Spotify Me: Facebook-Assisted Automatic Playlist Generation

Arthur Germain† and Jacob Chakareski⋆
†*Ecole Polytechnique Fédérale de Lausanne (EPFL);* ⋆*University of Alabama*

***Abstract*—We design a novel method for automatically generating a playlist of recommended songs in the popular social music sharing application *Spotify* that are liked with high probability by a user. Our method employs multiple seed artists as an input that are obtained via the *Facebook* likes of artists and the listening history of songs of a Spotify user. First, we construct an input vector comprising all the artists that the user likes on Facebook and listens to in Spotify. Then, we search for other artists and bands related to them using *EchoNest*, an online state-of-the-art machine learning platform. We assign a score to every artist in the thereby obtained collection, based on the frequency of his/her appearance. Finally, we construct a playlist comprising randomly selected popular songs associated with the most frequently cited artists. We examine the recommendation performance of our algorithm by computing its WTF score (fraction of disliked songs) and novelty factor (fraction of new liked songs) on playlists generated for different seed input sizes. We observe that our approach substantially outperforms the built-in *Spotify Radio* recommender. On 30 song playlists, we are able to improve the WTF score by 49% and the novelty factor by 42%, on average. Due to its general design, our method is broadly applicable to a variety of personal content management scenarios.**

I. Introduction

With the emergence of popular music streaming services, e.g., Spotify [1] or Deezer [2], everybody has access now to extensive collections of songs in just one click. The volume of available content is overwhelming, which motivates the need for automatic recommendations via a suitably generated playlist. There are different approaches to playlist recommendation existing in practice. For instance, Apple's Genius application [3] builds a playlist from a single seed song by selecting similar songs in the user's local library, while the web radio application, Pandora [4], creates a custom radio station from a single seed song. Similarly, the popular social music sharing platform Spotify has a playlist engine called *Spotify Radio* [5], which recommends songs based on a single track or a bag of tracks. Existing recommender systems exhibit two shortcomings. First, most of their recommendations are very similar to what the user has already listened to. Second, they are limited to generating playlists exclusively from a single seed input, as otherwise their recommendation accuracy becomes quite poor. We overcome these drawbacks by designing a playlist generation method that suitably integrates a content novelty factor and multiple seed input into its operation. Its effectiveness is demonstrated against *Spotify Radio* that it outperforms by a wide margin. Next, we describe existing recommendation approaches in greater detail.

II. Related work

Automatic playlist engines have been proposed for the first time in [6]. Since then, extensive research has been carried out to improve their performance and design them to operate automatically. We define a playlist as a collection of songs meant to be listened together. There is no concrete definition of what comprises a good playlist. Still, several important aspects need to be taken into consideration [7]: the character of the songs, their playback order, and the transitions between them. Furthermore, Leong et al. [8] showed that users expect to have a variety in their playlists, i.e., they should not be a bag of similar tracks, and randomness in the tracks' order matters.

There are two approaches to playlist generation. The first is collaborative filtering, used by the popular web service *last.fm* [9] and Apple's Genius [10]. The second is content-based filtering, employed by the web radio *Pandora* [11]. Many recommendation engines combine both approaches [12] to enhance performance. Interestingly, the authors in [10] have shown that Genius can capture song-specific aspects of acoustic similarity, similarly to content-based systems. This is achieved by removing the evaluators' bias caused by their familiarity with the recommended songs' titles and artists.

A. *Collaborative filtering*

Collaborative filtering compares the musical tastes of different users [13]. Such systems record every song and artist a user listens to and look for overlaps in the musical records of others. They are then able to suggest new songs or artists based on the degree of overlap between users. This approach is illustrated in Figure 1. The *last.fm* web radio employs collaborative filtering to recommend new content to users based on their musical preferences specified in their profiles.

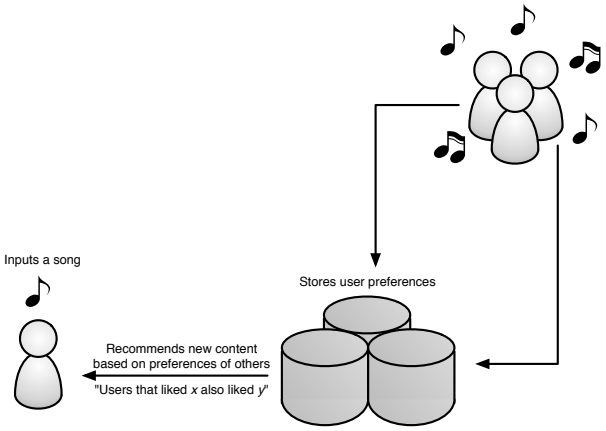


Fig. 1. Collaborative filtering

B. *Content-based filtering*

Content-based filtering examines the content of existing songs in order to recommend similar ones to the user. Song characteristics such as tempo, melody, rhythm, and instruments are analyzed and similar content is proposed using a nearest-neighbor algorithm. The process is visually described in Figure 2. *Pandora* employs this approach via its *Music Genome Project* [11]. Concretely, a team of trained musicians listens to every song and assigns a value to approximatively 400 attributes employed to characterize it. Songs are then classified by Pandora according to these attributes and recommended to users based on the classification.

C. *Hybrid methods*

Spotify Radio is the state-of-the-art representative of this category. It utilizes *EchoNest*, a machine learning platform for music data, developed in collaboration between Berkeley, Columbia and MIT

MMSP'13, Sept. 30 - Oct. 2, 2013, Pula (Sardinia), Italy.
978-1-4799-0125-8/13/$31.00

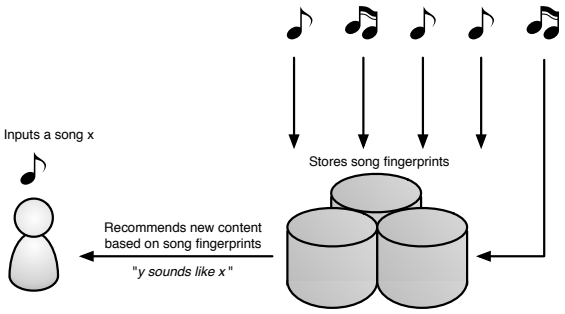


Fig. 2. Content-based filtering

over more than 12 years [14]. Its real-time API provides access to more than 5 billion data points that connect more than 30 million songs and 1.5 million artists that can be used to search for similar content. EchoNest classifies songs based on their fingerprints (i.e., tempo, key, rhythm, etc.) and metadata (i.e., blog entries, ratings, reviews, popularity, etc.). Given a seed song, *Spotify Radio* searches for a similar track via *EchoNest*. The process is then repeated sequentially on every song that is subsequently returned by EchoNest. Research articles that have examined combining social tags and audio signal information for content recommendation in music sharing sites include [15, 16].

Next, we describe our framework in Section III. Then, we evaluate its recommendation performance via experiments in Section IV. Finally, we conclude in Section V.

III. Playlist Generation Framework

Our algorithm utilizes the Facebook likes (of bands) and listening history (most popular performers) of a Spotify user. To study its advantages over the state-of-the-art *Spotify Radio*, we also utilize the EchoNest API in its operation. Our approach is fully automatic, i.e., a user does not need to explicitly declare his preferred artists. The playlist we generate is usually very eclectic and contains a lot of different artists that represent the user's tastes. In our implementation, the playlist size N_s is empirically set to 30, to correspond to a two-hour long playlist, on average. Our system is illustrated in Figure 3.

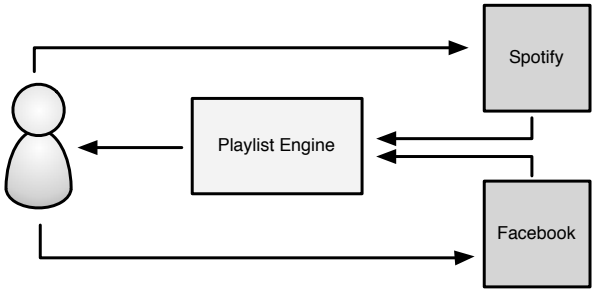


Fig. 3. A high-level block diagram of our framework.

A. Constructing the input vector

We fetch all artists that the user *likes* on Facebook and store them in a vector L_k. For every entry of L_k, we search for its influences on Facebook (an artist's influences are other artists that have been declared as influencing its music style). We then build an influence matrix I, where the first column stores all the artists liked on *Facebook* (the vector L_k), while the entries $I_{j,2}$ to I_{j,N_i+1} represent the N_i influences of artist j. Note that some entries $I_{j,2}$ to I_{j,N_i+1} can be empty, if the artist j did not specify any influences on Facebook. We empirically set N_i to three, as many bands do not cite more than three influences. Finally, we reshape the matrix I into a vector F_b column-wise.

Next, we look for all unique artists in the user's Spotify library and store them in a vector S_{p1}. In addition, we look for the ten artists that the user listened to the most recently and store them in a vector S_{p2}. The Spotify API can return a list of up to twenty most popular artists in the user's listening history. We empirically established that using more than the first ten does not lead to further notable improvement in recommendation performance. Finally, we merge the vectors F_b, S_{p1}, and S_{p2} into one input vector X. Only unique entries are kept in X. This process is summarized in Figure 4.

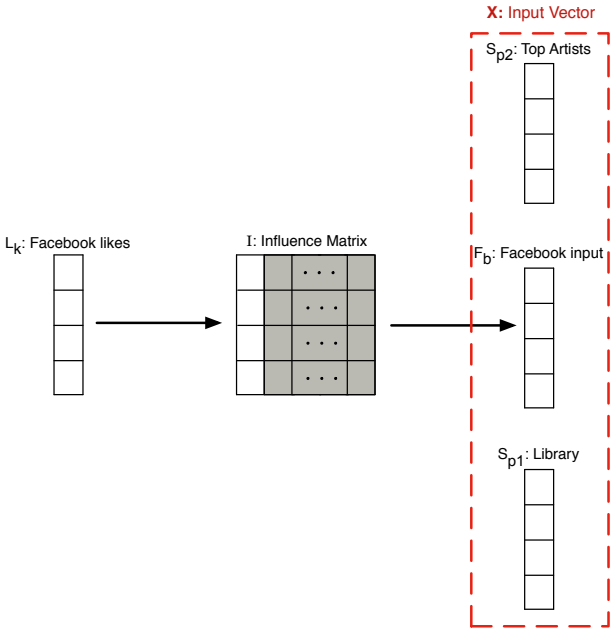


Fig. 4. Construction of the input vector

B. Recommended playlist generation

For every artist in X, we search for N_a similar artists using EchoNest [17] and store them in a matrix E such that:

$$X_i \xrightarrow{f_{EN}} E_{i,1\ldots N_a}\,, \tag{1}$$

where f_{EN} denotes the EchoNest function call that returns similar artists to a given input artist[1]. In our implementation, N_a is set to four, as it represents a good compromise between recommendation accuracy and processing time.

A matrix S is then created as a concatenation of X and E, as illustrated in Figure 5a, i.e.,

$$S = [X, E]\,, \tag{2}$$

where the first column of S represents the artists in X, and the remaining entries $S_{i,2}$ to S_{i,N_a+1} correspond to the artists similar to $S_{i,1}$, according to EchoNest (1).

We score every unique artist a in S according to the aggregate number of its occurrences in S, as illustrated in Figure 5b. In particular, when an artist appears in the first column of S, it means that it

[1]For instance, the following call returns artists similar to the band Sum 41: http://developer.echonest.com/api/v4/artist/similar?api_key=FILDTEOIK2HBORODV&name=Sum\%2041

was present in the input vector X. Therefore, the user likes this artist with a high probability. Hence, we weight the artists' occurrences in $S(:,1)$ by a factor $\alpha > 1$. In our experiments, we empirically select $\alpha = 2$, as it provides a good tradeoff between recommendation accuracy and recommendation novelty. Concretely, if α is set too high, it will cancel out prospective new recommendations, since they cannot be scored high enough in order to appear on the list of selected artists, as explained henceforth. If α is set too low (≈ 1), we can not differentiate the artists that are surely liked by the user, from possible new recommendations. Occurrences in columns $n > 1$ of the matrix S receive a score of one. The computation of the overall score is formally described below

$$Score(a) = \sum_{0<i<m} \alpha I_{\{S_{i,1}=a\}} + \sum_{\substack{0<i<m \\ 1<j<N_a+1}} I_{\{S_{i,j}=a\}}, \quad (3)$$

where m is the number of rows of the matrix S. Each unique artist's label (ID) and its score are stored in a two-column matrix A.

We sort the artists in A according to their score, in decreasing order. Let A_S denote the thereby obtained matrix of sorted artists. That is, $A_S(1,1)$ is the ID of the artist with the highest score, $A_S(2,1)$ is the artist with the second highest score, and so forth. The column $A_S(:,2)$ represents their corresponding score values, as illustrated in Figure 5c. We limit the size of A_S to N_s rows (artists). The last step is to select a song for each artist in A_S. The Spotify API allows developers to access the most popular songs of an artist. Thus, for every entry in $A_S(:,1)$, we randomly select a song from the three most popular ones of this artist, as returned by Spotify. This allows the playlist to be dynamic, since every time a new one is generated, songs or artists might be different.

Our method is formally described in Algorithm 1.

Algorithm 1 Automatic playlist generation with multiple seeds

Input initial vector X containing a set of artists
$S =$ echoNestSearch(X)
$A =$ scoreAndSelect(S)
$P =$ getPlaylist(A)
return P

Subroutine: echoNestSearch(X)

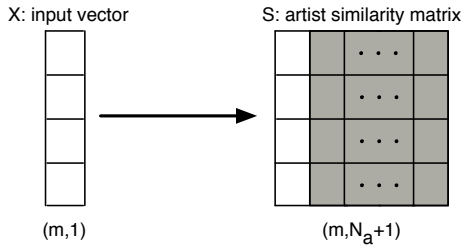
for $i = 1$ to $|X|$ **do**
  $E(X_i,:) = f_{EN}(X_i)$
end for
$S = [X, E]$
return S

Subroutine: scoreAndSelect(S)

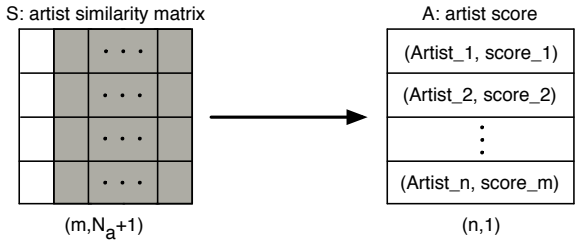
$U =$unique(S)
for every $u \in U$ **do**
  $s_u =$ Score(u)
  Store (u, s_u) in L
end for
$A =$ sortByScore(L,'descend')
$A = A$[1:N_s , :]

Subroutine: getPlaylist(A)

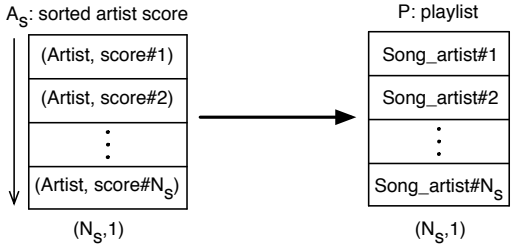
for every $a \in A(:,1)$ **do**
  Get most popular song using Spotify API
  Store in P
end for
return P



(a) Mapping from X to S: For every item i in X, look for N_a similar artists and store them in $S_{i,2}$ to S_{i,N_a+1}



(b) For every unique artist in S, compute (3) and store its score in A



(c) Sort the artists according to their score in A_s, look for one of their most popular songs, and add it to the playlist P.

Fig. 5. Different steps of our algorithm

IV. Experimental results

We employ the built-in recommender Spotify Radio as our reference method. Assessing the quality of a playlist is difficult and subjective. Ten participants were recruited for the testing, equally distributed among genders and with different music preferences (from Rock, Punk, and Pop to Reggaetón and Classical). We generated multiple input vectors X of different sizes (from 1 to 150 entries), by randomly down sampling the originally generated vector in Section III-A to that specific size. Then, we generated for every X a playlist using Spotify Radio and our approach. Finally, for each playlist we computed its WTF score and Novelty Factor, based on the participants' responses to the entries of the playlist.

The WTF score has been defined by Lamere in [18] as *counting the number of head-scratchers in a playlist*. In our case, we formally define WTF score as counting the number of dislikes in the generated playlist. The higher the WTF score, the less liked by the evaluator the playlist has been. The novelty factor of a playlist represents the fraction of songs in the playlist that the user liked, but did not expect to see, i.e., they surprised him/her. Therefore, the novelty factor represents a good indicator of the performance of a recommendation algorithm in proposing new unknown content to the user that matches his/her taste. Sample playlists generated with Spotify Radio and our algorithm for different $|X|$ can be viewed at [19].

A. WTF score

Figure 6 shows the WTF scores for the two methods under comparison, as a function of the input vector X's size.

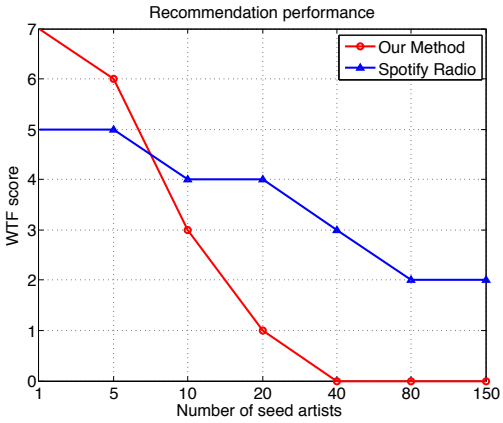


Fig. 6. WTF scores for our algorithm (in red) and Spotify Radio (in blue), as a function of the number of seed artists ($|X|$).

As expected, the recommendation accuracy of both methods improves with the size of X. However, the WTF score of our method decreases faster with $|X|$ than that of Spotify Radio, even though Spotify Radio appears more effective for very small seed sizes. Furthermore, the WTF score of the proposed method quickly reaches zero and stays consistently at that value for subsequently larger $|X|$. On the other hand, Spotify Radio exhibits a non-zero WTF score even for very large seed artist sizes. We believe that the observed performance difference between the two methods stems from the fact that our approach integrally considers X when proposing new content. On the other hand, Spotify Radio may be recommending content based on the entries of X independently, which would make it more susceptible to the impact of artist outliers. On average, we have been able to reduce the WTF score for 49% relative to Spotify Radio, for a 30 song playlist.

B. Novelty factor

Figure 7 shows the fraction of liked songs (LS) and new liked songs (NLS), in percent of N_s. We can clearly see that our method leads to better LS and NLS scores. In particular, we slightly improve LS by 4.5% (its value is 88% for Spotify Radio and 91.9% for our method). However, a remarkable improvement of the novelty factor of 42% has been achieved (NLS is 61% for our method and 43% for Spotify Radio). We believe that the latter gains are due to the design of our method that searches for related artists according to influences and then randomly selects popular songs from the top scored artists.

V. Conclusion

We have designed a novel algorithm for automatic playlist generation in Spotify that utilizes Facebook information on liked bands by a user and his/her listening history. Our approach is low-complexity and very effective in recommending good new content to the user, in particular when the number of input seed artists is at least 10. Relative to the built-in recommender Spotify Radio, we substantially improve the average WTF score by 49% and the novelty factor by 42%. As future work, we consider integrating genre discrimination into our method in order to recommend content based on a single genre, exclusively. Our methodology is general, which allows for its application to diverse personal content management platforms that exist in practice today.

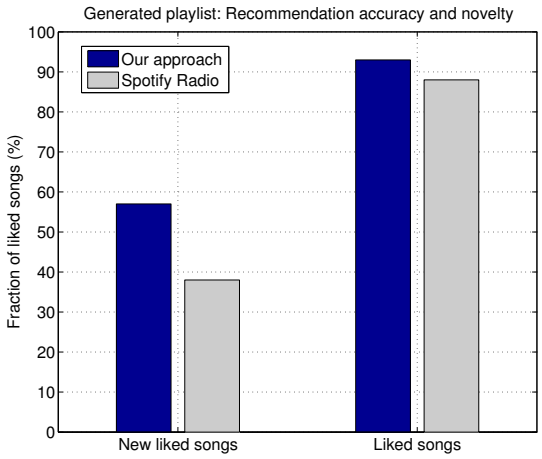


Fig. 7. Recommendation performance: Fractions of liked songs and new liked songs in percent of N_s.

References

[1] Spotify Ltd. (2007-2012) Home. [Online]. Available: http://www.spotify.com

[2] Blogmusik SAS. (2006-2012) What is Deezer? [Online]. Available: http://www.deezer.com

[3] Apple Inc. (2012) iTunes A to Z. [Online]. Available: http://www.apple.com/itunes/features

[4] Pandora Media Inc. (2012) Home. Only available in the US - 11/25/2012. [Online]. Available: www.pandora.com

[5] Spotify Ltd. (2012) Artist radio. [Online]. Available: http://www.spotify.com/se/about/features/artist-radio/

[6] F. Pachet and P. Roy, "Automatic generation of music programs," in *Proceedings of Constraint Programming Conference, CP 99*, ser. LNCS 1713/2004. Washington, VA: Springer Verlag, 1999, pp. 331–345.

[7] A.M. de Mooij and W.F.J. Verhaegh, "Learning Preferences for Music Playlists," Master's thesis, Technische Universiteit Eindhoven, 1997.

[8] T. W. Leong, F. Vetere, and S. Howard, "Randomness as a resource for design," in *Proceedings of the 6th conference on Designing Interactive systems*, ser. DIS '06. New York, NY, USA: ACM, 2006, pp. 132–139. [Online]. Available: http://doi.acm.org/10.1145/1142405.1142428

[9] Last.fm. (-) About last.fm. [Online]. Available: http://www.last.fm/about

[10] L. Barrington, R. Oda, and G. R. G. Lanckriet, "Smarter than genius? human evaluation of music recommender systems," in *Proc. ISMIR*, Kobe, Japan, Oct. 2009, pp. 357–362.

[11] Pandora Media Inc. (2005-2012) About the music genome project. [Online]. Available: http://www.pandora.com/about/mgp

[12] K. Yoshii, M. Goto, K. Komatani, T. Ogata, and H. G. Okuno, "Hybrid collaborative and content-based music recommendation using probabilistic model with latent user preferences," in *Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR*, 2006, pp. 296–301.

[13] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Commun. ACM*, vol. 35, no. 12, pp. 61–70, 1992.

[14] The Echo Nest. (-) Dev center. [Online]. Available: http://developer.echonest.com/

[15] M. Levy and M. Sandler, "Music information retrieval using social tags and audio," *IEEE Trans. Multimedia*, vol. 11, no. 3, pp. 383–395, 2009.

[16] S. Tan, J. Bu, C. Chen, B. Xu, C. Wang, and X. He, "Using rich social media information for music recommendation via hypergraph model," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 7S, no. 1, pp. 22:1–22:22, Oct. 2011.

[17] The Echo Nest. (-) Finding similar artists. [Online]. Available: http://developer.echonest.com/raw_tutorials/artist_api/raw_artist_02.html/

[18] P. Lamere. (2011, May) How good is googles instant mix? [Online]. Available: http://musicmachinery.com/2011/05/14/how-good-is-googles-instant-mix/

[19] A. Germain. (2012) Additional material - spotify-me. [Online]. Available: http://www.germainarthur.com/MMSPpaperAG