A Scalable, High-performance Algorithm for Hybrid Job Recommendations

Toon De Pessemier
iMinds - Ghent University
Technologiepark 15
B-9052 Ghent, Belgium
toon.depessemier@ugent.be

Kris Vanhecke
iMinds - Ghent University
Technologiepark 15
B-9052 Ghent, Belgium
kris.vanhecke@ugent.be

Luc Martens
iMinds - Ghent University
Technologiepark 15
B-9052 Ghent, Belgium
luc.martens@intec.ugent.be

ABSTRACT

Recommender systems can be used as a tool to assist people in finding a job. However, this specific domain requires expert algorithms with domain knowledge to recommend jobs conformable to people's expertise and interests. This is the topic of the Recsys Challenge 2016, which aims for an algorithm that predicts the job postings that a user will positively interact with. Our solution is a hybrid algorithm combining a content-based and KNN approach. The content-based algorithm matches features of candidate recommendations and job postings of historical interactions. The KNN approach searches for the job postings that are the most similar to the postings the user interacted with in the past. The resulting combination is a lightweight algorithm that is fast and scalable, generating recommendations with a proper evaluation score.

CCS Concepts

•Information systems → Recommender systems;

Keywords

content-based, KNN, light-weight, scalable, high-performance

1. INTRODUCTION

Recommender systems are software tools and techniques providing suggestions for items to be of interest to a user such as videos, songs, or news articles. Driven by this success, more application domains have adopted recommender systems to reduce the information overload by generating personalized suggestions. Also for recruitment scenarios, in which applicants search for suitable job offers, recommender systems are a useful tool for job candidates, recruiters, as well as platforms that connect both. Specific characteristics of the domain require special attention for job recommendations. Given the typical features of a job (such as discipline or industry), content-based recommender systems are often used to match user profile and job description [4]. Job specific restrictions (such as education level or location) can be an obstacle for job applicants. And job offers typically have a limited lifetime, unlike videos, books, or songs. As soon as the vacancy is filled, the job offer will be unavailable and removed from the platform. Despite these metadata dependencies, the use of machine learning algorithms can further improve the results obtained with CB similarity measures.

Job recommendation is also the topic of the RecSys Challenge 2016 [2], which is co-organized by XING. XING is a social network for business. XING assists applicants in the search for a job and helps recruiters to find the right candidate for a job. The aim of the job recommendation system of XING is to predict the job postings that are likely to be relevant to the user. This is also the viewpoint of the challenge. Though "being relevant to the user" is very difficult to measure in an offline setting, i.e. based on historical data without any engagement or interaction of the user. Therefore, evaluation is performed through an evaluation metric that reflects the typical XING use cases and that is based on precision, recall, and user success (i.e. whether at least one relevant item is recommended for a given user). For calculating the recommendations, various data sources can be used such as a user profile (with personal information such as university degree, home region, years of work experience, etc.), a list of impressions (i.e. items which were shown to the user by the existing XING job recommender), a list of interactions that the user performed on the job postings (clicking, replying, bookmarking, or deleting a recommended item), and details about the job postings that should be recommended to the user (such as discipline, industry, tags, etc.). For specific details regarding the evaluation and the data, we refer to the GitHub page of the challenge [1].

In the context of this challenge, we developed a hybrid recommendation algorithm for jobs. Since complex and computationally intensive algorithms sometimes require excessive engineering costs to be applied in a practical case [3], we focused on a fast and lightweight algorithm with a reasonable evaluation score.

Besides offering relevant recommandations to the user, we formulate the following goals for the hybrid recommendation algorithm of this paper:

- Scalable in terms of the number of users and jobs in the system. Given the growing importance of digital communication tools, social networking services offering job postings have the potential to grow considerably in the next decade. Therefore, the algorithm

RecSys Challenge '16, September 15 2016, Boston, MA, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4801-0/16/09…$15.00
DOI: http://dx.doi.org/10.1145/2987538.2987539

should still be usable if the number of users and items increases rapidly.

- Enabling incremental updates of the model. In an offline test case, such as the RecSys Challenge, all historical data is available at the start of the calculation process. In contrast in a real-life case, new users, job postings, and user interactions with these job postings are continuously added. Since calculating recommendation models can be a computationally intensive process, recomputing the models can be an inefficient use of the computational resources. Updating the existing models with additional data at a small computational cost is a desirable feature of the algorithm.

- Fast score calculation for new job postings. When new job postings are inserted into the system, suitable applicants want a recommendation for these job items as soon as possible. Also for recruiters, a fast distribution of their job postings is important. Algorithms that require hours or even days to generate recommendations for new items introduce an undesirable slowdown in the information chain.

2. FINDINGS

2.1 The challenge is a prediction task.

In a real-life case, recommendations are personal suggestions for the user. This means that the recommendations are offered to the user who can consider the content and decide if the recommendation is useful or not. Usefulness is an important quality metric from the user perspective [5]. Moreover, the user's interaction behavior with the content is influenced by observing the recommendations. In contrast, the goal of the RecSys Challenge is to predict the interaction behavior of the users without these recommendations. In this offline evaluation, the recommendations of the algorithm are not offered to the user and a personal evaluation of the usefulness is not possible. The difference in evaluation methodology between online and offline cases is especially noticeable for surprising or serendipitous items, i.e. job items that are difficult to find but interesting for the user. A significant probability exists that the user will interact with recommendations for these jobs. So, it is a good recommendation in an online case. However in the offline case, the user will not find the item without recommendation, and there will be no interactions. Since the recommendations are evaluated based on the interactions, the serendipitous items are unfairly evaluated as "poor recommendations".

2.2 The information value of impressions is limited.

Impressions are defined as a subset of the items that were shown by the existing job recommender. Since not all recommended items are available as impression, and there is no guarantee that the item was in the viewpoint of the user, we consider an impression as a weak expression of interest for the item. Since these impressions are originating from another recommmender system, any recommendations based on these impressions might be biased. Therefore, our algorithm uses only the impressions of users who have fewer interactions than a cold-start threshold. For the other users, the impressions are neglected without decreasing the score of the recommendations. Reducing the dataset by neglecting part of the impressions reduces the complexity of the recommender and offers an advantage in terms of computation load.

2.3 Items with a limited visibility get a penalty.

The goal of the challenge is to predict which items a user will positively interact with within the next week. Quality metrics such as serendipity and novelty are not taken into account. Since job items with a low visibility have a low probability of interaction, these job items get a penalty in our algorithm. The visibility of a job item is estimated by the number of interactions in the dataset. Although this penalty favors the popular items, and is not always desirable in a real-world recommender, it is used to increase the evaluation score in this challenge.

2.4 The influence of the user's region should not be overestimated.

One might expect that the user's interest for jobs is focused on job items located in the region of the user, or in an adjacent region. However, in a considerable number of cases, a user interacts with a job item located in an non-adjacent or far away region with respect to the home region of the user. E.g., although Lower Saxony and Baden-Württemberg are not adjacent, there is a considerable interest of inhabitants from the former region for jobs in the latter region.

2.5 Traditional classification does not work.

User interactions are characterized by a type: click, bookmark, reply, or delete, which can be used to partition the interactions into distinct classes. Based on the historical interactions of a user, a model can be constructed to classify the other job postings into one of the three positive classes (click, bookmark or reply) or the negative class (delete). From the classification of job items, recommendations can be inferred by selecting the items that are most characteristic for one of the positive interaction classes with an active user engagement (bookmark or reply). However for the use case of the RecSys Challenge, recommendation based on classification of the job items resulted in a poor score. The reason might be the underlying meaning of the user interactions. Interactions of the type 'delete' are originating from delete actions of the users on their list of recommendations. These 'deletes' are performed by clicking on the "X" button. This action has the effect that the recommendation will no longer be shown to the user and that a new recommendation item will be loaded and displayed to the user. The way in which interactions of the type 'delete' are gathered does not necessarily imply a disinterest of the user or a negative feeling towards the job item. Users might have a neutral feeling towards the job, without the intention to interact. Besides, users might click on the "X" to receive a new recommendation without having an aversion for the item or even without thoroughly inspecting the item. Moreover, the delete interactions are not sampled from the set of available job items, but are interactions on the user's personal recommendations. Therefore these delete interactions on job items are biased. These items might be more similar to the user's interests than items covering the whole range of job postings. As a result, for items outside the user's recommendations, negative evaluations are missing.

2.6 The data contains many cold start users.

The dataset contains 150,000 users for whom recommendations have to be calculated. However for 39,755 of these users, no interactions with job items are registered in the data set. Impressions can be used to enrich the profile of 25,238 of these cold start users. As a result, for 14,517 users or 9.7% no information about interactions or impressions is available. So, a fallback solution in case of none or insufficient interactions and impressions is very important. For these users, only some basic profile information might be available, without links to specific job items. As a result, collaborative filtering approaches cannot handle these users. But also content-based techniques can experience difficulties if interaction and impression data is missing. If profile data is rather general (e.g., by specifying a very broad discipline or industry), too many job items might be considered appropriate, all ending in a tie. If the profile data is too specific (e.g., in case of a very niche discipline or industry), the number of matching job items can be very limited, resulting in too few recommendations. Moreover, job applicants do not always update their user profile in a timely manner [6], making it quickly obsolete without dynamic updates based on interactions or impressions.

3. HYBRID JOB RECOMMENDATIONS

The recommendation algorithm consists of two approaches, a content-based and a KNN approach, which are combined using a weighted average, depending on the number of interactions of the user. The weight of the KNN approach increases as more interaction data of the user is available.

3.1 Content-Based

The base of the content-based approach is a user profile with the explicitly specified user information. The profile can be updated offline using historical interaction data of the user. For each interaction that the user performed on the job postings, a set of *positive counters* is updated. For each metadata attribute of the item (title, career level, discipline, industry, country, region, location, employment, and tags) the associated counter is incremented. If an attribute of the item contains the IDs of multiple features, multiple associated counters are incremented. This is often the case for the fields title, discipline, industry, and tags. The increment value of the counter is determined by the interaction type. We assume that bookmarking an item or the intention to reply on a job posting is a much stronger expression of interest than a click on an item. In our implementation, we used the increment values 0 (delete), 1 (click), 10 (bookmark) and 10 (reply). Since no significant effect on the score was witnessed for different positive or negative values of delete events, the value of 0 was used. For clicks, bookmarks and replies, the weight of the interaction type is significantly influencing the score with differences up to 5% compared to the unweighted case.

Besides the positive counters, the profile also contains a set of negative counters, which represent the user's disinterest in a specific metadata feature. For each interaction, IDs of features that are not associated with the item get a penalty by increasing the negative counters. Again, the increment value of the counter is determined by the interaction type. The reasoning behind the combination of positive and negative counters is as follows. The user's preferences are characterized by the number of times the user selects a job item with a set of specific features. But also the number of times the user selects a job item without a specific feature is important to characterize the missing link between that feature and the user.

The output of the content-based approach is a recommendation score for each user-item pair $s(u, i)$. This recommendation score is a linear combination of the differences between the positive ($pos_{f,u}$) and negative ($neg_{f,u}$) feature-based profile counters. The parameter α (0.5 in our implementation) represents the relative importance of the positive and negative profile counters. These differences are multiplied by the logarithm of the ratio of the total number of items N and the number of items characterized by the specific feature n_f. This logarithm is the analog of the inverse document frequency (IDF) used for ranking documents' relevance. The IDF compensates for the frequency of a feature across all job postings in the system. This way, rare tags or disciplines get a higher weight than common titles or industries. The relative influence of various metadata features (title, country, career level, etc.) can be reflected by a weight w_f. The denominator stands for the number of features that characterize the item and compensates for a different number of features for various job items.

$$s(u,i) = \frac{1}{|\{f \in i\}|} \sum_{f \in i} w_f \cdot (pos_{f,u} - \alpha \cdot neg_{f,u}) \cdot log(\frac{N}{n_f}) \quad (1)$$

This light-weight content-based recommender was designed keeping in mind the scalability and high-performance on big data sets. To speed up the computations, various factors of equation 1 can be precomputed, before the candidate job items are available. The logarithm (IDF) is rather stable over time and can be calculated offline. The counters are depending only on the historical interactions of the user and are calculated offline as well. If new information becomes available, such as additional interactions, the counters can be incremented to update the user model. As a result, the online calculation load is limited when a new job item is introduced into the system. Calculating recommendations for a new user-item pair is extremely fast. Moreover, the calculations of the recommendations for all users can run in parallel and scale linearly in terms of the number of users and job items in the system.

3.2 KNN

As counterpart of the content-based recommender, a k-nearest neighbor (KNN) algorithm is used. In this KNN approach, the search space consists of all job items that the user positively interacted with (click, bookmark, or reply). In contrast to traditional NN solutions, in which only interactions are used for calculating the distance, our implementation uses a combination of features (two items are similar if they share the same metadata features) and interactions (two items are similar if users have interacted with both of them). The dimensions of the search space are the features of the item (such as discipline, industry, country, tags, etc.) as well as the IDs of the users who interacted with the job item. Metadata attributes of items that are specified as a comma-separated list of various terms, are expanded into multiple nominal features; one for every option. The various dimensions are normalized to compensate for scale differences. This results in a fine-grained distance function for the items in which differences in interactions as well as metadata are taken into account and the risk of ties is reduced.

To calculate the distance $Dist(i,k)$ between an active item i and neighbor k over all dimensions, the Euclidean metric is used with a weight for each dimension. To take into account the relative frequency of the various features, the distance in one dimension (i.e. the distance due to a single metadata feature f) is multiplied with the IDF: $log(\frac{N}{n_f})$ as a weight.

For each active item, i.e. each candidate recommendation, the k-nearest neighbors are selected in the search space. In other words, for each active item the algorithm selects the most similar items that the user interacted with in the past. The reasoning is a follows. If a candidate item is very similar to the items that the user interacted with in the past, it will probably be an interesting item to recommend. To obtain the recommendation score, the distances to these neighbors are calculated. These distances are converted into relative proximities. The relative proximity of a candidate item to the user's interactions is defined as the difference between the maximum distance and the item's distance to the nearest job item that the user interacted with. To normalize this difference, it is divided by the maximum distance. The recommendation score $s(u,i)$ is than calculated as the average of the relative proximity to the k-nearest neighbors.

$$s(u,i) = \frac{1}{k}\sum_{k}\frac{Dist_{max} - Dist(i,k)}{Dist_{max}} \tag{2}$$

Our KNN approach is built on top of the Weka Framework [7]. More specifically, the BallTree implementation of the NearestNeighbourSearch package is used. The value of k is determined as a linear function of the number of interactions the user has performed on job items.

Also for the KNN approach, the focus was on scalability and high-performance. As common in collaborative filtering implementations, the item similarities (or distances) can be calculated offline and stored for future recommendation calculations. If item distances are precomputed, calculating the recommendation score consists of a fast look-up operation of item-item distances and aggregating the distances of item-item pairs through averaging. Similar to the content-based calculations, the KNN calculations are parallelized in our implementation. The distance calculation provides additional opportunities to speed-up the recommendation process. E.g., if the partially computed distance exceeds a threshold that precludes the item from being one of the k-nearest neighbors, calculations can stop early.

3.3 Results and Fallback

Using the evaluation measure of the challenge, the pure content-based approach obtains a score of 286041.10. The NN recommender scored 298316.85 for all target users. With the hybrid combination of both algorithms, we obtained a score of 344264.37. If the KNN algorithm is unable to generate recommendations (e.g. for cold start users), the hybrid recommender is replaced by the content-based (26.5% of the users have no interactions). For users without explicit profile, the hybrid recommender can be replaced by the KNN algorithm (but this was not necessary in the challenge). If the algorithms fail to generate recommendations based on interactions, the impressions of the user are used to generate recommendations (16.8% of the users). A solution without fallback to impressions, only based on the explicit profile for cold-start users, scored 292909.26.

If the combination of the user's explicit profile, interactions, and impressions is still insufficient information for generating recommendations, the system falls back on recommending the most popular items. For 1485 of the target users, the 30 most popular items are recommended as a fallback mechanism. The most popular items are defined as those items that received the most positive interactions in the data set. The hybrid solution with fallback to impressions but without fallback to popular items scored 344241.51. Recommending the most popular items for all users resulted in a score of 73298.13.

4. CONCLUSIONS

Recommendation challenges are a playground for mathematicians and statisticians to design complex and exotic algorithms. However, these algorithms often come with the risks of extreme hardware demands (e.g. computational and memory requirements) and/or scaling problems. Therefore, we proposed a light-weight solution that is fast, scalable, and allows incremental data updates. Hybrid recommendations combining a content-based and KNN approach can handle the user and item metadata, interactions and impressions, and cold-start users for whom this data is missing. The resulting recommendations have shown to be well above average in terms of the evaluation score. For future work, we see some opportunities for improvement, such as the hybridization of the algorithms and the optimization of the feature weights.

5. REFERENCES

[1] F. Abel, D. Kohlsdorf, and R. Pálovics. Training Data of the RecSys Challenge 2016, 2016. Online available at https://github.com/recsyschallenge/2016/blob/master/TrainingDataset.md.

[2] ACM - Xing. RecSys Challenge 2016, 2016. Online available at https://recsys.xing.com/.

[3] C. Johnston. Netflix Never Used Its $1 Million Algorithm Due Do Engineering Costs, 2016. Online available at http://www.wired.com/2012/04/netflix-prize-costs/.

[4] M. Diaby, E. Viennet, and T. Launay. Toward the next generation of recruitment tools: An online social network-based job recommender system. In *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*, pages 821–828, Aug 2013.

[5] S. Dooms, T. De Pessemier, and L. Martens. A user-centric evaluation of recommender algorithms for an event recommendation system. In *Proceedings of the RecSys 2011 : Workshop on Human Decision Making in Recommender Systems (Decisions@RecSys'11) and User-Centric Evaluation of Recommender Systems and Their Interfaces - 2 (UCERSTI 2)*, pages 67–73, 2011.

[6] W. Hong, S. Zheng, and H. Wang. Dynamic user profile-based job recommender system. In *Computer Science Education (ICCSE), 2013 8th International Conference on*, pages 1499–1503, April 2013.

[7] T. C. Smith and E. Frank. *Statistical Genomics: Methods and Protocols*, chapter Introducing Machine Learning Concepts with WEKA, pages 353–378. Springer, New York, NY, 2016.