A Hybrid Approach to Job Recommendation

Recommender Systems, Politecnico di Milano

Enrico Fini
Politecnico di Milano
Piazza Leonardo da Vinci 32
Milan, Italy 20133
enrico.fini@gmail.com

Nico Montali
Politecnico di Milano
Piazza Leonardo da Vinci 32
Milan, Italy 20133
nico.montali24@gmail.com

ABSTRACT

This paper represents a detailed description of the algorithm we developed for Politecnico di Milano's Recommender Systems Competition. The dataset we exploited for building the model is an extract of the famous RecSys 2017 dataset, provided by XING, a social network for business. The challenge we faced during the competition was to generate a set of user specific job recommendations. To achieve this objective we made use of a hybrid approach, mixing content based and collaborative filtering strategies. This technique allowed us to reach a score of 0.02+, evaluated using MAP@5 metric. Of course, besides that we needed to preprocess data and postprocess our generated recommendations to obtain such a high score.

1 INTRODUCTION

Recommender Systems have become increasingly popular in recent years, and are utilized in a variety of areas including movies, music, news and social networks. With this work we tried to apply those Systems to job recommendation, to make it easier for users to find a suitable job opportunity.
In the field of Recommender Systems, Collaborative Filtering is a technology that aims to learn user preferences and make recommendations based on user and community data. In practice it explores the social network created by User-Item interactions to highlight patterns. It is a complementary technology to Content-Based Filtering which instead uses the features of the Items or Users to define a similarity measure. Both the previously mentioned strategies have their strengths and weaknesses. For example the quality of a recommendation returned by a Content-Based Filtering strategy is strongly affected by the quality of the descriptions in the dataset. Collaborative filtering system doesn't have this shortcoming, because there is no need for a description of the items being recommended, the system can deal with any kind of information. However, on the other hand, Collaborative filtering shows problems when the dataset is not sufficiently massive or when trying to recommend a user that has no interaction so far. This is the reason why we chose to develop a Hybrid Recommender System and performance proved us right.

2 RELEASE HISTORY

Here are a list and a brief description in chronological order of all the releases we went through during the entire competition:

(1) **Top Popular**: simple algorithm returning as a recommendation for every user the most interacted items;
(2) **Item-Item Content-Based Filtering**: exploits the features of the Items (job offers) to define similarity between them. Features were weighted according to an euristic approach;
(3) **Item-Item Collaborative Filtering**: leverages users' behaviour (interactions) to define similarity between Items;
(4) **Weighted Item-Item Collaborative Filtering**: same as the previous one but similarity takes into account the number of interactions (i.e. if a user interacts more than once with an Item, it becomes more important);
(5) **User-User Content-Based + Item-Item Collaborative Filtering Hybrid**: keeping the core algorithm explained in (4) we added a Content-Based routine for inactive users[1];
(6) **User-User Content-Based + Item-Item Collaborative Filtering + Tags-Titles Similarity Hybrid**: we built an hybrid similarity which exploits the Tags and Titles features of the Item (Content-Based), for active users[2] only. This release drastically increased performance with respect to the previous ones;
(7) **User-User Content-Based + Item-Item Collaborative Filtering + Tags-Titles IDF Similarity Hybrid**: Tags and Titles features are weighted by means of IDF method. Another time, big improvement;
(8) **Hybrid + Postprocessing**: recommendations reordered using a trending measure.
(9) **Hybrid + Preprocessing (final)**: keeping the algorithm untouched, we narrowed the set of Recommendable Items[3] by means of Item specific interaction Standard Deviation[4]. This caused our model to lose Completeness but increased performance;

Moreover here under you can find Table 1 containing the MAP@5 scores and dates of our releases (referenced using their number in the previous list).

3 FINAL RELEASE

This section describes the main features of our final release (the one with the highest score).

3.1 Weighted Item-Item Collaborative Filtering

Collaborative Filtering represents a core part of the algorithm. We first implemented this method with a basic approach, using the standard Cosine Similarity for implicit rating (i.e. intersecting

[1]Users with no interaction
[2]Users with one or more interaction
[3]the set of items which can be recommended by the algorithm
[4]more information in the next chapter

Table 1: MAP@5 Score

Release Index	Date	Score
1	October 2016	0.00147
2	October 2016	0.00451
3	November 2016	0.00767
4	November 2016	0.01075
5	November 2016	0.01385
6	December 2016	0.01532
7	December 2016	0.01799
8	January 2017	0.01926
9	January 2017	0.02026

sets of users). Later in the competition we noticed we were not exploiting the full potential of the dataset, because we weren't keeping track of the repeated interactions. We started to think that if a user comes back to an item to check it again, maybe it is because the interest on that item is really high. To make the algorithm take into account that intrinsic information we decided to *weight* the score produced by candidate Items with the following rule:

$$sim_{u,i,c} = n_{u,i} * cosineSimilarity(i, c)$$

Where $s_{u,i,c}$ is the final similarity between the interacted Item i and the candidate Item c for user u and $n_{u,i}$ is the number of times user u interacted with item i.

3.2 Interaction Type Bonus

A key information in the dataset, given that we had implicit interactions by users (and not explicit ratings), was to estimate the real relevance of an item for the user. To estimate it, we used, first of all, information about the type of interaction:

- Type 1: Click
- Type 2: Bookmark
- Type 3: Application

We computed a **Bonus Score** c, that depended on the number of interactions (specific to the item and user's total), the interaction type and the timestamp of these interactions.

$$c_{u,i} = \frac{\sum_{1 \leq t \leq 3} n_{u,i,t} * w_t}{w_0 + \sum_{j \in I_u} (\sum_{1 \leq t \leq 3} n_{u,j,t} * w_t)}$$

Where the he factor $c_{i,j}$ is the Bonus for user i for item j, w_t are weights to account the different importance of interaction types, w_0 is used as a shrinking factor, $n_{i,j,t}$ is the number of interactions of a single type made by a user on a specific Item and I_u is the set of all items interacted by user u. We made many tests to understand how this Bonus Score affected the result, and we understood that it couldn't be used as an explicit rating. So, instead, we used it as an additional bonus in the Cosine Similarity.

3.3 Item Specific Interaction Standard Deviation

We implemented the pre-processing of our dataset by removing items that met some requirements from the recommendable items dataset. The first requirement for an item to be recommendable is to be *active*, so the job offer was still open at the time of the recommendation.

3.3.1 Timestamp threshold. The second requirement for items, that was checked in the majority of our releases, is expressed by the empirical concept of *"An item that has not been interacted in a while has lost its popularity"*: we simply checked if the item had, at least, one interaction in a period of time precedent to the recommendation time (that we assumed to be the timestamp of the last interaction in the dataset).

$$R = \{i \in I | \exists x \in X s.t. x.item = i \land x.ts > MAX\_TS\}$$

where R is the set of recommendable items, I is the set of items, X is the set of interactions, ts stands for timestamp. The maximum timestamp is computed as

$$MAX\_TS = max(X.ts) - MAX\_DAYS * 86400$$

We made different tests to understand the best value for MAX_DAYS, finding a value around 14 days.

3.3.2 Mean and standard variation timestamp threshold. A later implementation (used only in the latest release) tried to expand the previous concept: instead of only checking the latest interactions, we also focused on the overall interaction profile of the item. In this way, an item that was not interacted in the last days but had a very broad interaction profile (and not popular in a narrow time interval) would be still recommendable. To understand the distribution of interactions we compute the mean μ and standard variation σ of the timestamps (specific to interactions on an item). As before, we then reduced the set of recommendable items by applying a condition on μ and σ, testing again the suitable value to obtain an high score. The check required to sum the mean value to the standard variation multiplied by a tuned coefficient k. In this way, we computed a timestamp such that an approximative percentage of interactions happened before it. We have fixed again a threshold for this value, to determine if the item was to be discarded or not.

$$PIVOT\_TS_i = \mu_i + k * \sigma_i$$

$$R = \{i \in I | PIVOT\_TS_i > MAX\_TS\}$$

As before, the values k and MAX_TS were tuned to get the best score possible.

3.4 Tags-Titles IDF Similarity

At some point in the competition we felt Collaborative Filtering was not enough to provide a really focused recommendation so we started focusing on the features of the Items. Analysing them, one by one, we found out that the only two features that really increased the amount of information about the Items were:

- **Titles**: concepts that have been extracted from the job title of the job posting;
- **Tags** : concepts that have been extracted from the tags, skills or company name.

Our first implementation of what we called *Tags-Titles Similarity* was a standard Cosine Similarity between vectors of Tags/Titles. However later in the competition we discovered that not all the Tags and the Titles were equally important. We had to find a way to weight them. In the end it turned out that using a simple IDF approach could strongly boost the performance of the model. In formulas:

$$w_t = \log \frac{n_{tot}}{n_t}$$

Where w_t is the final weight of the Tag/Title, n_{tot} is the total count of Tags/Titles and n_t is the count of the specific Tag/Title t. Also, we had to integrate this new weighting method inside our code in some way. We decided to create a new Cosine-like Similarity, called **Modern IDF Similarity**:

$$sim(i,j) = \frac{\sum_{t \in (T_i \cap T_j)} w_t}{h_0 + \sqrt{|T_i| * |T_j|}}$$

Where T_i and T_j are the sets of Tags/Titles belonging to items i and j respectively, w_t is the weight of a particular Tag/Title t, h_0 is the Shrinking Factor and $|T_i|$ and $|T_j|$ are the Cardinalities of sets T_i and T_j. Finally, to create a hybrid similarity we had to combine it with the *Cosine Similarity* for Collaborative Filtering. The final result, after a little bit of tuning, was a significant increase in performance.

3.5 Content-Based User-User Similarity

In the introduction of this paper we presented some problems you can face while implementing a Collaborative Filtering based algorithm. It turned out that this dataset was not massive enough to avoid that kind of issues. In fact while implementing our version of Collaborative Filtering we discovered that some users were *inactive*. In simple terms this means that they had no interaction with any Item. Hence we decided to exploit a *K-Nearest-Neighbor* Content-Based approach, since it was not possible to build a social network starting from the user's interactions. To do that it was necessary to weight every feature with a Heuristic Value. Building this Heuristic has been a really intricate task because the risk of overfitting was significant. In the end we come out with a quite balanced weighting for the features. Using these Heuristic Values we were able to highlight, given a specific user, all the users that were, in some way, similar to it. Keeping the scores obtained from the previous computation, the *similar users* vote, ranking the items from the most suitable to the irrelevant ones. This approach turned out to increase performance but, in general, to be really unstable and not always reliable.

3.6 Post-processing

At the final stages of the Competition we knew that the recommendations our algorithm was returning were quite accurate and that a further increase in performance wouldn't have been so easy. Hence we decided to keep the core algorithm untouched and to work on presenting the recommended items in the most suitable order. Doing this we discovered that this dataset was, in some way, affected by a *Trend Bias*. This means that the items which were mostly interacted in the last period of time (with respect to the minimum and maximum interaction timestamp) where much more likely to be good recommendations, and, hence, to increase performance. This is the reason why we decided to insert a reorder routine at the end of the computation. In formulas the new ranking is computed as below:

$$r_i = K * (N - pos_i) + n_i$$

Where K is the weighting of the hybrid ranking, N is the number of items to reorder (around 10 was a good choice), pos_i is the position of Item i in the hybrid ranking and n_i is the number of interactions on a specific Item i in the last days (4/5 days turned out to be suitable).

3.7 Conclusions

After many different approaches, we found out that Collaborative Filtering was very effective on the dataset, since a pure content-based approach was very unstable on our data. We selected tags and titles after an analysis of the dataset, considering them a very relevant factor in item classification: from a user point of view, the information in the title are the ones that make him click on it. So our solution was to boost a pure collaborative filtering (when possible) with tags and titles additional data.
We experienced a great performance boost using the pre-processing and post-processing of data, when correctly tuned. The reason we applied these methods, and so abandoning completeness, is that the dataset was strongly time-dependent: this is probably caused by how Xing works and present items to the user.