Enhancing next-track recommendation based on skipping behavior

Jacob O'Bryant

10 August 2017

Abstract

Previously we built a next-track music recommender system that used skipping data and session-based collaborative filtering to generate recommendations. To improve the system, we added two modifications. First, we balanced exploration with exploitation by adding a content-based component to the system and by giving more weight in the collaborative filtering component to songs that haven't been played very often. Second, we avoided over-playing songs by calculating a freshness score for each possible recommendation. Usage data we collected shows that the system performs better than random shuffling; however, we currently don't have enough data for more detailed analysis.

1 Introduction

In [1], we described a next-track music recommender system that uses skipping data and session-based collaborative filtering to generate recommendations. The system takes as input a set of songs (the user's music library) and the user's listening history, and it outputs which song should be played next.

That system had an important flaw: it tended to play the same songs over and over again. There were two different causes for this problem. First, the system did not actively explore the user's library. After it found a small set of songs that the user liked, it would repeatedly recommend those songs instead of continuing to find more favorable songs. Instead of finding a balance between exploration and exploitation, the system only exploited the data it already had.

The second cause was that the system did not take into account how recently each song had been played. The system should be less likely to play songs that aren't "fresh," i.e. songs that have been played recently.

This paper expands on our previous work by adding components for song freshness and exploration. In addition, we switched from a neighbor-based collaborative filtering approach to a model-based approach.

2 Related Work

To model song freshness, [2] and [3] use the forgetting curve, $1 - e^{-t/S}$, where t is the amount of time elapsed since a particular song was last played and S is the strength of the user's memory. We expand on this model in our work.

[3] also uses the Bayes-UCB (upper confidence bound) algorithm to balance exploration and exploitation in music recommendation. This work inspired the much simpler model we used.

3 Methods

Our original system used a neighbor-based collaborative filtering system. In order to reduce run time, we switched to a model-based system. For each candidate song c to be considered as the next recommendation, we used the model to calculate cf-score(c), a value from 0 to 1 representing how well c fits with the user's current session. To help with exploration, we calculated margin(c), the margin-of-error of cf-score(c). We also calculated freshness(c), a value from 0 to 1 representing how fresh c is. We combined these values to generate a final score as follows: score(c) = (cf-score(c) + margin(c)) $\cdot$ freshness(c).

If the model did not contain data for c, we instead used the formula score(c) = content-score(c) $\cdot$ freshness(c), where content-score is a fallback value calculated using content-based modeling instead of collaborative filtering. Each candidate thus had a score that was calculated using either cf-score or content-score.

To balance exploration and exploitation when picking the next song, 50% of the time we chose the best song (the song that maximizes score(c)) that was calculated using cf-score (exploitation). The other 50% of the time we chose the best song calculated using content-score (exploration).

In the following sections, we describe how each of these values was produced.

3.1 Model-based CF

For our model, we calculated cf-sim(a, b), the similarity[1] between two songs a and b, for each pair of songs in the user's library. Let U be the set of sessions containing both a and b. Then cf-sim(a, b) = $x/(x + y)$, where x is the number of sessions in U where both a and b were listened to and y is the number of sessions in U where either a or b was listened to but the other song was skipped. (We assume that if both a and b were skipped in the same session, it doesn't imply anything about their similarity). Note that cf-sim(a, b) is undefined if $x = y = 0$.

The desirability score of c given a single song s in the current listening session and its skip value skipped?(s) is calculated using formula 1.

If skipped?(s) is true and cf-sim(s, c) $<$ 0, then desirability(s, c) is undefined (if we skipped a song that is dissimilar to c, we don't assume that means c is a desirable candidate). desirability(s, c) is also undefined if cf-sim(s, c) is undefined. cf-score(c) is given by the average of desirability(s, c) for each s in the current session such that desirability(s, c) is defined. If desirability(s, c) is undefined for all s in the current session, then cf-score(c) is undefined.

3.2 Exploration

3.2.1 Content-based modeling

content-score was also calculated by taking the average of desirability(s, c) over the current session, but cf-sim(s, c) was replaced by a different function, content-sim(s, c). Using each song's title and artist, we queried Spotify's database for a vector of eight audio features (each with values ranging from 0 to 1): danceability, energy, mode, speechiness, acousticness, instrumentalness, liveness and valence. content-sim(s, c) was then given by the cosine distance of the feature vectors of s and c.

In cases where Spotify didn't have feature data for both s and c, we used a simple fallback similarity measure based on artist. Using the same method as described in "Model-

[1] By "similarity," we mean the likelihood of co-occurrence, which does not necessarily mean similarity of song content. Two acoustically dissimilar songs would be given a high similarity if the user listened to both songs together frequently.

$$\text{desirability}(s,c) = \begin{cases} \text{cf-sim}(s,c) & \text{if skipped?}(s) = \text{ false} \\ -\text{cf-sim}(s,c) & \text{if skipped?}(s) = \text{ true and cf-sim}(s,c) \geq 0 \end{cases} \quad (1)$$

based CF," we calculated cf-sim(A, B) for each pair of artists A and B. In this case, x was the number of times that some song a from artist A and some song b from artist B were listened to in the same session, and y was the number of times where one song was listened to and the other was skipped. We then defined content-sim$(s, c) =$ min(cf-sim$(S, C), 0.8)$ where S, C are the artists of s and c, respectively. We capped content-sim at 0.8 when using the artist-similarity fallback data in order to give preference to songs that were given a high content-sim score by the Spotify feature data.

If content-score(c) was still undefined (because none of the songs from the candidate song's artist have ever been played in the same session as any songs from the artists in the current session), we used a default score of 0.8. In effect, this artist similarity measure makes it so the system won't keep playing new songs from the same artist if the user always skips songs from that artist.

3.2.2 Upper confidence bound

Even within the set of candidate songs for which cf-score is defined, there is still a trade-off between exploration and exploitation. Our level of confidence in cf-score will vary based on how many data points were used to calculate it. Thus, we want to continue exploring songs that have low confidence levels because their cf-score might increase as more data is added.

[3] describes using the Bayes-UCB algorithm to balance exploration and exploitation in music recommendation. The essence of this algorithm is that instead of considering the estimated payoff that a certain action will have, we should consider the estimated upper confidence bound of the payoff. Actions with more uncertain payoffs will then be given higher priority until they have been explored more.

We used the arbitrary margin-of-error function margin$(c) = 1.1^{-n}$, where n is the number of songs in the current session for which cf-sim(s, c) is defined. Using cf-score(c)+margin(c) in our formula for score(c) gives us a rough upper confidence bound on the correct value for cf-score.

3.3 Freshness

[2] and [3] use the forgetting function $1-e^{-t/S}$ to model the freshness of a song, where t is the amount of time elapsed since the song was played last and S is a parameter that represents the strength of the user's memory. We used a modified version of this formula:

$$\text{freshness}(c) = (1 - e^{-t_1/S_c}) \times (1 - e^{-t_2/S_c}) \times \ldots \times (1 - e^{-t_n/S_c})$$

where t_i is the amount of time elapsed since the ith time in the past the candidate song c was played and S_c is the strength of the user's memory with regard to c.

This formula thus takes into account the entire play history of the song instead of just the most recent play. In addition, we dynamically learned the value of S_c for each candidate. A user will be willing to listen to some songs more frequently than others, so a single S value would be insufficient.[2] Given a value for S_c, we defined an error function as follows.

[2]Incidentally, S_c is an absolute measure of the user's favor for c.

Let observed$(i) = 0$ if c was skipped on the ith play, and let observed$(i) = 1$ otherwise. Let freshness(c, i) be the freshness score calculated such that only the first $i - 1$ plays of c are used and t_k is the time elapsed from the kth to the ith time c was played. (freshness(c, i) is the freshness score right before c was played for the ith time.) Then the error is given by

$$\sum_{i=1}^{n} (\text{observed}(i) - \text{freshness}(c, i))^2$$

For ease of implementation, we used a list of fixed values and then for each candidate chose the S_c that minimized the error function. The values we used were 0.2, 0.5, 1, 1.5, 2, 3, 5, 8, 13 and 21 (measured in days).

This error function assumes that if c was skipped, the freshness score should have been 0, and if c was not skipped, the freshness score should have been 1. This is turn implies that c is appropriate for the current session and the only reason the user would skip it is if it isn't fresh. This assumption is clearly not always the case, but we used it to simplify the implementation.

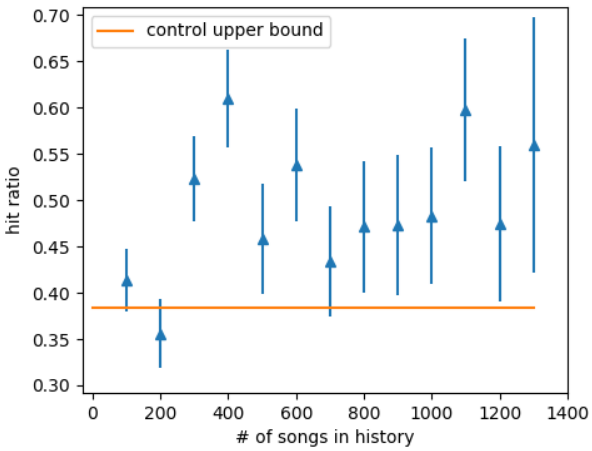

Figure 1: Hit ratios and margins-of-error for Smart Shuffle vs. hit ratio + margin-of-error for control data set

4 Results

To generate a control data set, the system chose a random song with probability 0.2 every time a new song was played. We distributed the app, Smart Shuffle Player, on Google Play and accumulated 13 users. There was a total of 4,395 listening events (each event being either a skip or a listen). The hit ratios (the percentages of played songs that weren't skipped) are given in table 1.

Figure 1 attempts to give a more detailed breakdown of the system's performance. Although table 1 shows that the system overall performs better than random shuffling, we see from the lack of a clear pattern in figure 1 that there isn't enough data yet to show how quickly the system improves.

Note: when this data was collected, we had not implemented content-based modeling yet. The freshness score at that time was also calculated with only the single most recent play of the song, and we used a hard-coded value (two days) for the S value.

5 Future Work

5.1 Data Analysis

An important step in collecting more data will be to improve the Smart Shuffle Player app so it can attract more users. It mainly plays only songs from the user's local music collection (although it has rudimentary support for playing songs from Spotify). Many users store their music in cloud services such as Google Play Music. Adding support for these cloud services would increase the number of people that can use our app. Creating a version of the app for iOS would also increase the potential user base.

	Hit ratio	Margin of error
Control	0.34	0.04
Smart Shuffle	0.47	0.02

Table 1: Hit ratios (margin of error calculated with 95% confidence)

After collecting more data, we plan to graph the hit ratios with respect to cf-score and content-score. This would tell us how useful the scores are and would assist in identifying which parts of the system need to be improved the most.

More data analysis would also help us tune parts of the code that are currently set arbitrarily, such as the margin-of-error function used in calculating the upper confidence bound of cf-score.

5.2 Exploration

As mentioned, the system should decide automatically how often to explore or exploit. The hit ratio data discussed in the previous section could be helpful for this. If there is a target hit ratio that the system tries to maintain and it knows the respective probabilities that the user will skip if we either explore or exploit, the system could calculate how often it can afford to suggest new songs without descending below the target hit ratio.

When the user starts using the system, the system should also make sure that it doesn't exploit too often while there is only a small number of songs that it knows the user likes. Otherwise the system will tend to play songs that aren't fresh and are thus likely to be skipped.

5.3 Freshness

It would be useful to verify if the forgetting curve actually describes how users perceive music. Perhaps instead there is a threshold for the number of times the user is willing to listen to a song in a certain period of time, and if that threshold is crossed, the freshness of the song declines steeply and requires a long amount of time to recover. If this is the case, the system could play songs more repeatedly as long as it takes care not to cross that threshold. Crossing the threshold would be like the straw that broke the camel's back.

More work can be done to learn the S_c values also. For instance, we could improve the error function so that it takes into account how confident the system was that c is appropriate for the current session.

6 Conclusion

We have presented a next-track music recommender that learns from the user's skipping data and have shown that it performs better than random shuffling. We presented a method for balancing exploration and exploitation by combining collaborative filtering and content-based modeling into a hybrid system and by using upper confidence bounds. We also presented an improved model for song freshness. Finally, we have laid out a clear path for future improvements to be made to this system.

References

[1] J. O'Bryant, "Moodful next-track music recommendation based on skipping behaviour," Apr. 25, 2017. [Online]. Available: `http://jacobobryant.com/about/papers/musicrec_2017_apr.pdf`.

[2] Y. Hu and M. Ogihara, "NextOne player: A music recommendation system based on user behavior.," in *ISMIR*, 2011, pp. 103–108. [Online]. Available: http://www.academia.edu/download/38297540/PS1-11.pdf (visited on 03/05/2017).

[3] Z. Xing, X. Wang, and Y. Wang, "Enhancing collaborative filtering music recommendation by balancing exploration and exploitation.," in *ISMIR*, 2014, pp. 445–450. [Online]. Available: http://www.terasoft.com.tw/conf/ismir2014/proceedings/T081_140_Paper.pdf (visited on 04/05/2017).