A Light-Weight Approach to Recipient Determination When Recommending New Items

Malte Ludewig
TU Dortmund
Germany
malte.ludewig@tu-dortmund.de

Michael Jugovac
TU Dortmund
Germany
michael.jugovac@tu-dortmund.de

Dietmar Jannach
TU Dortmund
Germany
dietmar.jannach@tu-dortmund.de

ABSTRACT

Recommending new items can be a challenging problem in practical applications. Since no past user interactions (e.g., ratings) for such items are available, collaborative filtering techniques cannot be directly applied. Even if item feature information and a content-based approach are used as a fallback, the new items might not show up in the recommendation lists of many users. A possible approach in such situations is to determine a limited set of users to whom the new items are recommended even if there are other, longer existing items that might have a higher assumed relevance.

In this paper we present a light-weight approach to this *recipient determination* problem in the context of the recommendation of job offers on a business social network. At its core, the method uses a nearest-neighbor technique and a set of additional domain-specific heuristics to assess the relevance of new job offers for a user. The evaluation in the context of the ACM RecSys 2017 challenge showed that the method, despite its simplicity, led to good results both in the offline and the online setting.

CCS CONCEPTS

•Information systems →Recommender systems; Nearest-neighbor search; Content analysis and feature selection;

KEYWORDS

Item Cold Start, Recipient Determination, Job Recommendation

ACM Reference format:
Malte Ludewig, Michael Jugovac, and Dietmar Jannach. 2017. A Light-Weight Approach to Recipient Determination
When Recommending New Items. In *Proceedings of RecSys Challenge '17, August 27, 2017, Como, Italy, ,* 6 pages.
DOI: http://dx.doi.org/10.1145/3124791.3124795

1 INTRODUCTION

Automatically provided and often personalized item recommendations are an important feature on many of today's websites, in mobile applications, and even on smart televisions. These suggestions support customers in discovering items that they might like, and can simultaneously help businesses to boost their revenue, e.g., by increasing the customer satisfaction [1, 4–7, 10].

To provide useful recommendations, many of the commonly used approaches from the recommender systems literature apply collaborative filtering techniques, which require a substantial amount of interaction data between users and items, e.g. ratings or click logs. Such data is, however, not always available for the whole user/item base, e.g., when a user visits a web shop the first time (user cold-start) or a new movie is added to a video platform's catalog (item cold-start).

Cold-start scenarios like these are common in practice. In particular for item cold-start situations, one possible approach is to resort to feature information and "content-based" approaches to determine the suitability of a new item for a known user. However, when applying such a strategy, the problem can arise that the new item does not make it into the recommendation lists of many users. On a movie streaming platform, for example, there might be many other items that are already available longer and considered a better match for the users.

One strategy in such situations is to determine a limited set of key *recipients* to which the new item is recommended even if there are other items that might be more relevant. The algorithmic task is then to identify these key recipients for which the new items are a reasonable match and/or recipients of which we assume that they will react to the recommendation. The feedback of these recipients can then be used in an explore-exploit scheme. Generally, while one goal of such a strategy can be to collect some initial feedback, there are also scenarios where the recommendations are paid for and the recommendation service provider is obliged to present the new items to a minimum number of users.

In this paper, we propose a light-weight approach to determine key recipients in such scenarios. We consider the method to be light-weight because it does not rely on complex machine learning models but implements a nearest-neighbor strategy and a number of domain specific heuristics.[1] Roughly speaking, given a new item, the main idea of our method is to identify other users who (positively) interacted with similar items in the past.

The method was evaluated in the context of the 2017 ACM RecSys Challenge[2], where the problem was to determine a set of recipients for newly posted job offers on the German business social network XING[3]. Despite its simplicity, the method has shown to lead to good results in the competition, both in the offline evaluation (fifth place) and in the online evaluation (sixth place).

[1]Previous research has shown that in some domains nearest-neighbor methods can be comparably effective despite their simplicity [8, 9, 11, 15, 16].
[2]http://2017.recsyschallenge.com/
[3]https://www.xing.com/

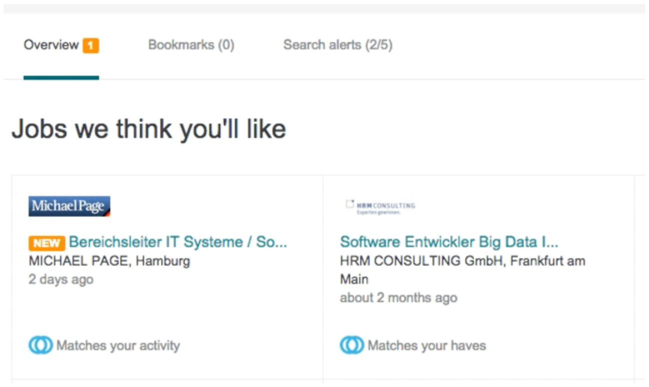

Figure 1: Screenshot of a recommendation presented on XING's job search page

2 PROBLEM DESCRIPTION AND SETTING

In this section, we define the task, the recommendation scenario and the given data in more detail [3].

Task. Given a new job posting *p* that was just now submitted to the website, the goal is to identify those users that

a) may be interested in receiving the job posting as a push recommendation and
b) are also appropriate candidates for the given job.[4]

Generally, the specific problem setting in this challenge is thus different from the job recommendation scenarios and approaches described in the literature, in which interaction data is often provided for both users and jobs [2, 12–14, 17, 18].

Once a set of suitable users is determined, the recommendations are distributed over multiple channels. One of these distribution channels is shown in Figure 1, where an entry is added to the user's job search page. Also, recommendations are actively pushed to users by e-mail and to recruiters through XING's online tools. The specific challenges of this scenario can be summarized as follows.

- **Cold-start.** As the job offers are new on the platform, some commonly used recommendation techniques cannot be applied.
- **Push notifications.** The job suggestions are automatically sent to the user, which raises the issue of when and how often users like to receive push notifications. Hence, not many recommendations can be distributed to a single user and the system needs to choose carefully which job is the most suitable for a user.
- **Trade-Offs.** As not only users are provided with job recommendations, but also recruiters receive suggestions of these users as potential candidates, a proper balance is necessary between these two interest groups. Furthermore, companies can pay for the promotion of job postings, which creates another problem of balancing item relevance and revenue.

The latter two balancing problems are not necessarily a result of the given recommendation scenario, but chosen by the organizers to make the competition more realistic by including these trade-offs in the evaluation metric (see Section 2.2.4).

[4]Official task description from http://2017.recsyschallenge.com/

Table 1: Content features provided for all items

Feature	Brief description
Id*	Unique identifier
Title*	A list of IDs referring to concepts extracted from the job title
Tags	A list of IDs referring to tags extracted from the job's content information
Career level*	A discrete scale from 1-6
Discipline*	ID representing the field of the job, for example, Human Resources
Industry*	ID representing the industry, e.g., Automotive, Finance, or Internet
Employment	1 of 5 types of employment like, e.g., full-time, freelancer, or intern
Country*	String to identify the country (only Germany, Austria, and Switzerland)
Region*	ID to determine the region (only for Germany)
Latitude	Latitude rounded to about 10 km
Longitude	Longitude rounded to about 10 km
Paid	Is this job offer paid for by a company?
Created at	Timestamp of the creation date

2.1 Data

In the given scenario, we were provided with a multitude of "content" features for both users and items as well as the recent user interaction log data of about 3 month. These log entries include unique identifiers for users as well as items, a precise timestamp, and an interaction type, which is one of the following:

- **Impression.** The most frequent interaction type, which indicates that the user was shown this job offer as a recommendation.
- **Click.** This event indicates that the user clicked on the job offer to visit its detail page.
- **Bookmark.** Within the notification and on the detail page, a user can bookmark a job offer to inspect it more closely later on.
- **Reply.** As XING is a business social network, users can directly apply for a job. In this case a "reply" event is recorded.
- **Recruiter.** This event is not initiated by the user, but instead indicates that a recruiter of a company that offers a job is interested in the user.
- **Delete.** The user actively removed the recommendation from the list/feed.

As mentioned above, a wide range of content features is available for each item. Table 1 gives an overview of these features, which we found important in the recommendation scenario, since for the cold-start items no other information is known. In addition to the item content information, we also have access to user content information based on the users' profiles. In Table 1, entries marked with a star appear both in the items' content information as well as the users' profile information.[5]

[5]The user profiles also contain some unique features that are not available for items, which are omitted here for space reasons and because they were not used in our approach.

2.2 Evaluation Procedure

The evaluation of the challenge was split into two parts: an initial offline evaluation and a real-world test that was executed afterwards on the production website.

2.2.1 Offline Evaluation. In the first part of the challenge, the participants were given recent interaction logs recorded over a time span of 3 month as well as a set of items and users with the associated content information. In addition, the teams received a set of about 47,000 target items, for which only content data but no interaction data was available. For each of these target item, a list of up to 100 matching users should be determined and uploaded to the competition website. The submitted solution was automatically evaluated using the metric, which will be explained in Section 2.2.4, resulting in a ranking of the teams. Some statistics about the data used at this stage are shown the first column of Table 2.

2.2.2 Local Evaluation. In order to optimize our approach without uploads, we created our own, local evaluation environment and used 45,000 items from the last two days of the interaction log as the target items. From the remaining interactions, we created our training set by removing all actions related to our test items to ensure the absolute cold-start scenario. Table 2 shows some statistics about this training set (called "Local").

2.2.3 Online Evaluation. The second part, the online evaluation, substantially differs from the previously described evaluation procedures except for the metric, which remained unchanged. The 20 best performing participants from the offline evaluation were invited to provide recommendations for the real-world system on a daily basis. Given a set of target users and some initial interaction data, whose statistics are shown in Table 2, we received a daily list of newly posted job offers and the corresponding content information as well as the interaction logs from the previous day. We had to rebuild the recommendation model within 24 hours and upload a solution file that matches the known users with the new jobs. As the recommendations were actually pushed to real users, the online evaluation heavily differs in the fact that each user could only be associated with one single item, i.e., each user could only receive one recommendation per day. Overall, the online challenge was conducted over a period of 5 weeks. The scores were determined per week, and the team with the best two weekly scores was considered the winner of the challenge.

2.2.4 Evaluation Metric. For both evaluation stages, the same metric was applied. For each item i and the recommended users U the score is determined as:

$$score(i,U) = \sum_{u \in U} uscore(i,u) + iscore(i,U) \tag{1}$$

Within Equation 1, *uscore* is defined as follows:

$$\begin{aligned} uscore(u,i) = (&1_{click}(u,i) \\ &+ 5 \cdot 1_{bookmark}(u,i) \\ &+ 5 \cdot 1_{reply}(u,i) \\ &+ 20 \cdot 1_{recruiter}(u,i) \\ &- 10 \cdot 1_{delete}(u,i)) \\ &\cdot 2_{premium}(u) \end{aligned} \tag{2}$$

Table 2: Characteristics of the data provided in the different evaluation phases

Characteristic	Offline	Local	Online
Users	1.2M	1.1M	468k
Items	614k	567k	692k
Interactions	322.8M	212.9M	93.9M
Impression	97.44 %	97.25 %	94.50 %
Click	2.13 %	2.27 %	3.78 %
Bookmark	0.09 %	0.11 %	0.20 %
Reply	0.04 %	0.04 %	0.10 %
Delete	0.28 %	0.31 %	0.38 %
Recruiter	0.03 %	0.02 %	0.04 %
Time span in days	93	91	85

And, the item score *iscore* is calculated with the function below.

$$iscore(i,U) = \begin{cases} 25 \cdot 2_{paid}(i) & \exists\, u \in U : uscore(u,i) > 0 \\ 0 & \forall u \in U : uscore(u,i) \leq 0 \end{cases} \tag{3}$$

In these formulas, $1_a(u,i)$ returns 1 if user u and item i are related with action a (e.g., bookmark) and 0 otherwise. On the other hand, $2_f(o)$ equals 2 if user or item o has feature f and 1 otherwise.

The metric takes into account both the users' as well as the recruiters' interest and weights the promotion of paid content and premium users higher. This should reflect the balance between different interest groups as well as between relevance and revenue.

3 TECHNICAL APPROACH

Given the task of the competition, the cold-start situation, and the available data, we considered an item-based nearest-neighbor approach as a straightforward approach to the problem. For a new job posting, our method finds users that were interested in similar items in the past. To calculate similarities between new and old jobs, we use the items' content features. In the following, we will describe the design of our approach in detail.

3.1 Basic Item-kNN

The scoring function of our approach, which determines the match between a user u and a job i, is the following:

$$conf(u,i) = \frac{1}{\sum_{i_n \in top_k(i)} 1_I(u,i_n)} \sum_{i_n \in top_k(i)} sim_{cb}(i,i_n) \cdot w_I(u,i_n) \tag{4}$$

The confidence of a user-job tuple is determined by adding up the similarity values of the k most similar neighboring job postings to job i ($top_k(i)$) multiplied with a weighting factor $w_I(u,i_n)$ and normalized in the end. In this function, $1_I(u,i_n)$ returns 1 if the user u interacted with job i_n and 0 otherwise. Furthermore, the weighting function is defined as follows:

Table 3: Similarity function used for each item feature

Feature	Similarity function
Title*	Jaccard index
Tags	Jaccard index
Career level*	Equality
Discipline*	Equality
Industry*	Equality
Employment	Equality
Country*	Equality
Region*	Equality
Latitude & longitude	Haversine distance

$$w_I(u, i_n) = \begin{cases} 0 & \text{No interaction of user } u \text{ with job } i_n \\ w_{click} & \text{User } u \text{ clicked job } i_n \\ w_{bookmark} & \text{User } u \text{ bookmarked job } i_n \\ w_{reply} & \text{User } u \text{ replied to job } i_n \\ w_{recruit} & \text{A recruiter found user } u \text{ for job } i_n \\ w_{delete} & \text{User } u \text{ deleted job } i_n \end{cases} \tag{5}$$

Finally, the content-based similarity sim_{cb} between two jobs with a set of content features F_I is defined as:

$$sim_{cb}(i_1, i_2) = \frac{1}{\sum_{f_i \in F_I} w_{f_i}} \sum_{f_i \in F_I} sim_{f_i}(i_1, i_2) \cdot w_{f_i} \tag{6}$$

Roughly speaking, the overall job similarity is a weighted sum of feature similarities. Table 3 indicates how the individual content features of the job offers are compared. Using the Jaccard index and exact equality checks were a natural choice, even though more elaborate schemes are possible. Regarding the geolocation, the Haversine function is applied to calculate the actual distance in steps of 10 km and the similarity is determined as the division of 1 by the number of steps.

The number of neighbors k and all weight factors were optimized with respect to the metric described in Section 2.2.4. This procedure is described in more detail in Section 3.4.

3.2 Additional Heuristics

To further optimize the performance, we integrated a number of additional heuristics into our basic Item-kNN technique.

Co-occurring feature values. We found in the data that for certain features some values frequently co-occur in the users' click histories, e.g., for the industry, the discipline, and the region. Two regions, for example, might not have the same ID, but are adjacent. Thus, they appear more often in a user's log together and should be assessed as more similar than a region that is far away. To account for these observations, we modified the similarity function for these features to factor in the co-occurrence ratio:

$$sim_{f_i}(i_1, i_2) = 1_{same_{f_1}}(i_1, i_2) + (1 - 1_{same_{f_1}}(i_1, i_2)) \cdot CC_{f_i}(i_1, i_2)^{\frac{1}{b_{cc}}} \tag{7}$$

Within the above formula $1_{same_{f_1}}$ equals 1 for a matching feature, 0 otherwise, and CC_{f_i} returns the co-occurrence ratio for feature f_i of item i_1 and item i_2. Hence, if the two features do not directly match, their co-occurrence ratio will be used instead, softened or amplified by a boosting parameter b_{cc}, respectively.

Minimum similarity and confidence. With an increasing number of neighbors k or outliers items, the similarity values can become very low and unsuitable users could unintentionally be included in the recommendation lists. To avoid this, we introduced two filtering parameters, min_{sim} as a minimal similarity threshold between items, and min_{conf} as a limitation of the user-item confidence.

Active user boosting. We assumed that active users that exhibited more positive interactions with a similar job offer are more likely to be interested in a job. Thus, we decreased the user-item confidence value for less active users with a sigmoid decay function depending on the number of positive interactions $pos(u, i)$ the user had with the k nearest neighbors items and a tuning parameter β_{pop}:

$$conf_{pop}(u, i) = \frac{1}{(1 + e^{-\beta_{pop} \cdot pos(u, i)})} conf(u, i) \tag{8}$$

In our scenario, positive interactions with items refer to clicks, bookmarks, replies, and the interest of a recruiter.

Disinterested user filter. On the other hand, more active users might also be more active in a negative way, i.e., they delete many job offers, which might indicate that they are not interested in a new job at all. To exclude these users from the recommendation lists in general, we set up a preprocessing routine that calculates a ratio between positive and negative interactions for every user and uses a threshold t_{neg} to filter users by this ratio. Afterwards, filtered users are never considered in the process of identifying suitable user-job combinations.

3.3 Fallback Strategy

Our Item-kNN approach only works for users that have at least one, but preferably multiple interactions in the given log data. Since users without any associated interactions, however, make up a considerable amount of the user base (16%), we implemented a simple content-based fallback strategy based on these users' explicit profiles. For each combination of cold-start user and target item, we calculate the similarity as stated below and fill up the recommendation lists for items that did not receive enough recommendations from the Item-kNN approach, e.g., user lists shorter than 100 for the offline evaluation.

$$sim_{ui}(u, i) = \frac{1}{\sum_{f_i \in F_{UI}} w_{f_i}} \sum_{f_i \in F_{UI}} sim_{f_i}(u, i) \cdot w_{f_i} \tag{9}$$

The similarity calculation is similar to the formula for the Item-kNN itself, and F_{UI} defines the set of features users and items have in common, which are, again, marked with a star in Table 3. Furthermore, we applied the same weights and similarity functions for the individual features as for the kNN method.

3.4 Framework Implementation and Parameter Optimization

To test and optimize the above-mentioned Item-kNN and fallback approaches, we implemented a recommendation and evaluation framework in Java. The maximum memory consumption in the tests was at most 35 GB, and recommendations for the official offline target item set could be generated in less than 2 hours on a virtual machine with 50 processor cores.

Even though the runtime of the recommendation framework is not excessively high, we had to be able to test different feature weight combinations and algorithm parameters quickly to optimize our kNN approach. To this end, we distributed the kNN implementation by splitting up the workload based on the target items onto multiple machines. Each of these virtual machines in our university computing cluster remained in an idle state with the whole data set loaded into memory at all times. A dedicated command-and-control server then instructed these slave machines via a REST web service to use a specific set of parameters and feature weights to generate recommendation lists for a chunk of the target items. Afterwards, the server automatically aggregated these results and calculated the evaluation score.

We used this distributed evaluation architecture to efficiently test different algorithm parameterizations and feature weight combinations on the previously mentioned *local* data set. The results of this optimization are reported in Section 4.

3.5 Adapting to the Online Evaluation

As previously described, the recommendation goal of the online evaluation phase substantially differed from the offline setting. In the offline phase, each user could be placed in the recommendation lists for multiple items. Thus, it was possible to give multiple recommendations to each user. In the online phase, however, each user could only receive one single recommendation per day.

To satisfy the above-mentioned constraint, we created a post-processing component in our recommendation framework. This component takes the confidence scores of the Item-kNN method and the fallback strategies to arrange all of the recommendation lists so that each user is only recommended for one item. In this context, two goals have to be balanced: (a) users should be assigned to an item for which they have a high confidence score and (b) items should not starve, i.e., they should receive at least a certain number of recommendations.

Our post-processing scheme balances these goals heuristically as follows. We first assign all users to the one item with the highest confidence score for them based on the Item-kNN method. This distribution would, in theory, give the best results in terms of user satisfaction, but it would leave many items with few or no assigned users. Thus, the satisfaction among job providers would be low, which is reflected in a low (item) evaluation score. To find an acceptable middle ground, we iteratively swap users from their originally assigned items to starving items. To this end, we define two thresholds. If an item has more than N_h users assigned to it, we consider it a swap candidate, i.e., we can swap users *away* from it. On the other hand, if an item has less then N_l users assigned to it, it is considered as starving, and we try to swap users *to* it until it reaches N_l assignments.

Our strategy will then greedily try to swap the best-matching users to each of the items with less than N_l assignments. In case these users are already assigned to an item with less then N_h assignments, it will try to find other suitable users with lower, but acceptable confidence scores and so on. This will continue until either all items have N_l or more users assigned to them or until no further swaps are possible due to the threshold constraints. Afterwards, a similar procedure is executed for the fallback strategy to, again, try to fill up starving items' recommendation lists.

Table 4: Optimized parameter configurations and features weights from the local evaluation that were applied in both parts of the challenge. Features annotated with stars were available in the same form both for users and for items.

Parameter	Value
k	425
w_{click}	1
$w_{bookmark}$	2
w_{reply}	1.7
$w_{recruiter}$	2
w_{delete}	-0.65
b_{cc}	1
min_{sim}	0.1
min_{conf}	0.1
β_{pop}	0.3
t_{neg}	-0.1

Feature	Weight
Title*	6
Tags	5
Career level*	0.7
Discipline*	1.8
Industry*	1
Employment	1.5
Country*	3
Region*	3.5
Distance	5

4 RESULTS AND OBSERVATIONS

In the end, our team was placed fifth of 80 registered teams in the offline evaluation and sixth among the 25 competitors who were allowed to participate in the online competition. In both cases, our final scores were about 20% lower than the winning team. In the following, we share some more information about the relative importance of the features that we used in our algorithm and other algorithms that we tested during the competition.

Importance of individual features. Table 4 shows the optimal parameter values and the finally used feature weights that led to the best results both in the offline and the local evaluation. The highest feature weights (in terms absolute numbers) were applied for the *job title* and the *tags* that were assigned to the job offers. However, these numbers have to be interpreted with care as the Jaccard index that was used as a similarity measure leads to absolute values that are not comparable with other feature similarities, i.e., they are generally lower than the values for other features. Nonetheless, matching job titles and job descriptions seems to be an important factor for the performance. The *geographic location* of the job is another important feature according to our optimizations, and the weight values for the country and the region are also comparably high. In that context, the probably even more important feature is the actual *distance* between the location of the job offers, which we calculated based on the given geo-locations. On the other hand, the weights for the *industry* and for the *career* level received comparably low values. One reason for that phenomenon could lie in the fact that the feature values are quite unbalanced. One career level, for example, appears much more frequent than others.

Heuristics and alternatives. In Section 3.2, we described a number of heuristics that led to positive effects on the results, including, e.g., filtering out potentially disinterested users. The impact of each individual heuristic on the final score is shown in Table 5.

Table 5: Effectiveness of the tested heuristics in the offline evaluation. All: Best configuration, No DU: No disinterested user filtering, No CF: No co-occurring feature values, No MS: No min similarity filtering, No AU: No active user boosting.

Heuristics	All	No MS	No CF	No DU	No AU	None
Result	57,121	56,689	56,555	56,529	48,140	47,416

Additionally, we made further experiments using a set of other heuristics, which we assumed plausible but which did not have the desired effects. We, for example, incorporated a temporal decay function for the interactions, which did, however, not perform well, possible due to the rather short time span that was covered by the log data. Furthermore, we converted the impressions entries in the logs into implicit negative feedback, in case there was no subsequent positive log entry. And, we tried to filter out users who did not reach a certain click-conversion rate threshold and thus implicitly showed a lack of interest in job offers. These heuristics did, however, not help to improve the performance, maybe because our approaches were too simplistic.

We also tested an entirely different approach and implemented a traditional content-based approach (CB) where we constructed feature-based user profiles dependent on the users' past interactions. While this method proved not to be competitive with our nearest-neighbor approach, it led to results that were much better than the classification-based baseline method provided by XING[6] Table 6 shows an overview of the results that were obtained with different strategies in the offline evaluation. Combining the individual techniques in a hybrid approach did not lead to any improvements.

Reflections on the design of the challenge. Since this year's RecSys challenge both included an offline and an online evaluation, it had the potential to be one of the rare opportunities where one can assess to which extent algorithms that work well in offline experiments also work well when deployed in a real environment. Looking at the leaderboard, we can indeed observe that several teams were successful in both evaluation setups. The scoring metric was the same in both scenarios, and the results therefore suggest that being able to "post-dict" interactions based on historical data to some extent transfers to the capability of predicting future events.[7]

A limitation of the challenge setup was that there were a number of differences between the offline and the online recommendation scenarios, as described above. The fact that the same teams achieved high scores in both settings could therefore partially be caused by the ability of the team to adapt to the different problem settings, and it is not certain that the same algorithms were used in the two settings. Another issue is that the recommendations that were uploaded by the teams went through an unspecified filter before being pushed out. The final outcome can therefore depend to some extent on how good a team was at finding recommendations that were not filtered. Furthermore, an unexplained phenomenon during the challenge was that the absolute scores of all teams varied considerably across the different evaluation weeks, which indicates that there were additional factors out of the control of the participants.

[6]https://github.com/recsyschallenge/2017/tree/master/baseline

[7]Whether or not the chosen evaluation measure is representative for the business value for the platform can however not be answered based on that observation.

Table 6: Results of the tested methods in terms of the challenge's metric (offline). kNN: The proposed method, CB: A content-based approach, BL: The baseline approach provided by XING, BL+: BL with additional features.

Approach	kNN+Fallback	kNN	CB	BL+	BL
Result	57,121	54,876	20,829	13,370	10,475

5 CONCLUSIONS

We presented a nearest-neighbors approach to select recipients to recommend new items. The method, which has different advantages over more complex models, led to good results in the ACM RecSys 2017 challenge. The specific problem setting of the challenge is highly relevant in practice but has in our view not been investigated to a sufficient extent in the academic literature so far. Our future works include the exploration of additional domain-specific heuristics and to apply more complex models to the problem.

ACKNOWLEDGEMENTS

We thank the organizers at XING for the opportunity to participate in this unique competition.

REFERENCES

[1] Paolo Cremonesi, Franca Garzotto, and Roberto Turrin. 2012. Investigating the Persuasion Potential of Recommender Systems from a Quality Perspective: An Empirical Study. *Transactions on Interactive Intelligent Systems* 2, 2 (2012), 11.

[2] Mamadou Diaby, Emmanuel Viennet, and Tristan Launay. 2013. Toward the Next Generation of Recruitment Tools: An Online Social Network-based Job Recommender System. In *Proc. ASONAM '13*. 821–828.

[3] Abel Fabian, Yashar Deldjoo, Mehdi Elahi, and Daniel Kohlsdorf. 2017. RecSys Challenge 2017: Offline and Online Evaluation. In *Proc. RecSys '17*. (forthcoming).

[4] Florent Garcin, Boi Faltings, Olivier Donatsch, Ayar Alazzawi, Christophe Bruttin, and Amr Huber. 2014. Offline and Online Evaluation of News Recommender Systems at Swissinfo.ch. In *Proc. RecSys '14*. 169–176.

[5] Carlos A. Gomez-Uribe and Neil Hunt. 2015. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *Transactions on Management Information Systems* 6, 4 (2015), 13:1–13:19.

[6] Dietmar Jannach and Gedas Adomavicius. 2016. Recommendations with a Purpose. In *Proc. RecSys '16*. 7–10.

[7] Dietmar Jannach and Kolja Hegelich. 2009. A case study on the effectiveness of recommendations in the mobile internet. In *Proc. RecSys '09*. 205–208.

[8] Dietmar Jannach and Malte Ludewig. 2017. Determining Characteristics of Successful Recommendations from Log Data - A Case Study. In *Proc. SAC '17*.

[9] Dietmar Jannach and Malte Ludewig. 2017. When Recurrent Neural Networks meet the Neighborhood for Session-Based Recommendation. In *Proc. RecSys '17*. (forthcoming).

[10] Evan Kirshenbaum, George Forman, and Michael Dugan. 2012. A Live Comparison of Methods for Personalized Article Recommendation at Forbes.com. In *Proc. ECML/PKDD '12*. 51–66.

[11] Lerche Lerche, Dietmar Jannach, and Malte Ludewig. 2016. On the Value of Reminders within E-Commerce Recommendations. In *Proc. UMAP '16*. 27–25.

[12] Yao Lu, Sandy El Helou, and Denis Gillet. 2013. A Recommender System for Job Seeking and Recruiting Website. In *Proc. WWW '13*. 963–966.

[13] Andrzej Pacuk, Piotr Sankowski, Karol Wegrzycki, Adam Witkowski, and Piotr Wygocki. 2016. RecSys Challenge 2016: Job Recommendations Based on Preselection of Offers and Gradient Boosting. In *Proc. RecSys Challenge '16*. 10:1–10:4.

[14] Ioannis Paparrizos, B. Barla Cambazoglu, and Aristides Gionis. 2011. Machine Learned Job Recommendation. In *Proc. RecSys '11*. 325–328.

[15] Roberto Turrin, Andrea Condorelli, Roberto Pagano, Massimo Quadrana, and Paolo Cremonesi. 2015. Large scale music recommendation. In *Proc. LSRS '15*.

[16] Koen Verstrepen and Bart Goethals. 2014. Unifying Nearest Neighbors Collaborative Filtering. In *Proc. RecSys '14*. 177–184.

[17] Wenming Xiao, Xiao Xu, Kang Liang, Junkang Mao, and Jun Wang. 2016. Job Recommendation with Hawkes Process: An Effective Solution for RecSys Challenge 2016. In *Proc. RecSys Challenge '16*. 11:1–11:4.

[18] Dávid Zibriczky. 2016. A Combination of Simple Models by Forward Predictor Selection for Job Recommendation. In *Proc. RecSys Challenge '16*. 9:1–9:4.