Automatic Music Playlist Continuation via Neighbor-based Collaborative Filtering and Discriminative Reweighting/Reranking

Lin Zhu
Ctrip Travel Network Technology Co., Limited.
Shanghai, P.R. China
zhulb@Ctrip.com

Bowen He
Ctrip Travel Network Technology Co., Limited.
Shanghai, P.R. China
bwhe@Ctrip.com

Mengxin Ji
University of California, Davis.
Davis, CA, USA
mji@ucdavis.edu

Cheng Ju
University of California, Berkeley
Berkeley, CA, USA
cju@berkeley.edu

Yihong Chen
Ctrip Travel Network Technology Co., Limited.
Shanghai, P.R. China
yihongchen@Ctrip.com

ABSTRACT

The focus of RecSys Challenge 2018 is automatic playlist continuation (APC), which refers to the task of adding one or more tracks to a playlist in a manner that does not alter the intended characteristics of the original playlist. This paper presents our approach to this challenge. We adopted neighbor-based collaborative filtering approaches since they are able to deal with large datasets in an efficient and effective way, and have previously been shown to perform well on recommendation problems with similar characteristics. We show that by choosing an appropriate similarity function that properly accounts for the list-song similarities, simple neighbor-based methods can still achieve highly competitive performance on the MPD data, meanwhile, by using a set of techniques that discriminantly finetune the recommendation lists produced by neighbor-based methods, the overall recommendation accuracy can be improved significantly. By using the proposed approach, our team HAIR was able to attain the 6th place in the competition. We have open-sourced our implementation on https://github.com/LauraBowenHe/Recsys-Spotify-2018-challenge.

CCS CONCEPTS

• **Information systems** → **Recommender systems**;

RecSys Challenge '18, October 2, 2018, Vancouver, BC, Canada

ACM ISBN 978-1-4503-6586-4/18/10…$15.00
https://doi.org/10.1145/3267471.3267481

KEYWORDS

Music recommender systems, automatic playlist continuation (APC), RecSys challenge

ACM Reference Format:
Lin Zhu, Bowen He, Mengxin Ji, Cheng Ju, and Yihong Chen. 2018. Automatic Music Playlist Continuation via Neighbor-based Collaborative Filtering and Discriminative Reweighting/Reranking. In *Proceedings of the ACM Recommender Systems Challenge 2018 (RecSys Challenge '18), October 2, 2018, Vancouver, BC, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3267471.3267481

1 INTRODUCTION

Thanks to the growing popularity of music streaming services such as Spotify, KKBOX, and Apple Music, music fans are now given access to a variety of different music resources, from which they can potentially discover a wealth of new favorite songs. Given such overwhelming quantities of available listening choices, personalized machine learning algorithms have gradually assumed the roles that originally belong to Radio DJs, namely to provide listen suggestions to users and increase their engagement. So far, a large number of work have been done to develop music recommendation systems both in academia and industry [1, 3, 4, 6, 19], nevertheless existing systems are still far from satisfactory and have much room for improvement [17]. Major unsolved issues include ones that are shared by recommendation systems in general, such as the so-called cold-start problem, i.e., due to lack of historical data, it is difficult or unreliable to apply such systems for new users or songs that are only recently added into the catalog [20], and ones that are specific to the music recommendation domain, such as the appropriateness of repeated recommendations [17]. In particular, music pieces are frequently consumed sequentially, i.e., in a **playlist**. For example, a study carried out by the Music Business Association in 2016, showed that playlists accounted for 31% of music listening time among listeners in the USA. Given the importance of playlists as a mode for music consumption, considerable attention has

been devoted to developing algorithms for automatic playlist continuation (APC) [2, 4, 15–18], which refers to the task of adding one or more tracks to a playlist in a manner that does not alter the same target characteristics of the original playlist. A accurate APC system can assist users with finding enjoyable songs beyond the end of a finite-length playlist, and thereby improve user experience.

In this year, the RecSys Challenge, a competition held annually as part of the prestigious ACM Conference on Recommender Systems (RecSys), challenges competitors from all over the world to help and build a better APC recommendation system. The Million Playlist Dataset (MPD), a large-scale dataset which consists of one million user-created playlists and associated metadata, was kindly produced by Spotify, a major online music streaming service with over 140 million active users, over 30 million tracks, and over **2 billion** playlists. A separate "challenge" set that consists of a set of playlists from which a number of tracks have been withheld was also provided. The task is to predict the missing tracks in those playlists.

To tackle this challenge, we focus on neighbor-based collaborative filtering approaches since they are able to deal with large datasets in an efficient and effective way, and have previously been shown to perform well on recommendation problems with similar characteristics [1, 22]. We show that by choosing an appropriate similarity function that properly accounts for the list-song similarities, simple neighbor-based methods can still achieve highly competitive performance on the MPD data, meanwhile, by using a set of techniques that discriminantly finetune the recommendation lists produced by neighbor-based methods, the overall recommendation accuracy can be improved significantly. The paper is organized as follows. A overview of the RecSys-Spotify challenge is presented in Section 2, Section 3 discusses the components of our approach, and the paper concludes in Section 4.

2 THE CHALLENGE DESCRIPTION

2.1 The Datasets

Two datasets are provided by Spotify for the competition:

- *The Million Playlist Dataset (MPD)*: a dataset consisting of one million user-created playlists and associated metadata. The metadata include the lists' name and descriptions and various statistical features, such the numbers of included artists/albums/tracks, etc.
- *The Challenge Dataset*, which contains 10,000 incomplete playlists. Each list is provided with associated metadata, and a number of seed songs.

2.2 The Challenge Task

The goal of the challenge is to develop a system for the task of automatic playlist continuation. For each incomplete list in the challenge set, the participants are required to generate a list of 500 recommended candidate tracks that can be added to that playlist, thereby 'continuing' the playlist. The challenge task are made up of 10 different subtasks, each of which has different amount of available (meta) data:

(1) Predict tracks for a playlist given its name only.
(2) Predict tracks for a playlist given its name and the first track.
(3) Predict tracks for a playlist given its name and the first 5 tracks.
(4) Predict tracks for a playlist given its first 5 tracks (no name).
(5) Predict tracks for a playlist given its name and the first 10 tracks.
(6) Predict tracks for a playlist given its first ten tracks (no name).
(7) Predict tracks for a playlist given its name and the first 25 tracks.
(8) Predict tracks for a playlist given its name and 25 random tracks.
(9) Predict tracks for a playlist given its name and the first 100 tracks.
(10) Predict tracks for a playlist given its name and 100 random tracks.

2.3 The Evaluation Metrics

Submissions are evaluated using three metrics. Formally, let the ground truth set of tracks be denoted by G, and the ordered list of recommended tracks by R, and the size of a set or list be denoted by $|\cdot|$, then three used metrics can be respectively defined as:

2.3.1 R-precision. The number of retrieved relevant tracks divided by the number of known relevant tracks:

$$\text{R-precision} = \frac{\left|G \bigcap R_{1:|G|}\right|}{|G|}. \tag{1}$$

2.3.2 Normalized discounted cumulative gain (NDCG). Normalized DCG (NDCG) is determined by calculating the discounted cumulative gain (DCG) and dividing it by the ideal DCG in which the recommended tracks are perfectly ranked:

$$NDCG = \frac{DCG}{IDCG}, \tag{2}$$

where $IDCG$ and DCG are defined as:

$$DCG = rel_1 + \sum_{i=2}^{|R|} \frac{rel_i}{\log_2 i}, \tag{3}$$

$$IDCG = 1 + \sum_{i=2}^{|G \bigcap R|} \frac{1}{\log_2 i}. \tag{4}$$

2.3.3 Recommended Songs clicks. This metric is based on 'Recommended Songs', a Spotify feature that recommends 10 tracks to add to a playlist given the tracks that it already contains, and the recommended list can be refreshed to produce 10 more tracks each time. Recommended Songs clicks is then defined as the number of refreshes needed before a relevant track is encountered:

$$\text{clicks} = \left\lfloor \frac{\arg\min_i \{R_i : R_i \in G\} - 1}{10} \right\rfloor. \tag{5}$$

3 OUR SOLUTION

3.1 Neighbor-based Collaborative Filtering for Binary Feedback

A commonly used approach for building accurate recommender models is collaborative filtering (CF), which could be abstracted as the problem of assigning weights to missing edges in a bipartite graph [21]. Concretely, for APC recommendation, one set of nodes (denoted as $\mathcal{P}$ henceforth) in the bipartite graph are *playlists*, and the other set of nodes (denoted as $\mathcal{T}$ henceforth) represent *tracks*, a edge between $t \in \mathcal{T}$ and $p \in \mathcal{P}$ represents that p contains t, and the challenge task can be viewed as the prediction of missing edges in the bipartite graph. Meanwhile, the edges are unweighted since a list either contains a track or not, and it is more difficult to extract graded relevance information from such binary data for recommendations [22].

Our CF solution for APC recommendation is based on the approach proposed in [1], which is an efficient variant of neighbor-based CF methods that often perform well on large-scale binary feedback datasets [1, 22].

As is the case with neighbor-based CF methods in general, the main idea behind our approach is that if playlists "similar" to playlist u contain track i then i should be recommended to continue u; similarly, tracks that are "similar" to the ones that u already contains should also be recommended. Formally, given the target playlist u, one can first assign to each candidate track i a feature vector $\mathbf{x}_i \in \mathbb{R}^{|\mathcal{P}|}$, defined as:

$$(\mathbf{x}_i)_{[v]} = \begin{cases} \frac{|\mathcal{T}(u) \bigcap \mathcal{T}(v)|}{|\mathcal{T}(v)|^{\alpha}} & \text{list } v \text{ contains } i, \\ 0 & \text{else}, \end{cases} \tag{6}$$

where $\mathcal{T}(u)$ is the set of tracks contained by the playlist u, $0 \leq \alpha \leq 1$ is a tunable hyperparameter that controls the influence of long playlists. The entries of vector (6) takes into consideration the similarities between u and other observed playlists. Then the similarity between tracks i and j is defined as:

$$s_{ij} = \frac{(\boldsymbol{x}_i)^T \boldsymbol{x}_j}{|\mathcal{P}(i)|^{\beta} |\mathcal{P}(j)|^{1-\beta}}, \tag{7}$$

where $\mathcal{P}(i)$ denotes the set of playlists that contain the track i, $0 \leq \beta \leq 1$ is another tuning hyperparameter.

Given (7), the score obtained on an track i for a target incomplete playlist u can be calculated as the arithmetic mean of similarity between i and the tracks that u is known to contain:

$$r_{ui} = \frac{1}{|\mathcal{T}(u)|} \sum_{j \in \mathcal{T}(u)} s_{ij}. \tag{8}$$

Once we scored all relevant candidates tracks using (8), we ranked them by score, and then recommended the top k with the tracks in $\mathcal{T}(u)$ excluded.

3.2 Recommendations with Only Title Information

Apart from existing tracks in a given playlist, the playlist's name also contains meaningful contextual information that can be exploited to infer its intended purpose and make recommendations [15]. Accordingly, the RecSys-Spotify challenge also offers a cold-start subtask where the participating teams are required to making recommendations for playlists with name information only. Our approach for this subtask is similar in spirit to the neighbor-based method discussed in the previous section: if playlists and the target playlist u have similar names, the tracks in these playlists should be recommended to continue u. To better group similar playlists based on their names, we first 'homogenized' the playlist names via basic text cleaning protocols, such as tokenization, conversion of all tokens to lower case, removal of punctuations and special tokens, etc. After that, by using n-gram-based text representation, each list $u \in \mathcal{T}$ can be mapped via its name to a bag/set of n-grams $\mathcal{G}(u)$. The similarity between each pair of track t and n-gram g is measured by the frequency that they "co-occur" in a playlist:

$$s_{ig} = \frac{|\{u | u \in \mathcal{P}(i), g \in \mathcal{G}(u)\}|}{|\{u | u \in \mathcal{P}(i)\}|^{\gamma}}, \tag{9}$$

where γ is a tuning parameter. Based on (9), the score obtained on an track i for a target playlist u can be calculated as the arithmetic mean of similarity between i and the n-grams that the title of u is contains:

$$r_{ui} = \frac{1}{|\mathcal{G}(u)|} \sum_{g \in \mathcal{G}(u)} s_{ig}. \tag{10}$$

3.3 Discriminative Reweighting

Although neighbor-based CF methods are easy to implement, such simplicity could potentially lead a sacrifice on recommendation quality [13]. Consider the approach discussed in Section 3.1 for example, the ranking score (8) can be rewritten in vectorial form as:

$$r_{ui} = \mathbf{s}_{ui} \cdot \left[\frac{1}{|\mathcal{T}(u)|}, \frac{1}{|\mathcal{T}(u)|}, \cdots, \frac{1}{|\mathcal{T}(u)|} \right]^T. \tag{11}$$

Equation (11) provides an alternative interpretation of our neighbor-based approach: firstly a feature representation $\mathbf{s}_{ui} \in \mathbb{R}^{|\mathcal{T}(u)|}$ is obtained for candidate track i by concatenating the similarity scores between i and the tracks in $\mathcal{T}(u)$, a linear aggregation function is then applied to $\mathbf{s}_{ui}$ to obtain the ranking score. Importantly, the aggregation function assigns equal weights to all feature dimensions, which is potentially suboptimal as the relative importances of features are neglected.

Based on the above considerations, we chose to learn more discriminative aggregation weights by solving a L2-regularized Support Vector Classification (SVC) problem:

$$\min_{\mathbf{w}} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i \in \mathcal{T}} \left(\max \left(1 - y_i \mathbf{w}^T \mathbf{s}_{ui}, 0 \right) \right)^2, \tag{12}$$

where the label y_i indicates whether $i \in \mathcal{T}(u)$ or not. The rationale behind choosing the objective function (12) is that a good aggregation function should at least give high rankings to tracks that are already known to exist in the playlist, while the L2 regularization term can prevent trivial solutions.

Our reweighting scheme presented here could be viewed as a variant of the Sparse LInear Method (SLIM) [13, 14, 24] for top-N recommender systems, albeit with some important differences:

- SLIM directly operates on the binary relationship matrix between users and items (lists and tracks in our case), thus all observed items (tracks) are assumed to be equally relevant, which may not be the best choice in the binary CF setting [22]. Instead, our formulation (12) adopted the calculated CF score (8) as features to further distinguish among observed tracks;
- SLIM learns the aggregation weights by solving a regression problem that attempts to reconstruct the binary relationship matrix, we instead opt to solve a binary classification problem, which performs better during the experiments;
- To further improve interpretability and avoid overfitting, additional constraints such as non-negativity are imposed in the SLIM formulation. These constraints were discarded in our approach as they did not improve the accuracy of APC recommendation during the experiments, and also would lead to large-scale quadric programming problems that cannot be efficiently solved by off-the-shelf SVC solvers such as LIBLINEAR [7].

3.4 Discriminative Reranking

From (7), it can be seen that similarity between tracks used in previous sections is measured purely using the frequency that they co-occur in observed playlists. However, the notion of "similarity" is quite general, and there is virtually no limit to the type and quantity of similarity measures that we can encode into our model, and suitable combination of these measures may further improve the result. Motivated by these considerations, we adopted a two-stage approach similar to the one proposed in [5] to improve the recommendation performance: a candidate set is firstly generated using the scoring function (8) (In our experiments we fixed the size of candidate sets as 1000), the relative importance among candidates was then fine-tuned using a discriminative objective function and a richer set of features for each pair of candidate track and the target list. The additional features that we incorporated include:

- Categorical features, such as the Spotify Uniform Resource Identifiers (URIs) of tracks, albums, and artists.
- Word2Vec-based features. We borrowed ideas from [2, 9], and treated each track as a word and each list as a sentence. The feature "w2v_track_sim" was then constructed by firstly performing Word2Vec [12] to learn embeddings for the tracks, and then by computing the average cosine similarity between a candidate track and observed songs in the target list. The features "w2v_artist_sim" and "w2v_album_sim" were constructed in a similar vein by embedding artists and albums instead.
- "track_title_sim", "artist_title_sim" and "album_title_sim", where we adopted the Python difflib module to compute the similarities between the title of the target list and the artist/album/track name of a candidate song.
- Miscellaneous other features, such as "track_popularity" which measures the number that a candidate song occurs in the MPD data, and "cf_rank" which is obtained by transforming the raw CF ranking score into relative rank.

With all features constructed, we cast APC recommendation as a binary classification problem where the learning objective becomes accurate classification of whether a specific track belongs to the list that needs to be continue. A wide variety of nonlinear/linear algorithms can be explored to fit the classification function. In this competition, we specifically adopted gradient boosting decision trees (GBDT) [8], a powerful machine-learning technique that has a wide range of successful applications [11, 23]. We used the software package LightGBM [10] for GBDT model fitting, as it scales well and could elegantly handle the numerous categorical features such as track_uri and artist_uri, without the requirement of cumbersome preprocessing steps such as one-hot encoding.

4 RESULTS

Due to limitation of computational resources, we could not fully test the performance of reweighting and reranking approaches for the 10 subtasks before the competition deadline. The algorithms that we managed to apply are separately listed for each subtask in Table 1. In order to tune the hyperparameters used in each algorithm, 90% of playlists in MPD were randomly selected and utilized for training, with the remaining lists used for constructing validation set based on the specifications of the corresponding subtasks. The leaderboard results that we achieved are listed in Table 2, which shows that combination of the discussed algorithms could significantly improve the recommendation performance. The overall solution achieved 6th position on the public leaderboard. For the GBDT model learned via LightGBM, we also report importance of chosen features in Table 3. These values correspond to numbers of times that tree splits occur on each feature when the data is trained.

5 CONCLUSIONS

We have presented our solution to the APC recommendation problem posed by the RecSys 2018 challenge. The best recommendation solution that we found is a hybrid of neighborbased CF and discriminative postprocessing strategies. To

Table 2: Experiment results of online evaluations.

Method	R-Precision	NDCG	nClicks
CF	0.1972	0.3571	2.3732
CF+Reweighting	0.2112	0.3738	2.3244
CF+Reweighting+Reranking	0.2167	0.3814	2.1930

Table 1: Algorithms that we applied to each subtask.

Subtask	Neighbor-based CF	+Reweighting	+Reranking
Name only	Yes	No	No
Name and the first track	Yes	No	Yes
Name and the first 5 tracks	Yes	No	Yes
First 5 tracks(no name)	Yes	No	No
Name and the first 10 tracks	Yes	No	Yes
First 10 tracks(no name)	Yes	No	No
Name and the first 25 tracks	Yes	Yes	No
Name and 25 random tracks	Yes	Yes	No
Name and the first 100 tracks	Yes	Yes	No
Name and 100 random tracks	Yes	Yes	No

Table 3: Feature importance from LightGBM.

Feature	Importance
Track_popularity	3299
Artist_title_sim	2310
W2v_album_sim	2087
CF_rank	2133
CF_score	1792
W2v_track_sim	1631
Album_title_sim	1483
W2v_artist_sim	1464
Artist_uri	1014
Album_uri	684
Track_title_sim	471
Track_uri	151

further improve the performance, one possibility is to perform more extensive feature engineering for the reranking algorithm. Another interesting area for future work is to explore a learning framework that more closely aligns with the target evaluation metrics.

REFERENCES

[1] Fabio Aiolli. 2013. Efficient top-n recommendation for very large scale binary rated datasets. In *Proceedings of the 7th ACM conference on Recommender systems*. ACM, 273–280.

[2] Da Cao, Liqiang Nie, Xiangnan He, Xiaochi Wei, Shunzhi Zhu, and Tat-Seng Chua. 2017. Embedding factorization models for jointly recommending items and user generated lists. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 585–594.

[3] Chih-Ming Chen, Ming-Feng Tsai, Yu-Ching Lin, and Yi-Hsuan Yang. 2016. Query-based music recommendations via preference embedding. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 79–82.

[4] Shuo Chen, Josh L Moore, Douglas Turnbull, and Thorsten Joachims. 2012. Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 714–722.

[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 191–198.

[6] Sander Dieleman. 2016. Deep learning for audio-based music recommendation. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS'16). ACM*.

[7] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research* 9 (2008), 1871–1874.

[8] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[9] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. 2015. E-commerce in your inbox: Product recommendations at scale. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1809–1818.

[10] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*. 3149–3157.

[11] Jianxun Lian, Fuzheng Zhang, Min Hou, Hongwei Wang, Xing Xie, and Guangzhong Sun. 2017. Practical Lessons for Job Recommendations in the Cold-Start Scenario. In *Proceedings of the Recommender Systems Challenge 2017*. ACM, 4.

[12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[13] Xia Ning and George Karypis. 2011. Slim: Sparse linear methods for top-n recommender systems. In *2011 11th IEEE International Conference on Data Mining*. IEEE, 497–506.

[14] Xia Ning and George Karypis. 2012. Sparse linear methods with side information for top-n recommendations. In *Proceedings of the sixth ACM conference on Recommender systems*. ACM, 155–162.

[15] Martin Pichl, Eva Zangerle, and Günther Specht. 2015. Towards a context-aware music recommendation approach: What is hidden in the playlist name?. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*. IEEE, 1360–1365.

[16] Martin Pichl, Eva Zangerle, and Günther Specht. 2016. Understanding playlist creation on music streaming platforms. In *Multimedia (ISM), 2016 IEEE International Symposium on*. IEEE, 475–480.

[17] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. 2018. Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval* 7, 2 (2018), 95–116.

[18] Andreu Vall, Massimo Quadrana, Markus Schedl, Gerhard Widmer, and Paolo Cremonesi. 2017. The importance of song context in music playlists. In *Proceedings of the Poster Track of the 11th ACM Conference on Recommender Systems (Rec-Sys), RecSys*, Vol. 17.

[19] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *Advances in neural information processing systems*. 2643–2651.

[20] Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. 2017. A Meta-Learning Perspective on Cold-Start Recommendations for Items. In *Advances in Neural Information Processing Systems*. 6907–6917.

[21] K. Verstrepen, K. Bhaduriy, B. Cule, and B. Goethals. 2017. Collaborative filtering for binary, positiveonly data. *ACM SIGKDD Explorations Newsletter* 19, 1 (2017), 1 – 21.
[22] Maksims Volkovs and Guang Wei Yu. 2015. Effective latent models for binary feedback in recommender systems. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 313–322.
[23] Qiang Wu, Christopher JC Burges, Krysta M Svore, and Jianfeng Gao. 2010. Adapting boosting for information retrieval measures. *Information Retrieval* 13, 3 (2010), 254–270.
[24] Yong Zheng, Bamshad Mobasher, and Robin Burke. 2014. CSLIM: Contextual SLIM recommendation algorithms. In *Proceedings of the 8th ACM Conference on Recommender Systems*. ACM, 301–304.