Two Recommendation Algorithms Based on Deformed Linear Combinations*

Alexander D'yakonov

Moscow State University, Moscow, Russia,
djakonov@mail.ru,

Abstract. Data mining for recommender systems has gained a lot of interest in the recent years. "ECML/PKDD Discovery Challenge 2011" was organized to improve current recommender system of the VideoLectures.Net website. Two main tasks of the challenge simulate new-user and new-item recommendation (cold-start mode) and clickstream based recommendation (normal mode). This paper provides detailed descriptions of two simple algorithms which were very successful in the both tasks. The main idea of the algorithms is construction of a linear combination equal to a vector of estimations of lectures popularity after viewing a certain lecture (or lectures). Each addend in the combination describes similarity of lectures using the part of the data. The algorithms are improved by transforming the combination to non-linear function. Lectures with the highest estimations of popularity are recommended to users.

1 Introduction

Algorithms which have taken the first places in the competition "ECML/ PKDD Discovery Challenge 2011 (VideoLectures.Net Recommender System Challenge)" [1] are described. The competition was focused on algorithm development for making recommendations for video lectures, based on historical data from the VideoLectures.Net website [2]. The competition consisted of two independent tasks. In the first task it was necessary to recommend a list of "new lectures" (which had been published on the portal recently). So there was not information on popularity of the new lectures, only their detailed descriptions were available. In the second task it was necessary to recommend lectures from the entire lecture set, using information on viewed triple of lectures. The tasks are described in detail below. We do not describe the evaluation metrics used by organizers and the data offered to participants that have not been used by our algorithms. Algorithms are simple enough, universal, can be used for different problems.

2 First Task "Cold Start"

Descriptions of the lectures from VideoLectures.net website are available. Every lecture has the lecture id, the language of the lecture ("English", "Slovene", "French", etc.),

* This work was supported by the Russian Foundation for Basic Research, project 10-07-00609; by the President of the Russian Federation, project no. MD-757.2011.9. The author is also grateful to the organizers of "ECML/PKDD Discovery Challenge 2011" for running the interesting competition.

the categories of the lecture (for example "Machine Learning", "Biology"), the total (aggregated) number of views, the date when the lecture was published on the portal, the authors of the lecture (their ids, names, e-mails, homepages), the name of the lecture (a sentence in the specified language), the lecture description (a small text). The lectures were given at events (conferences, summer schools, workshops, etc.). Similar information is available on the events. Besides, for every pair of lectures the total number of the users viewed both lectures is known (if the number is more than one). Other information, for example, the descriptions of the slides or the dates, when the lectures were recorded, was also available, however it was not used in the final version of the algorithm. Note that some data is unknown (for example, descriptions are not known for all lectures).

The set of the described lectures is divided into two subsets: "older lectures" (all information is available) and "new lectures" (which have been published on the portal recently, so the viewing information is not available). A test set is a subset of the older lectures set. The task is to recommend the list of 30 new lectures for every lecture from the test set (it is recommendation of new lectures to a new user who has watched one lecture).

3 Algorithm for Solving the First Task

Let some information on a lecture can be written as the n-dimensional vector $f = (f_1, \ldots, f_n)$. For example, if n is the number of the authors of all lectures than the binary vector f describes the authors of concrete lecture: $f_i = 1$ iff the i-th author is the author of the lecture. It is similarly possible to describe the language of the lecture, its categories, etc. Naturally, the vectors describing different types of information are of different dimensionality. In each case it is possible to estimate similarity of the lectures. For example, for the i-th lecture presented by the vector $f(i) = (f_1(i), \ldots, f_n(i))$ and the j-th lecture presented by the vector $f(j) = (f_1(j), \ldots, f_n(j))$ their similarity is estimated as changed cosine similarity [3]

$$\langle f(i), f(j) \rangle = \frac{f_1(i)f_1(j) + \cdots + f_n(i)f_n(j)}{\sqrt{f_1(i)^2 + \cdots + f_n(i)^2 + \varepsilon}\sqrt{f_1(j)^2 + \cdots + f_n(j)^2 + \varepsilon}} \ . \qquad (1)$$

The change "$+\varepsilon$" is for preventing division by zero (for example, if the authorship of a lecture is unknown). In the final version of the algorithm $\varepsilon = 0.01$.

Idea of the algorithm is very simple: for the test lecture to calculate similarity to each new lecture by summing (with some coefficients) values (1) for all presented "types of information" (language, categories, authors, etc.). First, we will mark the main modification of the algorithm which essentially improves performance. Together with similarity to the lecture from the test set it is necessary to consider similarity to co-viewed older lectures (similar from the point of view of users' behavior).

Let the set of older lectures be indexed by numbers from I, let $f(i)$ be the vector of the description of the i-th lecture, let m'_{ij} be the estimation of similarity of the i-th and the j-th lectures (from the point of view of users' behavior, see below). Then let

$$f'(i) = \sum_{j \in I} \left(m'_{ij} \frac{f(j)}{\sqrt{f_1(j)^2 + \ldots + f_n(j)^2 + \varepsilon}} \right)$$

and similarity to the new t-th lecture is calculated by summing of $\langle f'(i), f(t)\rangle$ for all types of information. Let us describe how values m'_{ij} are calculated. Let L be the number of lectures, m_{ij} be the number of the users that viewed both the i-th and the j-th lectures, $i \in \{1, 2, \ldots, L\}$, $j \in \{1, 2, \ldots, L\}$, $i \neq j$, and m_{ii} be the number of views of the i-th lecture divided by 2 (such "strange" definition of the diagonal elements is a result of optimization of algorithm performance). Then

$$m'_{ij} = \frac{m_{ij}}{\sum\limits_{t=1}^{L} m_{it}} \ .$$

The sense of this value is clear enough. If the numbers m_{ii} were equal to zero, $i \in \{1, 2, \ldots, L\}$, than it would be an estimation of probability that user viewed the j-th lecture under the condition that he viewed the i-th (the performance of the algorithm was 26.02%, see below). Nonzero diagonal elements are necessary to consider also similarity to the i-th lecture, not only to co-viewed lectures (the performance was 29.06%, without division by 2 the performance was 28.17%).

Let us enumerate types of information which were used to calculate similarity. For each type we will specify the vector

$$\gamma_{\text{index}} = (\langle f'(i), f(j_1)\rangle, \ldots, \langle f'(i), f(j_r)\rangle) \ ,$$

where $J = \{j_1, \ldots, j_r\}$ is the set of new lecture indexes.

1. Similarity of categories. Here $f(j)$ is the characteristic vector of lecture categories, i.e. a binary vector, in which the t-th coordinate is equal to 1 iff the j-th lecture belongs to the t-th category. As a result we receive the vector γ_{cat}.

2. Similarity of authors. Here $f(j)$ is the characteristic vector of lecture authors, i.e. a binary vector, in which the t-th coordinate is equal to 1 iff the t-th author is the author of the j-th lecture. As a result we receive the vector γ_{auth}.

3. Similarity of languages. Here $f(j)$ is the characteristic vector of lecture language, in which the first element corresponding to English is set to 1 (to make all lectures similar on lectures in English, because lectures in English are popular among Internet users). As a result we receive the vector γ_{lang}.

4. Similarity of names. At first all words which are included into names and descriptions of the lectures are parsed and reduced to word stems (we used Porter Stemmer [4], [5]). Note, all special symbols (brackets, commas, signs of arithmetic operations, etc.) were deleted, but stop words were reserved (it does not essentially influence performance of the algorithm). The name of every lecture is described by the vector $(h_1, \ldots, h_W)$, in which h_i is the number of words with the i-th word stem. Then we apply TF-IDF-like weighting scheme [3]:

$$f_i = \frac{h_i}{\sqrt{w_i + \varepsilon}} \ , \tag{2}$$

where w_i is the total number of words with the i-th word stem in names and descriptions of all lectures. Such vectors $(f_1, \ldots, f_W)$ are used for calculation of the vector γ_{dic}. Note that for calculation of similarity of names we use information on names and descriptions (for weighting scheme). Standard TF-IDF proved to perform a bit worse ($\sim$ 1–4%).

5. Similarity of names, descriptions, names and descriptions of events. Each lecture has the name, the description (may be empty), the name of appropriate event and the event description (if information on event is not present we consider that the even name and the event description coincide with the lecture name and the lecture description). All it is united in one text, that is described by the vector $(h_1, \ldots, h_W)$, and further operations are the same as in the previous item. As a result we receive the vector γ_{dic2}.

For task solving the algorithm construct the vector

$$\gamma = 0.19 \cdot \sqrt{0.6 \cdot \gamma_{\text{cat}} + 5.6 \cdot \gamma_{\text{auth}}} + \sqrt{4.5 \cdot \gamma_{\text{lang}} + 5.8 \cdot \gamma_{\text{dic}} + 3.1 \cdot \gamma_{\text{dic2}}} \ . \quad (3)$$

Here the square root is elementwise operation. At first the linear combination of the vectors γ_{cat}, γ_{auth}, γ_{lang}, γ_{dic}, γ_{dic2} was used. The coefficients in the linear combination were a result of optimization problem solving. But the usage of square roots gives small improvement of performance (coefficients also tuned solving optimization problem). For optimization problem the method of coordinate descent [6] was used. The algorithm recommends the lectures with the highest values of elements in the vector $\gamma = (\gamma_1, \ldots, \gamma_N)$. Such problem solving technology (selection of various types of information, construction of an appropriate linear combination, its further tuning and "deformation") is being developed by the author and is named "LENKOR" (the full description of the technology will be published in the near future).

In the final submitted solution one more change was made: the vector $\gamma = (\gamma_1, \ldots, \gamma_N)$ was transformed to the vector

$$\left(\gamma_1 \cdot \left(1 + \delta \frac{t_{\max} - t_1}{t_{\max} - t_{\min}}\right), \ldots, \gamma_N \cdot \left(1 + \delta \frac{t_{\max} - t_N}{t_{\max} - t_{\min}}\right)\right) \ , \quad (4)$$

where t_j is the time (in days) when the j-th new lecture was published, $t_{\min}$ is the minimum among all these times, $t_{\max}$ is the maximum. The transformation increased performance approximately on 5%. The reason of the transformation is that it is important how long is lecture available online (not only popularity of the lecture). In the final version of the algorithm $\delta = 0.07$, because this value maximizes performance of the algorithm in uploads to the challenge website [1] (37.24% for $\delta = 0.09$, 37.28% for $\delta = 0.07$, 36.24% for $\delta = 0.05$).

The described algorithm has won the first place among 62 participants with the result of 35.857%. We do not describe the evaluation metric, the interested reader can find the definition of the metric on the challenge website [1]. For local testing we used the same metric. After data loading and processing (it takes 1 hour, but can be launched once for recommender system construnction) the running time for the first task solving was 17.3 seconds on a computer HP p6050ru Intel Core 2 Quad CPU Q8200 2.33GHz, RAM 3Gb, OS Windows Vista in MATLAB 7.10.0. 5704 recommendations (30 lectures in each) were calculated. The dictionary consisted of 35664 word stems.

4 Second Task "Pooled Sequences"

In the second task the training set T consists of triples $\{a, b, c\}$ of lecture numbers. For every triple the number $n(\{a, b, c\})$ of users who viewed all three lectures is known. Besides, the pooled sequence [1] is specified. It is the ranked list of lectures which were

viewed by users after all lectures of the triple $\{a, b, c\}$. The numbers of views are also known (the pooled sequence is ordered according to these values). Definition and examples of pooled sequence construction can be found on the official site of the competition [1]. We formalize this concept by means of the vector $v(\{a, b, c\}) \in \mathbb{Z}^L$, where L is the number of lectures,

$$v(\{a, b, c\}) = (v_1(\{a, b, c\}), \ldots, v_L(\{a, b, c\})) \ ,$$

$v_j(\{a, b, c\})$ is the total number of views of the j-th lecture after triple $\{a, b, c\}$ (informally speaking, it is popularity of the j-th lecture after lectures from $\{a, b, c\}$). The test set also consists of triples (the test set is not intersected with the training set). The task is to define pooled sequences for triples from the test set, to be exact 10 first members of the sequences (10 highest elements of each vector $v(\{a, b, c\})$. So, these 10 lectures should be recommended to user after viewing the three lectures.

5 Algorithm for Solving the Second Task

At first, two normalizations of the vectors corresponding to triples from the training set T are performed, the first is

$$v'(\{a, b, c\}) = \left(\frac{v_1(\{a, b, c\})}{\log(|\{\tilde{t} \in T \,|\, v_1(\tilde{t}) > 0\}| + 2)} \cdots \right.$$
$$\left. \cdots \frac{v_L(\{a, b, c\})}{\log(|\{\tilde{t} \in T \,|\, v_L(\tilde{t}) > 0\}| + 2)} \right) \ . \quad (5)$$

It is clear that $|\{\tilde{t} \in T \,|\, v_j(\tilde{t}) > 0\}|$ is the number of triples from the training set such that their pooled sequences include the j-th lecture. The reason for performing such normalization is that lectures included into many pooled sequences are generally less relevant. The second normalization is

$$v''(\{a, b, c\}) = \left(\frac{v'_1(\{a, b, c\}) \cdot \log\left(\sum\limits_{j=1}^{L} v'_j(\{a, b, c\}) + 1 \right)}{\sqrt{3 \cdot n(\{a, b, c\}) + \varepsilon}} \cdots \right.$$
$$\left. \cdots \frac{v'_L(\{a, b, c\}) \cdot \log\left(\sum\limits_{j=1}^{L} v'_j(\{a, b, c\}) + 1 \right)}{\sqrt{3 \cdot n(\{a, b, c\}) + \varepsilon}} \right) \ , \quad (6)$$

$\varepsilon = 0.01$. It is difficult enough to describe sense of this normalization. It was a result of exhaustive search of different variants and increased performance by 1–2%, that was essential in competition.

Let

$$s(\tilde{d}) = \sum_{\tilde{t} \in T \,:\, \tilde{d} \subseteq \tilde{t}} v''(\tilde{t})$$

and $n(\tilde{d}) = |\{\tilde{t} \in T : \tilde{d} \subseteq \tilde{t}\}|$ be the number of addends in the sum, T be the training set. For example, $s(\{a,b\})$ is the sum of vectors $v''(\{a,b,d\})$ for all d such that $\{a,b,d\} \in T$. Let operation ω deletes from a vector all zero elements except one and adds one zero if there was not any zero element. For example, $\omega(1,0,0,2,0) = (1,0,2)$, $\omega(1,2) = (1,2,0)$. Let

$$\mathrm{std}(x_1,\ldots,x_n) = \sqrt{\frac{1}{n-1}\sum_{t=1}^{n}\left(x_t - \frac{1}{n}\sum_{i=1}^{n} x_i\right)^2}$$

(standard deviation [7]).

The algorithm is very simple: for the triple $\{a,b,c\}$ from the test set if there are at least two nonzeros among numbers $n(\{a,b\})$, $n(\{a,c\})$, $n(\{b,c\})$ (it corresponds to having enough information) than

$$\begin{aligned}\gamma = &\frac{\log(s(\{a,b\})+0.02)}{\mathrm{std}(\omega(s(\{a,b\})))+0.5} + \\ &\frac{\log(s(\{b,c\})+0.02)}{\mathrm{std}(\omega(s(\{b,c\})))+0.5} + \\ &\frac{\log(s(\{a,c\})+0.02)}{\mathrm{std}(\omega(s(\{a,c\})))+0.5}\ .\end{aligned} \tag{7}$$

Otherwise we add to this sum addends

$$\frac{\log(s(\{a\})+0.02)}{\mathrm{std}(\omega(s(\{a\})))+0.5} + \frac{\log(s(\{b\})+0.02)}{\mathrm{std}(\omega(s(\{b\})))+0.5} + \frac{\log(s(\{c\})+0.02)}{\mathrm{std}(\omega(s(\{c\})))+0.5}\ . \tag{8}$$

Here the log is taken elementwise. Elements of the received vector γ are treated as the estimations of popularity of lectures from pooled sequence (the higher value, the more popular). Lectures with the highest estimations are recommended.

Let us try to explain the process of the development of the algorithm. It is very logical to operate simply by a rule

$$\begin{aligned}\gamma &= \log(s(\{a,b\})) + \log(s(\{b,c\})) + \log(s(\{a,c\})) = \\ &= \log(s(\{a,b\}) \cdot s(\{b,c\}) \cdot s(\{a,c\})),\end{aligned} \tag{9}$$

where "$\cdot$" is the elementwise multiplication of vectors. Indeed, if there is no information on triple $\{a,b,c\}$ we parse triples $\{a,b,d\}$ for all d. Thus we sum vectors $v(\{a,b,d\})$ and receive a vector $s(\{a,b\})$. It corresponds to union of multisets [8]. Similarly for triples $\{a,c,d\}$, $\{b,c,d\}$ we receive vectors $s(\{a,c\})$ and $s(\{b,c\})$. Now it is logical to intersect the received multisets. Standard operation for intersection in the theory of multisets is min (minimum). However in our experiment product proved to be better:

$$s(\{a,b\}) \cdot s(\{b,c\}) \cdot s(\{a,c\})\ ,$$

this operation is popular in the theory of fuzzy sets [9] for intersection (the performance was 49%, and for min the performance was 47%). The expression

$$(s(\{a,b\})+\varepsilon) \cdot (s(\{b,c\})+\varepsilon) \cdot (s(\{a,c\})+\varepsilon)$$

is needed to prevent zeroing many elements of the vector and information losses (the performance became 57%). Then experiments on normalizations of vectors and scaling were made. As a result, division by $\mathrm{std}(\omega(s(\{\cdot,\cdot\}))) + 0.5$ increased performance approximately by 1–3%. Adding of (8) does not influence performance, it was made "just in case".

In this solution the ideas of "LENKOR" were also used: linear combination tuning (for this reason the product (9) was written down as the sum of logs) and subsequent "deformation" (we used data normalizations). Each addend in the linear combination is a vector estimated popularity of lectures. The algorithm did not use detailed descriptions of the lectures as in the first task. In our experiments the usage of descriptions did not improve performance.

The algorithm has won the first place among 22 participants with the result of 62.415%. The algorithm running time on computer HP p6050ru Intel Core 2 Quad CPU Q8200 2.33GHz, RAM 3Gb, OS Windows Vista in MATLAB 7.10.0 for one lecture recommendation is 0.0383 seconds, full task 2 solving (60274 recommendations) takes 38.33 minutes.

The algorithms (for the first and the second tasks) can be efficiently parallelized. Calculations (1)–(9) can be made on matrices to produce several recommendations at once. For this reason we used MATLAB in our experiments: there are efficient matrix calculations in this system.

References

1. N. Antulov-Fantulin, M. Bošnjak, T. Šmuc, M. Jermol, M. Žnidaršič, M. Grčar, P. Keše, N. Lavrač: ECML/PKDD 2011 – Discovery challenge: "VideoLectures.Net Recommender System Challenge", http://tunedit.org/challenge/VLNetChallenge/
2. http://www.videolectures.net/
3. Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze: Introduction to Information Retrieval, Cambridge University Press (2008). http://nlp.stanford.edu/IR-book/information-retrieval-book.html
4. Porter, Martin F.: An algorithm for suffix stripping. Program. 14(3), 130–137 (1980)
5. http://tartarus.org/∼martin/PorterStemmer/matlab.txt
6. T. Abatzoglou, B. O'Donnell: Minimization by coordinate descent, Journal of Optimization Theory and Applications. 36(2), 163–174 (1982).
7. http://en.wikipedia.org/wiki/Standard_deviation
8. Wayne D. Blizard: Multiset theory. Notre Dame Journal of Formal Logic. 30, 1, Winter, 36–66 (1989)
9. http://en.wikipedia.org/wiki/Fuzzy_mathematics