Automatic Playlist Continuation using Subprofile-Aware Diversification

Mesut Kaya
Insight Centre for Data Analytics
University College Cork, Ireland
mesut.kaya@insight-centre.org

Derek Bridge
Insight Centre for Data Analytics
University College Cork, Ireland
derek.bridge@insight-centre.org

ABSTRACT

The *ACM RecSys Challenge 2018* involves the task of automatic playlist continuation (APC), aiming to help users to create and extend their own music playlists. In this paper, we explain *teamrozik*'s approach to the Challenge. Our approach to APC is twofold: Cold-Start-APC for short playlists and SPAD-APC for other playlists. Cold-Start-APC is a rudimentary popularity-based recommender. SPAD-APC treats playlists as if they were user profiles. It builds an implicit matrix factorization model to generate initial recommendations. But it re-ranks those recommendations using SubProfile-Aware Diversification (SPAD), which is a personalized intent-aware diversification method. The SPAD re-ranking method aims to ensure that the final set of recommendations covers different interests or tastes in the playlists of the users, which we refer to as subprofiles. We show that such subprofiles do exist within playlists and we show that the SPAD method achieves higher precision than matrix factorization alone.

KEYWORDS

Music Recommender; playlists; automatic playlist continuation; diversity; subprofiles.

ACM Reference format:
Mesut Kaya and Derek Bridge. 2018. Automatic Playlist Continuation using Subprofile-Aware Diversification. In *Proceedings of the ACM Recommender Systems Challenge 2018, Vancouver, BC, Canada, October 2, 2018 (RecSys Challenge '18),* 6 pages.
DOI: 10.1145/3267471.3267472

1 INTRODUCTION

The emergence of online music streaming services like Spotify, Pandora, Deezer, Apple Music and Amazon Music has increased the value of research related to music recommendation. Although music recommender systems often successfully recommend songs that satisfy users, in the sense of fitting the users' preferences, there are still a lot of challenges to be tackled [8]. Automatic playlist continuation (APC) is one such challenge. The aim in APC is to help users to create and extend their own playlists.

RecSys Challenge '18, Vancouver, BC, Canada

DOI: 10.1145/3267471.3267472

The *ACM RecSys Challenge 2018,* organized by Spotify, the University of Massachusetts and Johannes Kepler University is all about APC. Using datasets of playlists made available by Spotify, participants build systems to automatically predict tracks that are missing from test playlists.

In the rest of this paper, we give an overview of the Challenge, then describe our approach to the Challenge in detail. We explain the resources we used and our experimental methodology, and then give some results.

2 CHALLENGE OVERVIEW

The Challenge focuses on music recommendation, specifically automatic playlist continuation (APC) [2]. The task is to recommend appropriate tracks to add to a playlist.

For this task, Spotify released the Million Playlist Dataset[1] (MPD), containing 1,000,000 playlists created by Spotify users. Some statistics about the MPD can be found in Table 1. Each playlist is represented by playlist metadata (the title, description if available, the number of unique tracks, etc.) and the tracks in the playlist, together with metadata about the tracks (such as the name of the track, album and artist).

Spotify also released the Challenge Set, comprising 10,000 incomplete playlists (i.e. some of the tracks are hidden). Specifically, in the Challenge Set, the number of seed tracks in a playlist has values from the set $\{0, 1, 5, 10, 25, 100\}$. Furthermore, the way in which the seed tracks were chosen (e.g. initial tracks or random tracks) and the availability of a playlist title mean that there are 10 different categories of playlists, with 1,000 playlists per category. This is summarized in Table 2.

The task is to recommend 500 tracks to each of the playlists in the Challenge Set. These recommendations are evaluated by Spotify against the ground-truth G, i.e. the tracks that were actually hidden. The evaluation metrics are precision@$|G|$, NDCG and a bespoke measure called Clicks that is based on rank within the ground-truth, combined by a Borda count.

However, this RecSys Challenge is split into two sub-challenges. In the Main Challenge, participants must train their prediction models exclusively on the MPD. In the Creative Challenge, participants are allowed to use public external data sources. Although we do not use any external data, we build our model using the union of the MPD and the Challenge Set. This means that our system is obliged to compete in the Creative Challenge.[2]

[1] https://recsys-challenge.spotify.com/

[2] On the RecSys Challenge forum, in reply to the question "[is it] allowed to include the information in the challenge set for the model training?", the reply was: "The rules say that for the main track you can only use the MPD. The challenge set is not part of the MPD, so for the main track, the answer is 'no'. For the creative track you can use publicly available data and the challenge set qualifies as that — so for the creative track,

Table 1: MPD statistics

# of playlists	1,000,000
# of tracks	66,346,428
# of unique tracks	2,262,292
# of unique albums	734,684
# of unique artists	295,860
# of unique titles	92,944
# of unique normalized titles	17,381
# of playlists with description	18,760
avg. playlist length	66.3

Table 2: Challenge Set statistics

# of playlists	title	tracks	cold-start
1000	✓	none	✓
1000	✓	first track	✓
1000	✓	first 5 tracks	X
1000	X	first 5 tracks	X
1000	✓	first 10 tracks	X
1000	X	first 10 tracks	X
1000	✓	first 25 tracks	X
1000	✓	random 25 tracks	X
1000	✓	first 100 tracks	X
1000	✓	random 100 tracks	X

In the next section, we explain our approach in detail.

3 OUR APPROACH

Let I be the set of all items (i.e. tracks) in the union of the MPD Set and Challenge Set. A playlist, which we will designate by u, is a *set* of items, $u \subseteq I$. Let title(u) be the title of playlist u, if it has one (or $\perp$ if it does not have a title). Let U_{MPD} be the set of playlists in the MPD and U_{CS} be the set of playlists in the Challenge Set; then $U = U_{MPD} \cup U_{CS}$. The *candidate* items that can be added to a given playlist $u \in U_{CS}$ are all the items in I less those that are already in u: $I \setminus u$. We compute recommendations in one of two ways, depending on the length of u. The first way we refer to as the Cold-Start-APC; the second we refer to as SPAD-APC. Our implementation is publicly available.[3]

We consider playlists $u \in U_{CS}$ to be cold-start playlists if they have a title and either zero or one track; see the last column in Table 2. We present Cold-Start-APC in Section 3.1. For the remaining 8,000 playlists in U_{CS}, candidates are scored by a matrix factorization algorithm and then re-ranked using a novel diversification technique, which we call SPAD, from the first author's Ph.D. research [6]; see Section 3.2.

Before presenting details of the two approaches, we make a few additional observations. First, we do not, in fact, use the full set of items I. As we explain in Section 4, we exclude some items from I to improve compute-times. Second, it follows from the formulation given above, that we are not using any metadata, except for the title of the playlist, where available (and, in fact, this is only used by Cold-Start-APC, not by SPAD-APC). Third, since we treat playlists as *sets* of items, for this prediction task we are ignoring the ordering of the items in the playlist. There is some debate about the significance of ordering in playlists. For example, Schedl et al. think that it is important [8] and there are approaches that take it into account (e.g. [1]). But according to Tintarev et al., there is actually little evidence that the exact order of the tracks matters to users [9]. In any case, in our solution we are not using the ordering. Fourth, we are, in effect, treating playlists in the way that a regular recommender would treat a user profile (and this explain why we designate them by u). In a regular recommender, a positive-feedback-only implicit ratings profile would just be a set of items: the ones the user likes; in our recommender, a playlist is just a set of items.

3.1 Cold-Start-APC

We use Cold-Start-APC for the 1000 playlists in the Challenge Set that have title only and the 1000 playlists that have a title and one track (their first).

For each candidate $i \in I \setminus u$, Cold-Start-APC computes a score, $s(u, i)$, based on the popularity of i across the playlists in U_{MPD}, and recommends the 500 candidate items that have the highest scores. To improve the scoring, we use a 'normalization' function provided by Spotify which converts track titles to lowercase and removes punctuation symbols. As shown in Table 1, 92,944 different titles become 17,318 unique titles after normalization.

For a cold-start playlist u, the predicted score $s(u, i)$ for a candidate track $i \in I \setminus u$ is computed as follows:

$$s(u, i) = \sum_{v \in U_{MPD}} \mathbb{1}(u, i, v) \tag{1}$$

In the case where u has a title but no tracks:

$$\mathbb{1}(u, i, v) = \begin{cases} 1 & \text{if } i \in v \text{ and title}(u) = \text{title}(v) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

In the case where u has a title and one track:

$$\mathbb{1}(u, i, v) = \begin{cases} 2 & \text{if } i \in v \text{ and } u \subseteq v \text{ and title}(u) = \text{title}(v) \\ 1 & \text{if } i \in v \text{ and } (\text{title}(u) = \text{title}(v) \text{ xor } u \subseteq v) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Note that, for some of the playlists with title only, it can be the case that the recommender cannot recommend 500 tracks with non-zero scores. In this case, we fill the rest of the recommendations with the most popular tracks in U_{MPD}.

There is no doubt that our cold-start solution is rudimentary, and there are many ways it could be improved, perhaps especially by using external data sources.

3.2 SPAD-APC

Our approach for the remaining 8000 playlists in the Challenge Set, having at least 5 tracks each, is based on the first author's Ph.D. work [6]. We have published early versions of this work [3, 4] but we have refined it subsequently and the description here is based on the latest version. The goal of the work is to generate a set of recommendations where each recommendation is relevant but the set of recommendations is diverse. Our approach to this is called SubProfile Aware Diversification (SPAD) and we are finding,

the answer is 'yes'." https://groups.google.com/forum/#!topic/recsyschallenge-2018/1F-QCl2se4E

[3] https://github.com/mesutkaya/SpotifyRecSysChallenge2018

across multiple datasets, that SPAD consistently improves both precision and diversity. This is notable because many approaches to diversification trade-off accuracy for diversity.

It is worth asking first: are there reasons for thinking that diversity will be helpful for APC? Lee et al. report the results of a user study into playlists that were generated automatically using content-based similarity [5]. A common concern among the participants in the user study was that consecutive tracks in the playlists were too similar; they also complained about a lack of variety in the playlists. Interpretations of 'variety' differed from user to user, e.g. variety in terms of genres, styles, artists, etc. A tentative implication is that users like playlists to have some diversity and, if creating their own playlists (like the ones provided by Spotify), they might aim to give them a level of diversity. If this tentative implication is correct, then there are *a priori* reasons to try a diversification technique in this APC Challenge.

Before explaining how we use SPAD for the task of APC, we give an overview of SPAD in the next section.

3.2.1 SPAD. SPAD takes a greedy re-ranking approach to diversification [11]. It starts with a recommendation set *RS*, generated by a baselline recommender (e.g. matrix factorization). Each item in *RS* has a score $s(u, i)$, computed by the baseline recommender. In greedy re-ranking, a ranked list of recommendations *RL* is built by iteratively selecting items from *RS* based on an objective function that balances their score with their diversity with respect to the already-selected members of *RL*:

$$f_{obj}(i, RL) = (1 - \lambda)s(u, i) + \lambda \,\mathrm{div}(i, RL) \tag{4}$$

SPAD is novel in the way in which it measures diversity.

In the past, the diversity of a set of recommendations was given by the average all-pairs dissimilarity between the items in the set. In other words, items are chosen so that they are different from each other. More recently, under the name of Intent-Aware Diversification, items have been chosen in such a way as to cover the user's interests, as revealed by her profile. Most commonly, this is done using item features [10]. For example, if her profile contains lots of jazz, a little punk rock, and a modest amount of thrash-metal, then *RL* can be constructed from *RS* in a way that preserves the distribution of interests in the profile. SPAD is a form of intent-aware diversification but it does not rely on item features, which are sometimes not available or may not fully represent a user's interests. Instead, we compute *subprofiles* of a user's profile. In a positive-feedback-only implicit ratings scenario, a profile is just a set of items that a user likes, and a subprofile is a subset of those items, intended to capture one of the user's tastes or interests.

Let u be the user's profile, let $\mathcal{S}_u$ denote all the subprofiles of u: each subprofile $S \in \mathcal{S}_u$ is simply a subset of u, $S \subseteq u$. Then, the set *RS* is greedily re-ranked using the objective function given as Equation(4) with $div(i, RL) = \mathrm{div}_{\mathrm{SPAD}}(i, RL)$, where:

$$\mathrm{div}_{\mathrm{SPAD}}(i, RL) = \sum_{S \in \mathcal{S}_u} [p(S|u)p(i|u, S) \prod_{j \in RL} (1 - p(j|u, S))] \tag{5}$$

$p(S|u)$ is estimated as:

$$p(S|u) = \frac{|S|}{\sum_{S' \in \mathcal{S}_u} |S'|} \tag{6}$$

$p(i|u, S)$, the probability of choosing i from a set of recommendations *RS* given subprofile S of user u, is estimated as:

$$p(i|u, S) = \frac{\mathbb{1}(i, S)s(u, i)}{\sum_{j \in RS} \mathbb{1}(j, S)s(u, j)} \tag{7}$$

where:

$$\mathbb{1}(i, S) = \begin{cases} 1 & \text{if } i \in \bigcup_{j \in S} \mathrm{KNN}(j) \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where $\mathrm{KNN}(j)$ is the set of j's k-nearest-neighbours in I. In other words, i must be a neighbour of a member of S.

What this does not yet explain is how we compute the subprofiles. This is the part of our work that has undergone most refinement. Early versions are in [3, 4]. More recently, we take the following approach. We create a candidate subprofile for each $i \in u$. The candidate subprofile for $i \in u$ contains i itself and also $j \in u$ if j's nearest-neighbours contain i, i.e. the candidate subprofile for i is $\{j | j \in u, i \in \mathrm{KNN}(j), i \neq j\} \cup \{i\}$. $\mathrm{KNN}(j)$ is computed by selecting the top-k similar items to i based on a similarity score, $sim(i, j)$, for which here we are using cosine similarity on rating vectors. Candidate subprofiles are pruned to a final set of subprofiles by excluding those that are wholly contained in any of the others. Different subprofiles can be of different lengths; the number of subprofiles differs from user to user; but there can be no more than $|u|$ subprofiles. See [6] for details.

3.2.2 SPAD for APC. We take the idea of SPAD and apply it to APC. As already mentioned, we treat each playlist $u \in U$ as if it were a user's profile. Our baseline recommender (whose recommendations get re-ranked by SPAD) is a fast alternating-least-squares (ALS) implementation of matrix factorization for implicit and explicit datasets [7]. We chose this as our baseline because in our previous work [4] for different datasets it was the most accurate baseline recommender.

Ordinarily, SPAD diversifies a set of recommendations to cover the different tastes (subprofiles) that we extract from a user's profile. It is not obvious that a playlist will similarly contain different tastes and therefore not obvious that SPAD-style diversification will be of benefit to the APC task. Evidence of the benefit is given in Section 6.

4 RESOURCES

Some of our decisions were constrained by available resources, which are explained here.

By the rules of the Challenge, participants could submit only one set of predictions for evaluation per day. We started working (part-time) on the Challenge in its last three weeks, so this gave us a very small number of opportunities to test the performance of our approach on the Challenge Set. However, this did motivate us to create a validation set (see next section) so that we could test algorithm variants and find good values for hyper-parameters. Not only was this expedient, we hope that it helped us to avoid overfitting our solution to the Challenge Set.

We ran our algorithm on a personal laptop with a 2.5 Ghz Intel Core i7 and 24 GiB memory. Running experiments using a laptop for this large dataset was challenging, especially for the matrix factorization algorithm. It took some time to prepare a daily submission to the public leaderboard. This in turn constrained the size of validation set that we were able to work with (next section).

It also resulted in us making a major decision. Before applying our approach, we eliminated all tracks that appear in only one MPD playlist. The number of unique tracks was cut from 2,262,292 to 1,189,252. Of course, this reduction in the item space improved runtimes considerably. But it has a negative effect on recommendation accuracy: since we can never recommend the eliminated tracks, we lose out (for resource reasons, rather than algorithmic reasons) if those tracks ever appear in the ground truth.

5 METHODOLOGY

The baseline matrix factorization algorithm that we use has two hyper-parameters: the number of latent factors k and a confidence level α. SPAD also has its own hyper-parameters, which are the number of neighbours while detecting the subprofiles (referred to here as K_1) and the number of neighbours of Equation (8) (referred to here as K_2). For the re-ranking in Equation (4), there is λ, controlling the balance between accuracy and diversity.

To find good values for these hyper-parameters, we used a validation set. We randomly selected 10,000 playlists. We kept 80% of their tracks as part of the training data and held out 20% as validation data. This is quite a small validation set (10,000 playlists compared with 1,000,000); it may not be representative enough to optimize hyper-parameter values. Its size was determined by the resources we had available (previous section).

We used a grid search. For k, α, K_1, and K_2, we tested values from $\{10, 20, 30, \ldots, 100\}$. For λ, we tested values from $\{0.1, 0.2, 0.3, \ldots, 1.0\}$. We selected the values that maximize precision, which were as follows: $k = 100$, $\alpha = 50$, $K_1 = 30$, $K_2 = 70$ and $\lambda = 0.4$.

6 RESULTS

We divide this section into three: first we analyze the subprofiles that we found in the playlists; then we give some experimental results on the validation set; finally, we use data from the public leaderboards to compare our team's performance with other teams.

6.1 Subprofile Analysis

We begin by asking: do playlists actually contain sub-tastes? Or, in other words, do playlists, when treated as if they were user profiles, contain subprofiles? If they do not, then there is little prospect that SPAD will work well on the APC task. We show some data here that suggests that playlists do contain sub-tastes (subprofiles).

We applied SPAD's subprofile detection method (explained in Section 3.2) to the 8,000 playlists in the Challenge Set that have at least 5 tracks. Figure 1 is a histogram showing the frequencies of different numbers of subprofiles. In other words, it shows how many of the 8,000 playlists contain just one subprofile, how many contain two subprofiles, how many contain three, and so on.

If a playlist has just one subprofile, then that subprofile comprises the whole playlist, and this would be a playlist with no sub-tastes. From the histogram, we see that the number of playlists that fall into this category (i.e. ones with just one subprofile) is very small. Perhaps surprisingly, playlists do contain sub-tastes. Four subprofiles is most frequent, but some playlists have as many as 100.

What also matters is how long these subprofiles are, and this is shown in Figure 2. From the histogram, we see that some are very short: even a single song where that song is not a good enough neighbour to other songs in the playlist for it to join their subprofile. But most subprofiles comprise two or more tracks.

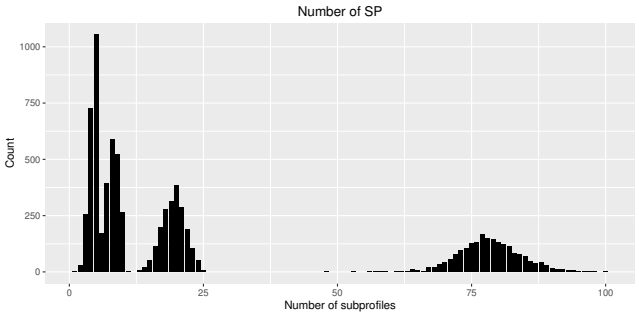

Figure 1: Number of subprofiles

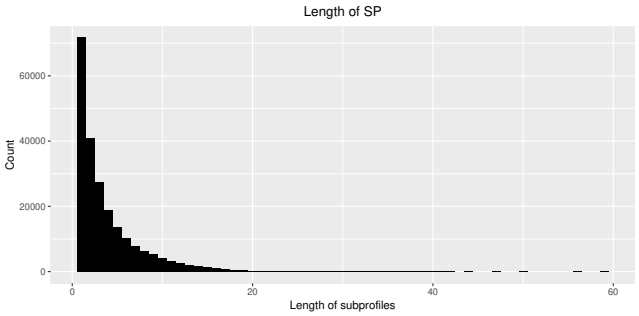

Figure 2: Subprofile length

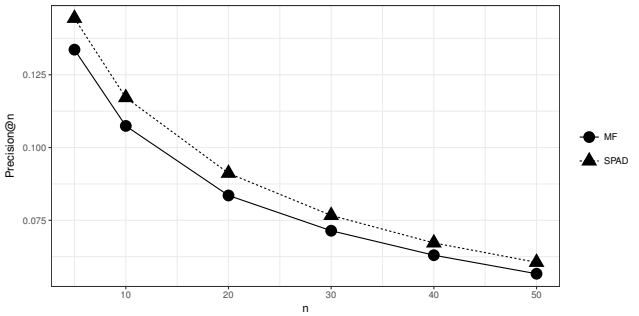

Figure 3: Precision@n on validation set by n

6.2 Validation set results

Here we show a few validation set results that confirm that SPAD re-ranking provides an advantage over using just its baseline matrix factorization algorithm.

In Figure 3, for all 10,000 playlists in the validation set, we plot Precision@n for different values of n. It can be seen that re-ranking the matrix factorization recommendations using SPAD always increases precision over matrix factorization.

We also look at the precision for different playlist sizes and for different numbers of subprofiles. For the 10,000 playlists in the

Table 3: Creative Challenge leaderboard

rank	team name	RPREC	RPREC rank	NDCG	NDCG rank	Clicks	Click rank	Borda
1	vl6	0.2234	1	0.3939	1	1.7845	1	90
2	Creamy Fireflies	0.2197	2	0.3846	2	1.9252	4	85
3	KAENEN	0.209	3	0.3746	3	2.0482	6	81
4	cocoplaya	0.2022	7	0.3656	6	1.8377	2	78
5	BachPropagate	0.2024	6	0.3659	5	2.0029	5	77
6	Trailmix	0.2059	4	0.3703	4	2.2589	9	76
7	**teamrozik**	0.2055	5	0.3609	7	2.1636	8	73

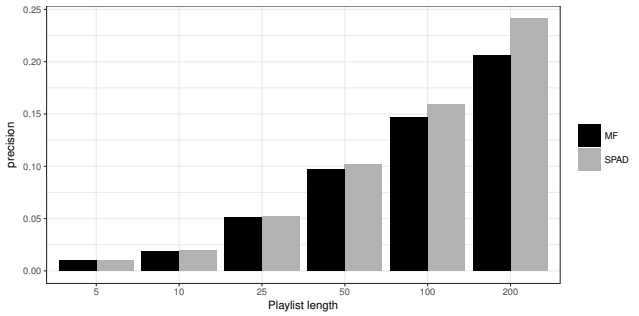

Figure 4: Precision@10 on validation set by playlist length

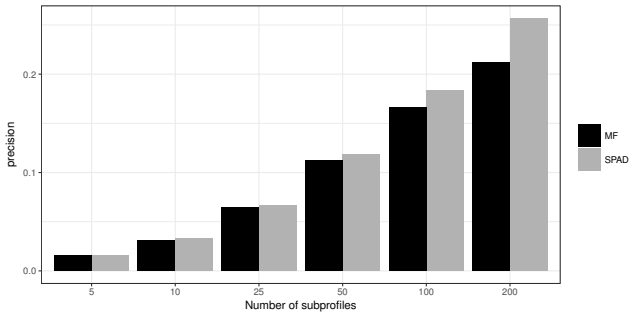

Figure 5: Precision@10 on validation set by number of subprofiles

validation set, Figure 4 shows the Precision@10 of the baseline and of SPAD for playlists of up to 5 tracks, from 6 to 10 tracks, 11 to 25 tracks, 26 to 50 tracks, 51 to 100 tracks, and 101 to 200 tracks. Similarly, Figure 5 shows the Precision@10 for the validation set but this time showing results for playlists that have up to 5 subprofiles, from 6 to 10 subprofiles, 11 to 25, 26 to 50, 51 to 100 and 101 to 200.

The Figures show that, as the playlist length and the number of subprofiles increases, the amount by which SPAD outperforms matrix factorization also increases. The more songs there are in a playlist, the more subprofiles there are, but also the more precision benefits from SPAD re-ranking. Arguably, the Figures also show that for the shortest playlists, precision for both SPAD and matrix factorization is so low that we might have benefited instead from applying a cold-start strategy to more playlists than we did.

6.3 Challenge Set results

Among 32 teams, *teamrozik* came seventh in the Creative Challenge. Table 3 shows the final leaderboard published by the Challenge organizers, which evaluates the teams on all of the Challenge Set.

7 CONCLUSION

In this paper, we presented our approach to the *ACM RecSys Challenge 2018*, in which the task is automatic playlist continuation (APC). For cold-start playlists, we used a popularity recommender. For the remainder of the playlists, we built a model using matrix factorization to generate sets of recommendations. But our contribution is that we re-ranked those recommendations to increase their diversity. The approach that we used for re-ranking, SubProfile Aware Diversification (SPAD), is being developed in the first author's Ph.D. It seeks to extract subprofiles from a user's profile or, in this case, from a playlist, and then to ensure that the recommendations cover those subprofiles. Using a validation set, we showed that SPAD re-ranking results in more accurate recommendations. Our analysis supports the claim that user-generated playlists do contain subprofiles corresponding to different interests or tastes, and trying to cover those subprofiles in the final set of recommendations produces more accurate recommendations.

Due to resource limitations (Section 4), we excluded a large number of songs from the dataset on which we built our model. Being unable to recommend those songs will have negatively affected our precision results. Furthermore, in the time that we allowed ourselves for working on the Challenge, we were not able to improve the method we use for cold-start playlists. Our method is quite rudimentary and easily improved, e.g. to use string similarity instead of exact matching of playlist titles.

While SPAD has proven successful for the APC task, it is designed for making recommendations in general. A music streaming service, such as Spotify, could apply it to whole user profiles (instead of playlists), in which case it should produce accurate and diverse sets of recommendations, covering different tastes and interests within the user's profile.

ACKNOWLEDGMENTS

This paper emanates from research supported by a grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289 which is co-funded under the European Regional Development Fund.

REFERENCES

[1] Claudio Baccigalupo and Enric Plaza. 2006. Case-based sequential ordering of songs for playlist recommendation. In *Proceedings of the 8th European Conference on Case-Based Reasoning*. Springer, 286–300.

[2] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. 2018. RecSys Challenge 2018: Automatic Music Playlist Continuation. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*. ACM, New York, NY, USA.

[3] Mesut Kaya and Derek Bridge. 2017. Intent-Aware Diversification using Item-Based SubProfiles. In *Procs. of the Poster Track of the 11th ACM Conference on Recommender Systems*, Domonkos Tikk and Pearl Pu (Eds.). CEUR Workshop Proceedings, vol-1905.

[4] Mesut Kaya and Derek Bridge. 2018. Accurate and Diverse Recommendations Using Item-Based SubProfiles. In *Proceedings of the Thirty-First International Florida Artificial Intelligence Research Society Conference, FLAIRS 2018, Melbourne, Florida, USA. May 21-23 2018*. 462–467.

[5] Jin Ha Lee, Bobby Bare, and Gary Meek. 2011. How Similar Is Too Similar?: Exploring Users' Perceptions of Similarity in Playlist Evaluation.. In *ISMIR*. Citeseer, 109–114.

[6] Mesut-Kaya. forthcoming. *Subprofile Aware Diversificaton of Recommendations*. Ph.D. Dissertation. University College Cork.

[7] István Pilászy, Dávid Zibriczky, and Domonkos Tikk. 2010. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 71–78.

[8] 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.

[9] Nava Tintarev, Christoph Lofi, and Cynthia Liem. 2017. Sequences of Diverse Song Recommendations: An exploratory study in a commercial system. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*. ACM, 391–392.

[10] Saul Vargas, Pablo Castells, and David Vallet. 2011. Intent-oriented diversity in recommender systems. In *Procs. of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 1211–1212.

[11] Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. 2005. Improving recommendation lists through topic diversification. In *Procs. of the 14th International Conference on World Wide Web*. ACM, 22–32.