4), we consider that the program is relevant to the user's profile; in other words, the user is likely interested in the program. Then the filter engine will record the program for the user.

4 User Profile Learning with Relevance Feedback

Generally, there are two ways to learn and update the user's preference knowledge: (1) learn the user's preference knowledge according to his/her implicit feedback; (2) learn the user's preference knowledge according to his/her explicit feedback. We think the second method does not suit our system. Because in a PDR context it is inconvenient for the user to give explicit feedbacks to the DTV content they are viewing. After all, the original motivation and one major advantage of PDR is that with the intelligent assistant resident in the PDR, the user gets a more smooth and personalized viewing experience. Therefore we adopt the first method: learn the user's preference knowledge according to his/her implicit feedback.

There exist a variety of machine learning methods that can be used for learning a user profile. We adopt the relevance feedback method because it is both efficient and effective. Relevance feedback [5] was introduced originally for information retrieval. In relevance feedback method, users provide feedback to the system about the data items that they have been sent. The system then uses this feedback to adjust the user's profile.

In our system, when the user has watched a program for a period of time, then switched to another one. The user's profile will be refined and revised based on the feedback received. This is done through the modification of the preference terms and their weights respectively.

4.1 The Primary Learning Algorithm

The primary algorithm is depicted as follows:

For those terms already present in the profile, the term-weights are modified in proportion to the feedback.

$$w_i' = w_i + \alpha \times \beta \times f(i) \quad (6)$$

$$\beta = \frac{T_r}{T_t} \in [0,1] \quad (7)$$

$$f(i) = \frac{I_{\max} - i}{I_{\max}} \qquad 1 \le i \le I_{\max} \quad (8)$$

where w_i' is the weight of term t_i after the feedback, as well as w_i is the weight of term t_i before the feedback. α is the learning rate which indicates the sensitivity of the profile to user feedback. If the user think the feedback is very important and wants the profile learning to be fast, then α can be set larger, other wise α can be set smaller. β is the ratio of user's real watching time (T_r) to the program's total duration time (T_t). β can be considered as the user's evaluation to the program which he/she has viewed. $f(i)$ reflects the influence of the order of the term in user's viewing history to the weight update process. The more important terms with larger weight reasonably have more influence on the profile learning. So $f(i)$ should decrease with increasing order of the term. In the expression of $f(i)$, i is the order of term t_i in the user's profile; $I_{\max}$ is the maximum of i, in other words, it is the total number of terms in the user's profile. For instance, if t_i is the 2-th term in the profile, which holds 10 terms totally, then $f(i) = f(2) = \frac{10-2}{10} = 0.8$.

Terms not existing in the profile are handled as follows:

Firstly, calculate the term's weight

$$w_i = \alpha \times \beta \times f(i) \quad (9)$$

here w_i is the weight of the new term t_i, α and β have the same meanings as above. Since term t_i is not in the profile before, so $f(i)$ can't be calculated as above, we define it as a default value ε (such as 0.1).

Secondly, if the calculated w_i is higher than a preset threshold λ (such as 0.05), we will add it to the user's

profile, otherwise discard it, because it is too trivial.

4.2 The Improved Learning Algorithm

Usually the user will only watch what he/she likes unless he/she accidentally switches to something he/she does not like. We should eliminate these accidental switches' impact on user profile learning. Generally, these accidental wrong switches can be filtered from the user's viewing history by a TST (Trashy Switch Time). This TST is a threshold value, for example, we can set TST = 2s, and therefore all those viewing pieces in the user's viewing history which has a viewing time less that TST will be considered as accidental wrong switches. Through TST those accidental wrong switches can be filtered from the user's viewing history.

However, to different contents, TST has different significance. For example, 2s is long for an advertisement lasting 5s, but it is very short for a film lasting 2 hours. So the ratio of user's real watching time to the content's total duration time is significant. We assume that if $\frac{T_r}{T_t}$ is larger than a threshold μ (such as 0.01), the user really likes the content; otherwise, the user dislikes it.

Another thing needing further consideration is the expression of $f(i)$. Since top terms may have more influence on preference revision, while last terms have little impact. So $f(i)$ should differ markedly for the top terms, and differ slightly for the last terms. The descending of $f(i)$ should reflect this. In the primary learning algorithm, $f(i)$ is a linear decreasing function, which decreases at the same speed. The curve of the linear decaying $f(i)$ is shown in Figure 2. In order to get different decreasing speed, we adopt exponential decay mode to construct $f(i)$. The curve of the exponential decaying $f(i)$ is shown in Figure 3. In this curve, $f(i)$ decreases rapidly at the beginning, with i increasing, it decreases gently.

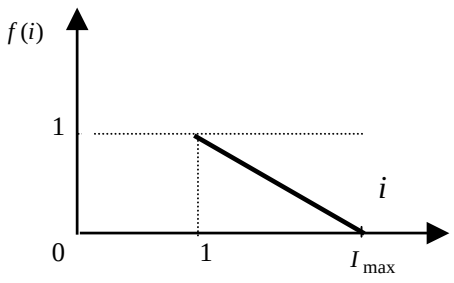


Figure 2: Linear decaying $f(i)$

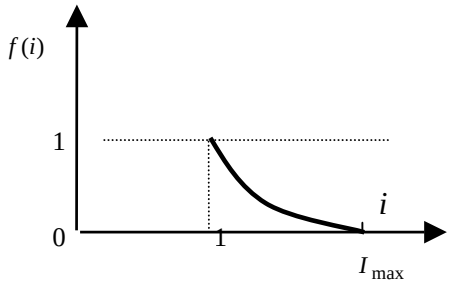


Figure 3: Exponential decaying $f(i)$

With above analysis, we improve the primary algorithm as follows (in following equations, w_i', w_i, α, β, T_r, T_t, i, and $I_{\max}$ have the same meanings as those in the primary learning algorithm.):

If T_r < TST then $w_i' = w_i$, that is not to update the user preference, because the user's switch is an accidental switch.

If $T_r \geq$ TST and $\frac{T_r}{T_t} \geq \mu$, which means the user really likes the program, then for those terms already present in the profile, the term-weights are modified according to the following equations.

$$w_i' = w_i + \alpha \times \beta \times f(i) \tag{10}$$

$$\beta = \frac{T_r}{T_t} \in [0,1] \tag{11}$$

$$f(i) = \begin{cases} e^{-\frac{i}{I_{\max} - i}} & 1 \leq i < I_{\max} \\ 0 & i = I_{\max} \end{cases} \tag{12}$$

Terms not existing in the profile are handled as follows:

Firstly, calculate the term's weight, $f(i)$ is defined as a default value ε.

$$w_i = \alpha \times \beta \times f(i) \tag{13}$$

Secondly, if the calculated w_i is higher than a preset threshold λ, we will add it to the user's profile, otherwise discard it.

If $T_r \geq$ TST and $\frac{T_r}{T_t} < \mu$, which means the user dislikes the program, then for those terms already present in the profile, the term-weights are modified according to the following equations.

$$w_i' = w_i - \alpha \times \beta \times f(i) \tag{14}$$

$$\beta = \frac{T_r}{T_t} \in [0,1] \tag{15}$$

$$f(i) = \begin{cases} e^{-\frac{i}{I_{\max} - i}} & 1 \le i < I_{\max} \\ 0 & i = I_{\max} \end{cases} \tag{16}$$

Terms not existing in the profile are handled as follows:

Firstly, calculate the term's weight, $f(i)$ is defined as a default value ε.

$$w_i = -\alpha \times \beta \times f(i) \tag{17}$$

Secondly, if the absolute value of calculated w_i is higher than a preset threshold λ (that is, $|w_i| > \lambda$), we will add it to the user's profile, otherwise discard it.

According to the improved learning algorithm, when the user has watched the film "Gone with the Wind" for 160 minutes (whose total time is 180 minutes), then switched to another program, and finally the weight of *Love* in his/her profile is revised as follows (supposing $\alpha = 0.5$, $I_{\max}$ =10):

$$W_{Love}' = W_{Love} + \alpha \times \beta \times f(3) = 2.41 + 0.5 \times \frac{160}{180} \times e^{-\frac{5}{10-5}} = 2.57$$

5 Evaluation

From the user's point of view, filtering effectiveness is very important. There are two criterions for evaluating filtering effectiveness, which are precision and recall. In general, precision can be used as a measure of the ability of our system to present only relevant programs.

$$\text{Precision} = \frac{\text{number of relevant programs recorded}}{\text{total number of programs recorded}} \times 100\%$$

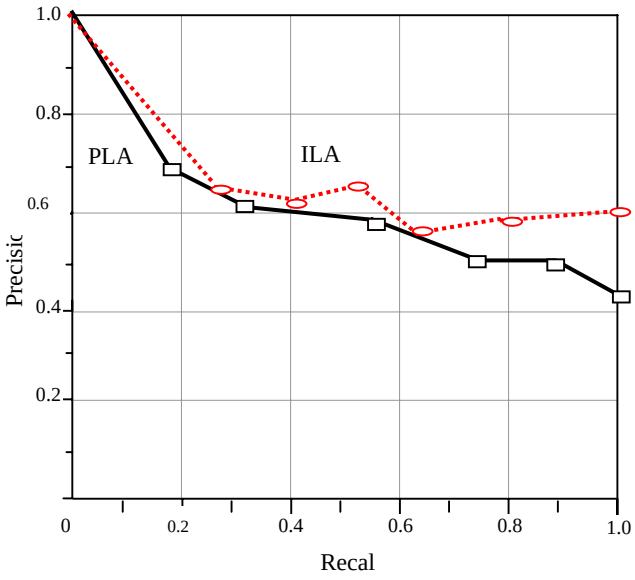
Recall can be used as a measure of the ability of our system to present all relevant programs.

$$\text{Recall} = \frac{\text{number of relevant programs recorded}}{\text{number of relevant programs in collection}} \times 100\%$$

Figure 4 shows the results of our experiment. The figure consists of two graphs. One is the experimental result employing the primary learning algorithm; the other is the experimental result employing the improved learning algorithm. Each graph is a plot of precision versus recall. In the figure, PLA denotes Primary Learning Algorithm, while ILA denotes Improved Learning Algorithm.

Comparing the results of the two learning algorithms, we can see that the improved learning algorithm is superior to the primary learning algorithm. This can be seen from that the improved learning algorithm's curve is closer to the upper right-hand corner of the graph (where recall and precision are maximized). The reason is that the improved learning algorithm is more reasonable and comprehensive by filtering user's accidental wrong switches, judging user's likes and dislikes on $\frac{T_r}{T_t}$, and varying the decreasing speed of $f(i)$.

Figure 4: Recall-precision graph



6 Conclusions

In this paper, we propose an adaptive program filtering system in the TV-Anytime environment, which is based on vector space model and relevance feedback. The system is designed to assist users by adapting to their personal preferences. We have made performance evaluation on the prototype of the system. The experimental results are encouraging, which show the system proposed here is useful to consumers. With this tool, when a user sits down to watch TV there are always some interesting programs to watch in the PDR.

Some improvements of the current prototype are under way. First, we will implement adaptive size of user profile. This can be done by adjusting the value of threshold λ (see the details in Section 4) through the system running. It is in fact a problem of tradeoff between storage space and accuracy. Another improvement is to optimize the preference learning algorithm. We believe these challenges will require more substantial thinking in our system.

Acknowledgements

This paper is supported by the National Natural Science Foundation of China (60073054).

References

[1] TV-Anytime Environment Requirements Document, TV035r6, TV-Anytime Forum, Aug. 2000
[2] TV-Anytime System Requirements Document, TV036r2, TV-Anytime Forum, Apr. 2000
[3] G. Salton. "Automatic Text Processing: The transformation, analysis, and retrieval of information by computer". Addison-Wesley, Reading, Massachusetts, USA, 1989.
[4] Tak W Yan and Hector Garcia-Molina. "Index Structures for Information Filtering Under the Vector Space Model", In Proceedings of the Tenth International Conference on Data Engineering, Houston, USA, 1994
[5] P W Foltz and S T Dumais. "Personalized information delivery: An analysis of information filtering methods". Communications of the ACM, 1992
[6] Yu Zhiwen, Zhou Xingshe, Wu Xiaojun, Gu Jianhua. "A Hybrid Filter for Program Personalization". In Proceedings of the 4th IEEE International Conference on Information Technology: Computers and Communications (ITCC2003), PP645-649, USA, 2003.