Neighborhood Troubles: On the Value of User Pre-Filtering To Speed Up and Enhance Recommendations

Emanuel Lacic
Know-Center GmbH
Graz, Austria
elacic@know-center.at

Dominik Kowald
Know-Center GmbH
Graz, Austria
dkowald@know-center.at

Elisabeth Lex
Graz University of Technology
Graz, Austria
elisabeth.lex@tugraz.at

Abstract

In this paper, we present work-in-progress on applying user pre-filtering to speed up and enhance recommendations based on Collaborative Filtering. We propose to pre-filter users in order to extract a smaller set of candidate neighbors, who exhibit a high number of overlapping entities and to compute the final user similarities based on this set. To realize this, we exploit features of the high-performance search engine Apache Solr and integrate them into a scalable recommender system. We have evaluated our approach on a dataset gathered from Foursquare and our evaluation results suggest that our proposed user pre-filtering step can help to achieve both a better runtime performance as well as an increase in overall recommendation accuracy.

1 Introduction

In the past decade, there has been a vast amount of research in the field of recommender systems, mostly focusing on developing novel recommendation algorithms [19] and improving recommender accuracy [14]. Thus, many well known methods are available, such as Content-based Filtering [1], Collaborative Filtering [16] or Matrix Factorization [7], which all have their unique strengths and weaknesses. With the arrival of the big data era, recommender systems are nowadays not only expected to analyze a lot of data, but also to handle frequent streams of new data. Traditional recommender systems usually analyze the data offline and update the generated model in regular time intervals. However, choices made by users depend on factors that are susceptible to change anytime and to re-train such models tends to be a time-consuming task (especially when the data is sparse [15]).

As such, the attention of the recommender systems' research community has recently shifted towards recommendation systems that process streaming data online and recommend entities in near real-time. For example, recent work from Huang et al. [5] presented TencentRec, a real-time recommender system that is based on Apache Storm. Specifically, they tackle item-based Collaborative Filtering and handle the data sparsity problem by recommending most popular entities from the user's demographic group. Another scalable item-based Collaborative Filtering recommender model was implemented by Chandramouli et al. [2]. This approach is based on a stream processing system and focuses entirely on using explicit rating data.

In our previous work [9], we presented a scalable recommender framework using a Microservices-based architecture to recommend a diverse set of entities in near real-time by leveraging the Apache Solr search engine. In this work, we focus on adapting the non-probabilistic user-based Collaborative Filtering (UB-CF) algorithm [18] to further improve its runtime performance by integrating a user pre-filtering step. This approach is especially useful in settings, in which it is not desirable to allocate additional resources but rather to optimize the usage of the available hardware.

Bottleneck. Collaborative Filtering is usually accomplished in two steps: (i) the k-nearest neighbors are determined using a similarity metric (e.g., cosine similarity), and (ii) entities of these neighbors are recommended that the target user u_t has not yet consumed [18]. As shown in previous work [11], both steps can be adapted to search the data space in a scalable way and to retrieve the relevant content in near real-time. However, one performance bottleneck in this workflow is the number of neighbors that need to be processed (i.e., users which rate the same item as u_t).

While the neighborhood size k is usually picked to be between 20 and 60 [4], it is still necessary to fetch the history of all neighbors and to calculate how similar a poten-

tial neighbor is to u_t. Moreover, the calculated similarities need to be sorted in order to pick the top-k similar users. Common implementations of such operations have a complexity of $O(n \times log(n))$[1]. As such, the larger the neighborhood of u_t is, the larger the impact on the runtime performance could be.

Contributions. In order to cope with such a performance bottleneck, in this paper, we present how to extend our scalable recommender system to save extra processing power by exploiting the Apache Solr search engine. We demonstrate that pre-filtering of users who exhibit a high number of overlapping entities can lead to better runtime performance as well as recommendation accuracy.

2 Approach

In this section, we present our approach for speeding up and enhancing CF-based recommendations with a user pre-filtering step.

2.1 Adaptation of Collaborative Filtering with User Pre-Filtering

In order to improve runtime, we could just decide to run the first step of CF in parallel, e.g., by having multiple processing nodes whose task is to fetch a user's history and calculate the similarity to the target user u_t. In this work, our aim is yet to increase the runtime performance in cases when there is also a limitation in terms of available processing resources.

Therefore, we propose to adapt the first step of UB-CF by pre-filtering the candidate set of possible similar users beforehand. We do that in a greedy way by finding the top-N candidate users with the highest overlap with respect to the available entity interactions (e.g., ratings). In this pre-filtering step, the similarity between u_t and a possible candidate user u_c is then calculated as follows:

$$OV(u_t, u_c) = |\Delta(u_t) \cap \Delta(u_c)| \quad (1)$$

where $\Delta(u)$ corresponds to the set of entities some user u has interacted with in the past. As we will show next in Section 2.2, this can be done very efficiently by exploiting the Apache Solr search engine. This way, we increase the probability that users with a high overlap will in the end be picked as the top-k similar users. Also, by picking a reasonable value for N, we aim to positively influence the runtime performance of those users, which exhibit many neighbors.

[1]For example, Java's popular TreeMap implementation (https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html). This could also be improved to a complexity of $O(n+k \times log(n))$ by implementing a partial sorting algorithm.

2.2 Implementation Details

In our previous work [11], we have introduced a scalable software architecture, which can be applied to various entity recommendation scenarios. As seen in Figure 1, such an architecture allows us to recommend entities in an isolated environment. That is, every module can be deployed and started multiple times either on the same or on different machines and runtime performance can be guarantied by scaling individual nodes horizontally. To keep track of and coordinate all deployed nodes, we make use of Apache ZooKeeper [2]. An entity recommender is then set up of five modules which leverage Apache Solr to perform user pre-filtering in an efficient way.

Service Provider is the main entry point which acts as a proxy for the specific entity recommendation scenario (e.g., venues, movies, songs, etc.). It provides a REST-based interface to modules that are designated to handle the calculation of the requested entity recommendations as well as to store new data (e.g., interactions with the recommendable entities).

Recommender Evaluator aims to simulate user behaviour by splitting the data into training and test sets (see e.g., [13]). That is, for each user, a given number of entities is removed from the training set and added to the test set to be predicted. The difference between the recommended and the real data from the test set is then used to determine the success of the prediction. In addition to providing a diverse set of well-established recommender evaluation metrics, the evaluation procedure allows to simulate varying loads in order to better grasp the impact on the runtime when an increasing number of recommendations are requested.

Recommender Engine contains recommender algorithms that use Apache Solr's efficient query language. In case of UB-CF, this allows us to immediately consider frequent data updates while providing real-time recommendations. Here, we calculate the probability that the target user u_t will like an entity e by the following formula [18]:

$$pred(u_t, e) = \sum_{u_c \in neighbors(u_t, e)} sim(u_t, u_c) \quad (2)$$

where $neighbors(u_t, e)$ is the set of pre-filtered candidate neighbors of u_t that have interacted with e and $sim(u_t, u_c)$ is the final similarity value between the users u_t and u_c.

Recommender Customizer allows to configure the implemented recommender approaches in form of recommender profiles. The sole purpose of these files is to customize a single recommender approach and to provide a reference to it. For instance, depending on the entity recommendation scenario that we wish to tackle, a Collaborative Filtering

[2]http://zookeeper.apache.org/

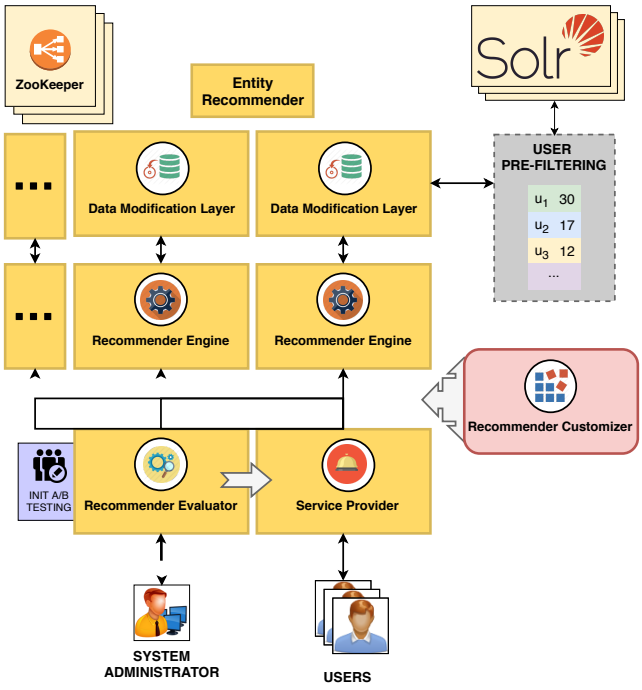

Figure 1: Architecture diagram of our scalable entity recommender framework that leverages the Apache Solr search engine to provide recommendations in near real-time. By pre-filtering users using Solr's facet functionality we not only receive the number of overlapping entites for the top-N users but also speed up the recommendation process.

approach can be configured to use different kinds of similarity metrics (e.g., as shown in [8]), neighborhood sizes and use either explicit (e.g., ratings) or implicit data (e.g., clicks) for recommending entities of similar users.

Data Modification Layer acts as an agent between the recommender system and the Apache Solr search engine. By utilizing Solr, we have the capability for horizontal scaling on the data storage side [10] by creating either shards (i.e., splitting the data into smaller indices to increase the performance of search queries for huge data sets) or replicas (cloning the existing shards to another machine to increase the fault-tolerance of the whole system).

User Pre-Filtering is performed very efficiently by exploiting Solr's facet[3] functionality [10]. This is basically an arrangement of search results into categories (e.g., user id's) along with numerical counts of matching documents. For example, if we perform a facet search on the user id over the whole document corpus (i.e., dataset) we would get for the top-N users the exact count of corresponding entity interactions, where N is a query parameter that needs to be provided to Solr and the resulting set is sorted in a descending order.

[3]https://lucene.apache.org/solr/guide/7_0/faceting.html

Data Type	Density	Mean	STD	Skewness	Kurtosis
user - items	0.000015	2.32	54.08	227.53	62,884.43

Table 1: Statistics of our Foursquare dataset.

By defining a filter within the facet query to look only into entities from the target user's u_t history, we not only reduce the search space but also get exactly the desired $OV(u_t, u_c)$ values for the top-N pre-filtered candidate users as defined in Section 2.1. Such a greedy pre-filtering of candidate users can be computed in milliseconds which in turns allows to speed up the generation of the final entity recommendations.

3 Evaluation

In this section, we describe our evaluation process, including our dataset, experimental setup as well as preliminary results.

3.1 Dataset

Traditionally, recommender systems deal with two types of entities, users and items. To show how user pre-filtering can speed up and enhance recommendations, we used the Foursquare dataset provided by the authors of [12, 17]. The dataset consists of 2,153,471 users, 1,143,092 venues (i.e., items) and 2,809,581 ratings. In order to make our dataset comparable, we present the summary of common statistical data measures for the user-item relationships in Table 1. The data density shows the proportion of actually known entities (e.g., ratings) to all possible entities that could be known by the user. This is rather a sparse dataset as the rating density is 0.000015.

Besides the mean entity assignments and their standard deviation, skewness and kurtosis [6] are also two important statistical measures. The skewness is a measure of the symmetry of a distribution. A symmetric distribution has a skewness of 0. In our case, the skewness is greater than 0 which means that the distribution is right-tailed (i.e., most data is concentrated on the left side of its function). Kurtosis is a measure of the distribution "peakness", where a higher kurtosis value signifies lower concentration around its mean. Especially in the case of the user - item relationship, such a high kurtosis value means that the distribution has a sharper peak and broader tails.

3.2 Experimental Setup

We evaluated all users, which have at least one rated item in the training set. Thus, we extracted all users that interacted with at least 11 items (= 58,046 users in total) and split the dataset in two different sets (training and test set) using a method similar to the one described in [13]. In other words, for each user, we withheld 10 items from the dataset and added them to the test set to be predicted. The rest of the data was used for training.

Approach	Description
MP	A baseline that recommends most popular items
CF_{Full}	UB-CF which calculates similarities for all neighbors
$CF_{OV=20}$	UB-CF with a greedy pick of top-20 overlapping users
$CF_{OV=40}$	UB-CF with a greedy pick of top-40 overlapping users
$CF_{OV=60}$	UB-CF with a greedy pick of top-60 overlapping users
$CF_{OV=80}$	UB-CF with a greedy pick of top-80 overlapping users
$CF_{OV=100}$	UB-CF with a greedy pick of top-100 overlapping users

Table 2: We compare five variants of our user pre-filtering technique with a Most Popular baseline and a classic Collaborative Filtering approach.

Chosen Neighborhood Sizes and Recommendation Approaches. With respect to neighborhood sizes, the distribution is right-tailed and the average neighborhood size is 764, the median, however, is only 4, while the maximum neighborhood size of a user is 125,046. We determined values for N in line with the literature [4], i.e., between 20 and 60. Specifically, we hypothesize that the same interval of values is valid for a greedy pick of candidate neighbors. We also evaluated $N = 80$ and $N = 100$ to test the impact of a larger candidate set on the accuracy. As shown in Table 2, for each evaluated user seven recommendation approaches were evaluated.

Evaluation Metrics. In our evaluation, we report the mean and standard deviation of the runtime performance as well as the recommendation accuracy in terms of Precision (P), Recall (R), Normalized Discounted Cumulative Gain (nDCG) and User Coverage (UC) [19].

3.3 Preliminary Results

Our evaluation results are summarized in Table 3. The experiments have been executed on an IBM System x3550 server with two 2.0 GHz six-core Intel Xeon E5-2620 processors, a 1TB ServeRAID M1115 SCSI Disk and 128 GB of RAM using one instance and Ubuntu 14.04.1. with Apache Solr 4.10.2.

All performance metrics are reported for 10 recommended items (k=10)[4]. On average, CF_{Full} took approximately 2 seconds with a rather high standard deviation of 9.6 seconds. With the adapted CF approaches, where we calculated the similarity only on the top-N overlapping users, the runtime performance drastically improves (i.e., below 90 ms, which is more than 23 times faster than the CF_{Full}). Such a runtime is even comparable to the one of the simple MostPopular (MP) baseline.

Interestingly, the accuracy also increases when we pre-filter the candidate set of similar users, as also shown in terms of nDCG in Figure 2 for different values of k. We achieved the best performance, both in terms of runtime and accuracy, by utilizing the top-60 overlapping users.

[4]Please note that the literature usually uses the term k for both the number of similar users in the UB-CF approach as well as the number of items that are being recommended. In Section 3.3 we talk about the number of recommended items.

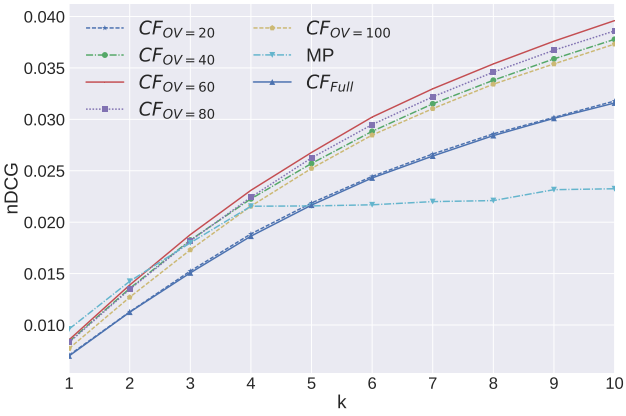

Figure 2: nDCG plot showing the difference in accuracy when using top-N overlapping users as candidate similar users.

Here, the runtime performance was almost the same as when running the MP baseline. This suggests that creating a pre-filtered candidate set of similar users not only yields better runtime performance but can also contributes to a higher recommendation accuracy.

4 Conclusion and Future Work

In this paper, we presented work-in-progress on adapting Collaborative Filtering by integrating a user pre-filtering step to speed up and enhance entity recommendations. Specifically, we adapt the approach by applying user pre-filtering, in which we generate a smaller set of candidate neighbors in a greedy fashion (i.e., by focusing on neighbors with a higher number of overlapping entities). Our results suggest that our pre-filtering approach can not only achieve a better runtime performance but also is able to increase the overall accuracy compared to a classic CF algorithm without user pre-filtering.

Limitations and Future Work. One limitation of our work is that we evaluated our approach only on one dataset. As a next step, we want to validate our results in a more comprehensive study using datasets with different types of entities that can be recommended. Here, we especially aim to validate our approach in course of the Analytics for Everyday Learning (AFEL) project[5] [3] for recommending learning resources. This would also allow us to evaluate this approach in course of an online study to measure the real user acceptance of the recommendations.

Acknowledgments. The authors would like to thank the Social Computing research area of the Know-Center GmbH and the AFEL consortium for their support. This work was funded by the Know-Center GmbH Graz (Austrian FFG COMET Program) and the European-funded H2020 project AFEL (GA: 687916).

[5]http://afel-project.eu/

	Approach	$\overline{T}$ (ms)	σ (ms)	$P@10$	$R@10$	$nDCG@10$	UC
	Most Popular	78.59	20.00	.0285	.0285	.0232	**100%**
CF	CF_{Full}	2,053.45	9,600.63	.0611 (.0918)	.0527 (.0792)	.0316 (.0475)	66.56%
CF	$CF_{OV=20}$	**59.56**	**60.08**	.0586 (.0890)	.0541 (.0821)	.0318 (.0483)	65.87%
CF	$CF_{OV=40}$	65.47	69.61	.0689 (.1042)	.0645 (.0974)	.0378 (.0571)	66.21%
CF	$CF_{OV=60}$	74.62	85.83	**.0724** (**.1095**)	**.0678** (**.1026**)	**.0396** (**.0599**)	66.10%
CF	$CF_{OV=80}$	82.40	102.75	.0707 (.1077)	.0661 (.1007)	.0386 (.0588)	65.62%
CF	$CF_{OV=100}$	87.38	115.17	.0693 (.1055)	.0646 (.0983)	.0373 (.0568)	65.70%

Table 3: For all approaches, we report the mean runtime, as well as the accuracy and user coverage (UC). The values in brackets represent the results normalized to the UC in the row, while the ones without are calculated on a 100% UC (i.e., all 58,046 users). Our neighborhood pre-filtering approach leads to improvements with respect to runtime and accuracy.

References

[1] M. Balabanović and Y. Shoham. Fab: content-based, collaborative recommendation. *Communication of ACM*, 40(3):66–72, Mar. 1997.

[2] B. Chandramouli, J. J. Levandoski, A. Eldawy, and M. F. Mokbel. Streamrec: A real-time recommender system. In *Proc. of SIGMOD'11*, 2011.

[3] M. d'Aquin, D. Kowald, A. Fessl, E. Lex, and S. Thalmann. Afel-analytics for everyday learning. In *Companion of the The Web Conference 2018 on The Web Conference 2018*, 2018.

[4] J. Herlocker, J. A. Konstan, and J. Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Information retrieval*, 5(4):287–310, 2002.

[5] Y. Huang, B. Cui, W. Zhang, J. Jiang, and Y. Xu. Tencentrec: Real-time stream recommendation in practice. In *Proc. of SIGMOD'15*, 2015.

[6] D. N. Joanes and C. A. Gill. Comparing measures of sample skewness and kurtosis. *Journal of the Royal Statistical Society. Series D, year = 1998, publisher = Wiley for the Royal Statistical Society,.*

[7] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug. 2009.

[8] E. Lacic, D. Kowald, L. Eberhard, C. Trattner, D. Parra, and L. Marinho. Utilizing online social network and location-based data to recommend products and categories in online marketplaces. In *Mining, Modeling, and Recommending 'Things' in Social Media*, pages 96–115. Springer, 2015.

[9] E. Lacic, D. Kowald, and E. Lex. Tailoring recommendations for a multi-domain environment. *Proc. of RecSysKTL'17 co-located with ACM RecSys'17*.

[10] E. Lacic, D. Kowald, D. Parra, M. Kahr, and C. Trattner. Towards a scalable social recommender engine for online marketplaces: The case of apache solr. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 817–822. ACM, 2014.

[11] E. Lacic, M. Traub, D. Kowald, and E. Lex. Scar: Towards a real-time recommender framework following the microservices architecture. In *Proc. of LSRS'15*.

[12] J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel. Lars: A location-aware recommender system. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, pages 450–461. IEEE Computer Society, 2012.

[13] D. Parra-Santander and P. Brusilovsky. Improving collaborative filtering in social tagging systems for the recommendation of scientific articles. In *Proc. WI-IAT'10*. IEEE Computer Society, 2010.

[14] C. Rana and S. K. Jain. A study of the dynamic features of recommender systems. *Artificial Intelligence Review*, 43(1):141–153, 2015.

[15] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proc. of ICML'08*. ACM, 2008.

[16] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[17] M. Sarwat, J. J. Levandoski, A. Eldawy, and M. F. Mokbel. Lars*: An efficient and scalable location-aware recommender system. *IEEE Trans. on Knowl. and Data Eng.*, 26(6):1384–1399, June 2014.

[18] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. The adaptive web. chapter Collaborative Filtering Recommender Systems, pages 291–324. Springer, 2007.

[19] G. Shani and A. Gunawardana. Evaluating recommendation systems. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*. Springer US, 2011.