# Recommending VideoLectures with Linear Regression

Martin Možina, Aleksander Sadikov, and Ivan Bratko

Faculty of Computer and Information Science
University of Ljubljana, Slovenia
{martin.mozina,aleksander.sadikov,ivan.bratko}@fri.uni-lj.si

**Abstract.** This paper describes our approach to the task 1 of the ECML PKDD 2011 VideoLectures.Net Recommender System Challenge. The task was to select a set of lectures to be recommended to a visitor of the VideoLecture.Net homepage after already seeing another lecture. Our proposed approach is a hybrid recommender system combining content and collaborative approaches. The core of the system is a linear regression model for predicting the rank of a lecture, whereas by rank we mean the lecture's position in the list of all lectures ordered by the interest of the visitor. Due to the complexity of the problem, the model could not be learned by a classical approach - instead, we had to employ the stochastic gradient descent optimization. The present paper furthermore, through evaluation, identifies and describes some interesting properties of the domain and of the algorithm that were crucial to achieve a higher prediction accuracy. The final accuracy of the model was enough to take the third place in the competition.

## 1 Introduction

In this paper, we describe our approach to the first task (cold start) of the ECML PKDD 2011 Discovery Challenge (VideoLectures.Net Recommender System Challenge) hosted by TunedIT[1]. The task was to make recommendations for video lectures on the VideoLectures.Net website.

Our proposed approach is a hybrid recommender system [2] based on linear regression for predicting the ranks of the newly acquired lectures after viewing some of the "older" lectures. First, we give a short description of the domain, of the learning problem, and present the attributes that we decided to use in learning. Afterwards, we define the problem of learning ranks, present a method for learning a linear regression based on stochastic gradient descent and use it to make content-based and collaborative-based predictions. In section 4, we evaluate the method and assess the results. We finish the paper with conclusions.

## 2 Domain description

The coldstart problem in the challenge was defined as:

1. after seeing an "old" lecture (a lecture present in the system for a while),

[1] http://tunedit.org/challenge/VLNetChallenge

2. recommend 30 of the "new" (published after the 1st of July 2009) lectures that the user might like. The results have to be ranked - more "likeable" lectures should come first.

The organizers collected 6983 training (old) lectures and 1122 test (new) lectures. The task of competitors was to make a selection of 30 test lectures for almost each training lecture and submit the results for evaluation.

The most important part of the provided data was the so-called pairs data set. It contained the frequencies of co-occurrences of lectures from the training set. Two lectures are said to co-occur if they were seen together (in the same browser session, not necessarily consecutively) and the user watched at least half of each of the two lectures. The co-occurrences between training and test lectures were withheld from the competitors and were used to produce the true rankings, which were used to evaluate the submitted solutions.

Provided lectures were described with the following attributes:

**type of lecture** (discrete; sample values: normal lecture, tutorial, debate, invited talk, etc.)
**language** (discrete; sample values: Slovene, English, etc.)
**recorded date** (date value)
**published date** (date value)
**title of lecture** (textual attribute)
**description** (a short textual description of the lecture)
**slide titles** (textual attribute)
**categories** (discrete; a lecture can belong to multiple categories, e.g. machine learning, reinforcement learning, etc.)
**event** (discrete; the event where the lecture was recorded)
**author** (discrete; a lecture can have several authors)
**views** (number of all views of the lecture; this information was given for the training lectures only)

For categories, events and authors, additional information was provided. Each category was specified by its name and a link to its description on Wikipedia. Events were described with: **type** (normal event, project, etc.), **language** of the event, **recorded** and **published dates**, **name** and a short **description** (both textual) of the event. Finally, for each author we were given his or her **name**, **email**, **homepage**, **gender**, and **organization**.

### 2.1 Attributes used in Learning

We manually constructed a set of attributes to be used in learning. Some of them were simply taken as originally provided, while some were combinations of the original attributes. The set of used attributes is summarized in Table 1. Published date is the number of days between the day the lecture was published and the day the snapshot of the database was taken (31st July 2010). Categories is a list of categories that the lecture belongs to. The authors attribute contains the names of actual authors and also their domain names extracted from their email addresses. Therefore, if a lecture was presented

by two persons coming from Google, lecture's value of authors attribute would be: "name of first author", "name of second author", and "google.com". Significant title words are a selection of specific words found in the titles of lectures that, as assumed, imply a higher number of views. These words are: "introduction", "tutorial", "basic", "lecture 1:", "lecture 2:","lecture 3:","lecture 4:", and "lecture". The last attribute, Title words, contains all words mentioned at least once in a title.

The constructed attributes include almost all the information provided, with the exception of the textual attributes, where only titles of lectures were used. Original attributes not included in our model are: recording date of lectures, lecture's description and slide titles, lecture's number of views, description of categories on Wikipedia, anything about events, authors homepage and organization.

**Table 1.** An overview of attributes used in learning. *No. of values* is the number of distinct values found in the training and test data. The column *Can have multiple values?* specifies whether a single lecture can have several values (e.g. a lecture can belong to several categories) or not.

| Name | Type | No. of values | Can have multiple values? |
|---|---|---|---|
| Published date | continuous | n/a | no |
| Type | discrete | 17 | no |
| Language | discrete | 9 | no |
| Categories | discrete | 291 | yes |
| Authors | discrete | 8109 | yes |
| Events | discrete | 520 | yes |
| Significant title words | discrete | 9 | yes |
| Title words | discrete | 12812 | yes |

## 3 Content Based Recommendations with Linear Regression

### 3.1 Terminology

Throughout the remainder of the paper we will use the following conventions. Lectures $L_O$ and $L_N$ will correspond to the "old" and "new" lectures, respectively. To address the attribute value of a lecture we will use a dot and the name of the attribute. For instance, $L_N.pub\_date$ is the published date of a new lecture. Greek letters $\beta$ and $\gamma$ will be used as parameters of the linear regression model. Abbreviation *Attr* will represent the set of learning attributes from Table 1, and $Train$ will be the set of all training lectures.

### 3.2 The Linear Regression Model for Predicting Rank

In this section, we will describe our linear regression model for predicting the rank of a lecture $L_N$ given that the user has already seen another lecture $L_O$. To learn such a model, we needed to estimate the true ranks of lectures in the training data and use them as the values of the class variable. Let $Rank(L_O, L_N)$ be the rank of lecture $L_N$, given

$L_O$ has been seen by the user, and let $Pairs(L_O, L_N)$ be the number of co-occurrences of $L_O$ and $L_N$ taken from the pairs data set. To compute ranks $Rank(L_O, L_N)$ for all training lectures $L_N$, we used the standard ranking algorithm:

1. For a given training lecture $L_O$, create a list of co-occurrences $Pairs(L_O, L_N)$ for all $L_N$, where $L_N \neq L_O$ and $L_N$ and $L_O$ are not from the same event.
2. Order list in ascending sequence.
3. The rank $Rank(L_O, L_N)$ is the position of $Pairs(L_O, L_N)$ in the above ordered list. The pair with the lowest $Pairs(L_O, L_{low})$ gets rank 1 and the pair with the highest value gets rank that equals the length of the list. In the case of ties, a mean rank is assigned.
4. Divide all ranks $Rank(L_O, L_N)$ by the length of the ordered list;

$$Rank(L_O, L_N) \leftarrow \frac{Rank(L_O, L_N)}{\text{Length of list}}.$$

This procedure is repeated for every training lecture $L_O$. We avoided using lectures from the same event (first step) to make learn data more alike to test data. Namely, two lectures, one from the learn set and one from the test set, always belong to different events. The second and the third step are standard steps in ranking. The normalization in the last step was applied to avoid different maximal ranks when the lengths of lists differ.

The main reason to use ranking, instead of predicting $Pairs(L_O, L_N)$ directly, was to minimize the influence of lecture $L_O$. For example, if $L_O$ is a tutorial, it will for this reason have a high number of views, and hence also $Pairs(L_O, L_N)$ will be relatively high for all $L_N$. On the other hand, the ranks depend only on $L_N$ and on similarities between $L_O$ and $L_N$, but not much on $L_O$ itself.

The linear regression model for predicting rank has the form:

$$\begin{aligned}\widehat{Rank}(L_O, L_N) = \beta_0 + \beta_{date} \times min(L_0.pub\_date, L_N.pub\_date) + \\ + \sum_{a \in Attr \setminus \{pub\_date\}} \left[ \sum_{v \in L_N.a} \beta_{a.v} \right] + \\ + \sum_{a \in Attr \setminus \{pub\_date\}} \left[ \sum_{v \in L_N.a \cap L_O.a} \gamma_{a.v} \right] \end{aligned} \quad (1)$$

The first term $\beta_0$ is the intercept and is usually close to 0.5, which is the average normalized rank of lectures. The second term contains $\beta_{date}$, the parameter that models the influence of published date. This parameter has to be positive, as lectures with low published date were published later and therefore had less chances to be viewed. The third term is the sum over all remaining attributes and values of the lecture $L_N$. Each attribute value has an assigned parameter $\beta_{a.v}$ that models the increase (or decrease) of the rank of a lecture if the lecture contains this value. The last term is again the sum over all attributes, however considering only the values that are the same for $L_O$ and $L_N$. The parameter $\gamma_{a.v}$, therefore, models the change of rank of $L_N$ if both lectures have the same value of attribute a. This parameter is usually positive, as lectures with same values are more alike, therefore it is more probable they that will be viewed together.

Such a process, where the recommendation relies only on the characteristics of the lectures, is commonly called content-based recommendation [2]. In section 4, we will describe a possible approach to extend this model to exploit some techniques from collaborative filtering.

### 3.3 The Learning Algorithm

The learning algorithm has to find such parameters ($\beta$s and $\gamma$s) to minimize the residual sum of squares. A penalty factor $\lambda$, as in ridge regression [1], was included to prevent overfitting:

$$Err = \left[ \sum_{L_O \in Train} \sum_{L_N \in Train}^{L_O.event \neq L_N.event} \left( \widehat{Rank}(L_O, L_N) - Rank(L_O, L_N) \right)^2 \right] + \\ + \lambda * \left[ \beta_{date}^2 + \sum_{a \in Attr \setminus \{pub\_date\}} \left( \sum_{v \in a} \left( \beta_{a.v}^2 + \gamma_{a.v}^2 \right) \right) \right] \quad (2)$$

Usually, the parameters of linear regression are fit easily using the formula for linear regression that involves matrix multiplication and inversion. However, considering 6983 training lectures, we have approximately $6983^2 = 48762289$ ranks to predict. Moreover, we are dealing with a large number of parameters, as adding all values from the Table 1 and multiplying by 2 sums up to 43544 parameters. The data matrix, in statistics called the design matrix, would therefore be enormous. If each multi-valued attribute would have only one value, it would still add up to 48762289*43544 values. Even with optimal coding, where each value takes only 1 byte, such matrix would still require over 2TB in memory. The data could be coded more efficiently by considering sparseness of the input (most of the values are 0), however we believe it would still present a problem for many implementations of linear regression.

Instead, we used the stochastic gradient descent algorithm. This algorithm iteratively updates the model given one learning example at a time and removes the need to have the complete data set stored. An outline of the algorithm is given in Alg. 1. The algorithm begins by setting all parameters to 0. With the third line, it commences an iteration over all attributes (including an attribute with constant value 1 to learn the $\beta_0$ parameter). In each iteration, the parameters for the values of only one attribute will be fit. The algorithm will keep optimizing parameters of one attribute until the error of the model is decreasing (5th line).

Lines 8-17 contain the core of the stochastic gradient descent algorithm. Here, the algorithm makes one sweep over the pairs of training lectures (avoiding those with the same event) and updates the values of parameters in the *updateFeatures* procedure. The update of parameters consists of the two classical steps of gradient optimization: 1) it computes the gradient of the error function (Eq. 2) on the current learning example for all relevant parameters, and 2) updates these parameters for a small negative ratio of the corresponding gradient.

The learning is governed by 4 hyperparameters of the algorithm:

- ITER: the number of passes over attributes made by the algorithm.

**Algorithm 1** Skeleton of the algorithm for fitting parameters of the linear regression model. Inputs: training data $Train$ and the hyperparameters: ITER, MP, $\lambda$, MF. Outputs: parameters $\beta$s and $\gamma$s of the linear model.

**set** values of all parameters ($\beta$s and $\gamma$s) to 0.
**for** $n = 1 \rightarrow$ ITER **do**
  **for each** $a$ **in** $Attr$ **do**
    $oldError, newError \leftarrow 1, 0$ {To satisfy the condition in while clause.}
    **while** $oldError > newError$ **do**
      $oldError \leftarrow newError$
      **call** *shuffle*($Train$) {Shuffling order of training examples is a standard step in stochastic gradient descent optimization.}
      **for each** $L_O$ **in** $Train$ **do**
        **for each** $L_N$ **in** $Train$ **do**
          **if** $L_O.event == L_N.event$ **or** $L_O$ in less than MP pairs **then**
            **continue**
          **end if**
          $newError \leftarrow newError + (predictRank(L_O, L_N) - trueRank(L_O, L_N))^2$
          **call** *updateFeatures*($L_O, L_N, a, \lambda$, MF)
          {Function *updateFeatures* computes derivatives and updates features accordingly.}
        **end for**
      **end for**
    **end while**
  **end for**
**end for**

- MP: the minimal number of pairs $Pairs(L_O, L_N)$ for a given $L_O$, where $Pairs(L_O, L_N) > 0$. A high number of non-zero pairs assures that the ranks $Rank(L_O, L_N)$ were estimated better.
- $\lambda$: the penalty factor from the error function in Equation 2.
- MF: minimal frequency of an attribute value. In order to update a parameter for an attribute value, at least MF of lectures must have this value. Otherwise, the parameter's value will remain at 0. For example, if MF=10, and since there are only 6 lectures in Russian language in the training set, the parameters related to Russian language would remain at 0.

## 4 Towards a Collaborative Approach

When the description of a lecture $L_O$ is inadequate or even wrong, the content-based approach by itself cannot provide good results. In such cases, it might be better to predict using some of the lectures that were often viewed together with $L_O$. The question is how to select these lectures? A natural choice to measure the importance of a "friendly" lecture is the number of its co-occurrences with $L_O$. A weighted sum, where weights are the co-occurrences, implements this idea:

$$Coll(L_O, L_N) = \frac{\sum_{j \in Train} Pairs(L_O, L_j) * \widehat{Rank}(L_j, L_N)}{\sum_{j \in Train} Pairs(L_O, L_j)}, \quad (3)$$

where ranks $\widehat{Rank}(L_j, L_N)$ are estimated with the linear model described in the previous section.

The above formula neglects the content-based prediction $\widehat{Rank}(L_O, L_N)$ completely. A "hybrid" approach, combining content and collaborative approaches, could result in a more accurate predictor:

$$Hybrid(L_O, L_N) = \frac{\widehat{Rank}(L_O, L_N) + \mathrm{CC} * Error(L_O) * Coll(L_O, L_N)}{1 + \mathrm{CC} * Error(L_O)}, \tag{4}$$

The term $Error(L_O)$ is the mean squared error of the content-based predictor with respect to the lecture $L_O$:

$$Error(L_O) = \frac{\sum_{L_N \in Train}^{L_N.event \neq L_O.event} \left(\widehat{Rank}(L_O, L_N) - Rank(L_O, L_N)\right)^2}{\text{Number of lectures in Train}}. \tag{5}$$

The motivation to use the error term is to give a larger weight to the collaborative predictor when the content-based predictor makes less accurate predictions. The parameter CC sets the proportion of the error to be used as a weight.

## 5 Evaluation

The submitted rankings were evaluated with the mean average R-precision score (MARp), a measure commonly used in information retrieval. Initially, we tried to produce a representative holdout set from the provided training data, which would be used to optimize the algorithm's hyperparameters. However, as we were unable to sample a set that would give similar results to those on the leaderboard, we decided to use the available number of submissions (60) to select the hyperparameters.

**Table 2.** The results of the experiments on 20% of test data (for the leaderboard) during parameter optimization. Only the relevant experiments are shown. The initial values were: MF=10, $\lambda$ = 200, MP=0, CC=0, ITER=4. The final values (in bold) are: MF=3, $\lambda$=200, MP=200, CC=3, ITER=8. Alongside the values, MARp's achieved on leaderboard are provided in brackets.

| | | | |
|---|---|---|---|
| MF | 10 (0.157) | 5 (0.162) | **3 (0.264)** |
| $\lambda$ | **200 (0.264)** | 100 (0.263) | 50 (0.262) |
| MP | 0 (0.264) | 50 (0.271) | 100 (0.288 ) **200 (0.290)** |
| CC (MP=100) | 0 (0.288) | **3 (0.290)** | 6 (0.289) |
| ITER | 4 (0.292) | **8 (0.307)** | |

Table 2 shows some of the relevant experiments. We omit several initial experiments and all experiments not leading to the final model. At the start, the values of the hyperparameters were MF=10, $\lambda$ = 200, MP=0, CC=0, and ITER=4.

We started off with larger values of MF and noticed a significant growth in performance when MF was reduced to 3. As it seems, there must be some attribute values

occurring in only three training lectures, yet are critical to make good predictions for the new lectures. The following parameter tested was $\lambda$, which appeared to have a small influence on prediction. There was a slight decline in precision when $\lambda$ was decreased, however the change was negligible.

After fixing MF and $\lambda$ we moved on to MP. We increased its value from 0 up to 200 and noticed that values over 100 clearly lead to more accurate models. As described above, with high values of MP the algorithm will learn from lectures with better estimations of ranks. In machine learning terms, we are removing examples with large noise levels in class value. Since the increase of precision was minimal when MP went from 100 to 200, we set 200 as the final value of the parameter.

Afterwards, we investigated whether the hybrid approach helps to improve the model. The CC weighs the importance of the collaborative predictor; setting it to zero will result in the use of content predictor only, high numbers will give preference to the collaborative predictor. As it turned out (to our surprise), different values of CC did not affect the performance much. The results suggest that both approaches make similarly good predictions, but their combination brings only a slight improvement. The final value of CC was set to 3.

Lastly, we explored the ITER parameter. This parameter is the number of passes made by the algorithm over all attributes. Initially, it was set to 4 and was then increased to 8. According to the obtained results, where 8 repeats lead to notably better results, it seems that our algorithm slowly converges towards the global optimum and it would be useful to let it run even longer. However, we were out of available submissions.

The final score on the leaderboard was 0.307, while the final results on the complete test set were only 0.277. Such a difference was expected as the hyperparameters of the algorithm were optimized using the leaderboard data. This was just enough to achieve the third place in competition with a small advantage over the fourth place (0.271). The first and the second place competitors scored 0.359 and 0.307, respectively.

## 6 Conclusions

In this paper, we have presented a model for predicting the ranks of lectures. It is a large-scale linear regression model that can not be fit by conventional methods, hence we used the stochastic gradient descent. We implemented content-based and collaborative-based approaches. Both approaches exhibited similar prediction accuracies with a small improvement if they were used together. The evaluation showed an interesting property of the data; there are some rare attribute values occurring in only 3 (out of 6983) training lectures that are critical for a good prediction on the test data. It would be interesting to further examine this matter and seek out which are these values. Furthermore, it turned out that it is important to have reliable class values. Removing the lectures with low number of views and learning the model from well represented lectures resulted in significantly better models.

During experiments we noticed that our stochastic gradient descent only slowly converged towards an optimum. Increasing the number of passes visibly improved the accuracy of the model and we believe that letting it run even further would result in even larger improvements. Another way of improving our model would be to consider the textual attributes, to which we did not pay much attention. Finally, it would be also useful

to construct more sensible attributes for the given domain. A possible approach would be to use argument based machine learning (ABML), where new attributes are constructed from arguments (explanations) given to some of the critical learning examples [4, 3].

## Acknowledgements

This work was partly supported by the Guru Cue d.o.o. company (http://www.gurucue.com/), and Slovene Agency for Research and Development (ARRS). We would also like to thank the organizers for putting up an interesting challenge.

## References

1. Hoerl, A.E., Kennard, R.W.: Ridge regression: Applications to nonorthogonal problems. Technometrics 12(1), 69–82 (1970)
2. Jannach, D., Zanker, M., Felfernig, A., Friedrich, G.: Recommender Systems: An Introduction. Cambridge University Press (2010)
3. Možina, M., Guid, M., Krivec, J., Sadikov, A., Bratko, I.: Fighting knowledge acquisition bottleneck with argument based machine learning. In: The 18th European Conference on Artificial Intelligence (ECAI). pp. 234–238. Patras, Greece (2008)
4. Možina, M., Žabkar, J., Bratko, I.: Argument based machine learning. Artificial Intelligence 171(10/15), 922–937 (2007)