Random Walk with Restart for Automatic Playlist Continuation and Query-Specific Adaptations

Master's Thesis

T. van Niedek

Radboud University, Nijmegen
timo.niedek@student.ru.nl

2018-07-16

First Supervisor
Prof. dr. ir. A.P. de Vries
Radboud University, Nijmegen
a.devries@cs.ru.nl

Second Supervisor

Radboud University

Abstract

In this thesis, we tackle the problem of automatic playlist continuation (APC). APC is the task of recommending one or more tracks to add to a partially created music playlist. Collaborative filtering methods for APC do not take the purpose of a playlist into account, while recent research has shown that a user's judgement of playlist quality is highly subjective. We therefore propose a system in which the characteristics of the input playlist are used as a proxy for its purpose, and build a recommender that combines these features with a CF-like approach.

Based on recent successes in other recommendation domains, we use random walk with restart as our first recommendation approach. The random walks run over a bipartite graph representation of playlists and tracks, and are highly scalable with respect to the collection size. The playlist characteristics are determined from the track metadata, audio features and playlist title. We propose a number of extensions that integrate these characteristics, namely 1) title-based prefiltering, 2) title-based retrieval, 3) feature-based prefiltering and 4) degree pruning.

We evaluate our results on the Million Playlist Dataset, a recently published dataset for the purpose of researching automatic playlist continuation. The proposed extensions lead to improved performance over the baseline random walk with restart method. Additionally, we observe that our methods are effective in reducing popularity bias, a common issue for collaborative filtering methods. The best performing recommender uses a hybrid degree pruning techinque, for which we report an R-Precision of [TBD], an NDCG of [TBD] and a Recommended Songs Clicks of [TBD]. This work was conducted in the context of the RecSys Challenge 2018.

Contents

Chapter 1

Introduction

As of today, many users enjoy the ease of access to music through online streaming services. The digitization of the music industry has brought many oppurtunities for instantaneously delivering a large variety of music to users at any time. It is common for these services to facilitate the creation of music playlists. Playlists allow users to create a manually curated list of music to listen to continuously in a single session. The importance of playlists is highlighted by a study that showed that 55% of music streaming service users create playlists [1].

With the emergence of online music streaming services, there have been many research efforts towards creating and improving music recommender systems. One subtopic is the task of automatic playlist continuation (APC). This task is to recommend songs to add to a playlist, given a partially created playlist with one or more tracks. A current challenge for APC is the problem of taking into account the intent of the playlist creator [2]. Current methods are unaware of the reasoning behind the inclusion or exclusion of tracks, while qualitative studies have shown that a user judgement of what is a good playlist is highly subjective [3, 4]. Based on the work of Lee et al. [4], we argue that every playlist is created with a specific purpose. For example, some playlists are created for a certain listening context or with a general theme in mind. By determining the characteristics of a playlist, we can focus the recommendations towards tracks that fit the same characteristics. In this work, we propose a recommender that adapts to the characteristics of the query playlist by generating recommendations from other playlists that share these characteristics.

Recommendations are generated from a bipartite graph representation of tracks and playlists. We call this graph the *playlist graph*. We adapt the Pixie recommender system by Eksombatchai et al. [5], which was created for the image sharing platform Pinterest [6]. The Pixie Random Walk algorithm can generate recommendations

from a large object graph efficiently and has the advantage that it is adaptive to user-specific demands. Instead of biasing the random walk based on these demands, we use features describing the query playlist in order to prefer recommendations from other playlists with the same feature set. A set of candidate playlists with similar characteristics to the query playlist is selected, after which we only generate recommendations from this subset. We investigate whether the playlist title or track metadata are useful filtering criteria in terms of the quality of the recommendations. Prefiltering based on playlist titles has been shown to lead to an increased performance for music recommendation [7]. As an additional step in helping the user create a playlist, we propose a recommender for the case when the user has decided on a title but has not added any tracks yet.

We build a playlist model from the track metadata and filter the candidate set of playlists and tracks in two ways: by discarding other playlists in the graph if the distance between playlist models is large, and by removing popular tracks from playlists in cases where the track does not match the other tracks in the playlist based on the metadata. Our metadata-based prefiltering methods are general and can be applied to any set features.

Our contributions to the task of automatic playlist contribution are 1) a novel strategy of generating recommendations for APC using random walks over bipartite graphs, 2) a number of extensions, including title-based prefiltering, title-based retrieval, feature-based prefiltering and degree pruning, and 3) an evaluation of the system on a large scale.

This project was conducted in the context of the ACM RecSys Challenge 2018 [8]. This challenge is supported by Spotify [9], a highly popular music streaming service. For the purpose of the RecSys Challenge, Spotify has released the *Million Playlist Dataset* (MPD) [10]. This dataset contains a million playlists created by users on Spotify, including track, artist and album names as well as the title of the playlist and additional metadata.

Chapter 2

Related Work

In this chapter, we further introduce the setting of our work by listing related research. We divide the related work into three parts: automatic playlist generation, related random-walk-based methods, and work on prefiltering for automatic playlist continuation.

2.1 Automatic Playlist Generation

Automatic Playlist Generation (APG) is a broad term for any task that involves generating a sequence of tracks given a collection of songs, background knowledge and target characteristics, as defined by Bonnin and Jannach [11]. Their definition stems from the process of how humans create playlists, that is, using background knowledge such as tempo, genre or mood to determine some desired features of the playlist and selecting songs from the collection accordingly. Multiple different variants of APG have been formulated in previous research, using different definitions of background knowledge and target characteristics. Bonnin and Jannach give an overview of the different variants. In this work, we focus on the subtask of Automatic Playlist Continuation (APC), which is described by Schedl et al. [2] as 'adding one or more tracks to a playlist that fit the same target characteristics of the original playlist'.

Even though a playlist is an ordered sequence of songs, research has shown that users in fact do not care about the ordering of the songs, or do not notice that there is an intended order [12]. Additionally, models incorporating sequential information do not gain an advantage when many popular songs are present [13]. Since the usefulness of a specific order is unclear, we do not take it into account when generating recommendations.

2.2 Random Walks

The most famous random-walk-based method is PageRank [14], which models the browsing behavior of a surfer on the internet, where the internet is represented as a graph. An extenstion of this model is personalized PageRank [15, 16, 17], which includes a probability that the surfer teleports to a personalized location in the graph. Random walk with restart is similar to personalized PageRank, but instead of teleporting to a set of preferred nodes, the walk has a probability of restarting at the node where it began [18, 19]. Random walk with restart has been proven useful for recommender systems in multiple domains. Examples include multimedia retrieval [18], link prediction in large social networks [20, 21] and recommendations for curated collections of images [5]. Another related technique to this work is SALSA and personalized SALSA [22], which runs random walks over a bipartite graph to find *hubs* and *authorities*.

The most closely related applications of random walk with restart is Pixie [5], a recommender for the visual catalog Pinterest [6]. Pixie is a recommender that computes the relative importance between two 'pins' (images) using random walk with restart on a bipartite graph of pins and 'boards'. Boards are manually curated collections of pins that can be created by any user. Our graph representation and baseline methods are inspired by Pixie, since Eksombatchai et al. have shown that their method leads to high quality recommendations and is highly scalable. We adapt Pixie to the domain of automatic playlist continuation and propose a number of extensions.

Instead of computing the stationary distribution of the transition probabilities [18], which suffers from scalability issues [23], our work uses the Monte Carlo method which simulates the random walks [23, 16].

2.3 Prefiltering Methods

In our work, we apply prefiltering methods to the playlist graph in order to focus the random walk with restart to playlists that have similar characteristics. A related technique is contextual prefiltering, selecting only the set of data that is relevant to the current context, and then generating ratings using any recommender system [24]. Pichl et al. [7] have shown that contextual prefiltering using playlist titles leads to a large improvement for music recommendation. We differ from their work since we do not consider the context as an explicit dimension in our model. Instead, we prefilter based on the playlist title, metadata and audio features as a general graph operation. This means that we also implicitly model context as part of the features, without

making any assumptions on what a context is.

Related to prefiltering based on audio features and metadata, Aucouturier and Pachet [25] use constraint-based prefiltering to remove irrelevant songs from the database. We use the query playlist as an indicator of the desired characteristics instead of explicit constraints, since our recommenders do not model the user. Our recommenders view the playlist as the user, and consider the set of tracks in the playlist as positive ratings, a concept described in [11].

Chapter 3

Data

In this chapter, we describe the datasets used in the present work. As part of the RecSys Challenge 2018, Spotify has provided participants with the Million Playlist Dataset (MPD), which is described in Section 3.1. To establish the playlist model which encodes playlist characteristics, we require track metadata. Since the MPD does not contain such metadata, they are gathered from the Spotify API [26]. This process is described in Section 3.3.

3.1 The Million Playlist Dataset

The Million Playlist Dataset [10] was created by Spotify for the RecSys Challenge 2018 [8] and contains playlists created by Spotify users. The MPD is a random sample of all playlists created by users in the United States from 2010 to 2017. Playlists can only be included if they meet the following quality criteria [27]:

- Created by a user residing in the United States, at least 13 years old
- Playlist is public
- Between 5 - 250 tracks long
- At least 3 unique artists
- At least 2 unique albums
- At least 1 follower, not including the creator
- Created between January 1, 2010 and December 1, 2017
- No offensive title
- Does not contain non-Spotify tracks that are stored locally on the device of the user

The playlists are anonymous in the sense that no user-identifyable information is

included in the dataset, and playlists are only included when its title is shared with a number of other playlists in the MPD.

A playlist is represented in the MPD as an ordered list of tracks along with some playlist metadata. The metadata include the name of the playlist, whether the playlist is collaborative or not, the latest date of modification, the number of edits, the number of followers (not including the creator) and the description created by the user. A mere 1.88% of all playlists contain descriptions since they are optional. Tracks are represented primarily by the unique track URI, which can be used to retrieve the track from Spotify. Addionally, the track name and artist name are included, as well as URIs that point toward the artist and the album. No further track information was provided.

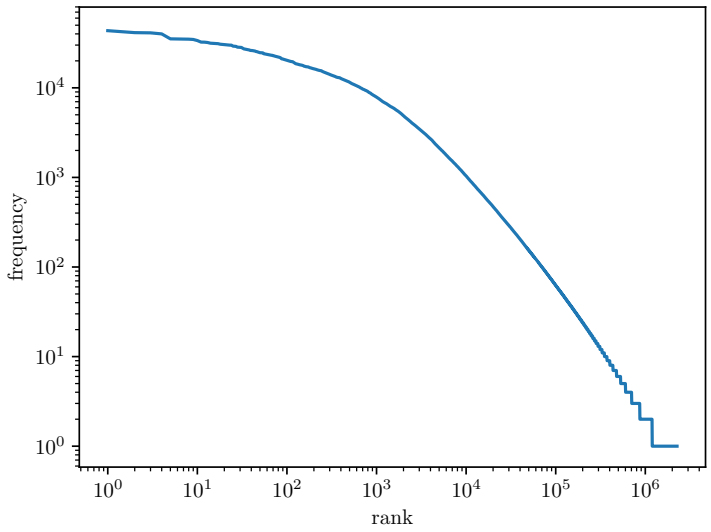


Figure 3.1: Log-log plot of track frequency in the Million Playlist Dataset.

The playlists included in the MPD contain in total roughly 2.3 million unique tracks. Fig. 3.1 shows the frequency of tracks in the MPD. There is a long tail of tracks that have been included in only one playlist. In fact, 47.45% of all unique tracks only appear in a single playlist.

3.2 Challenge Set

The Million Playlist Dataset (MPD) is used as our development set, and a separate challenge set is used for the final evaluation. The challenge set is also provided as part of the RecSys Challenge 2018 [8], and contains 10,000 incomplete playlists selected using the same criteria used for the MPD [28]. An additional criterion is that all tracks in the challenge set must appear in the MPD. The representation of playlists and tracks are the same as the representation in the MPD, as described in the previous section. The task is to recommend continuations for all incomplete playlists in the challenge set. The playlists are split up into ten different categories:

1. Title only
2. Title and first track
3. Title and first 5 tracks
4. First 5 tracks (no title)
5. Title and first 10 tracks
6. First 10 tracks (no title)
7. Title and first 25 tracks
8. Title and 25 random tracks
9. Title and first 100 tracks
10. Title and 100 random tracks

Each category contains 1000 playlists. The remaining tracks for each playlist are kept completely separate in order to ensure a fair evaluation.

3.3 Track Metadata

We retrieved additional metadata through the Spotify API [26]. Speficially, we use the release year of tracks and whether or not the track is marked as explicit [29]. Furthermore, the Spotify API includes an Audio Features endpoint, which returns a set of measures determined directly from the audio. While the methods for extracting these features are closed-source, a description is provided in the API documentation [30]. We retrieve the scores for *danceability*, *energy*, *speechiness*, *acousticness*, *instrumentalness*, *liveness*, *valence* and *tempo*.

We represent each song using the track metadata and audio features. These features are used to represent playlist characteristics as described in Section 4.4.1. We provide a brief description of each feature based on the Spotify API reference [26, 30, 29] in Table 3.1. We make sure that all features lie within the domain from 0.0 to 1.0. For the "Count" feature, we normalize by taking the log of the frequency

and dividing by the log of the maximum frequency. We take the log to ensure that the domain of this feature is not dominated by the high frequency tracks shown in Fig. 3.1. We clip the tempo feature to a minimum of 50 and a maximum of 180 beats per minute (BPM) since most tracks fall within this range, and the precise BPM is not as informative for extremely slow or extremely fast tracks. For any track labeled as released before 1920 or after 2017, we leave the "Year" feature empty, as manual inspection revealed that these tracks were almost always mislabeled in the metadata. Fig. 3.2 shows the feature distributions over all tracks in the MPD.

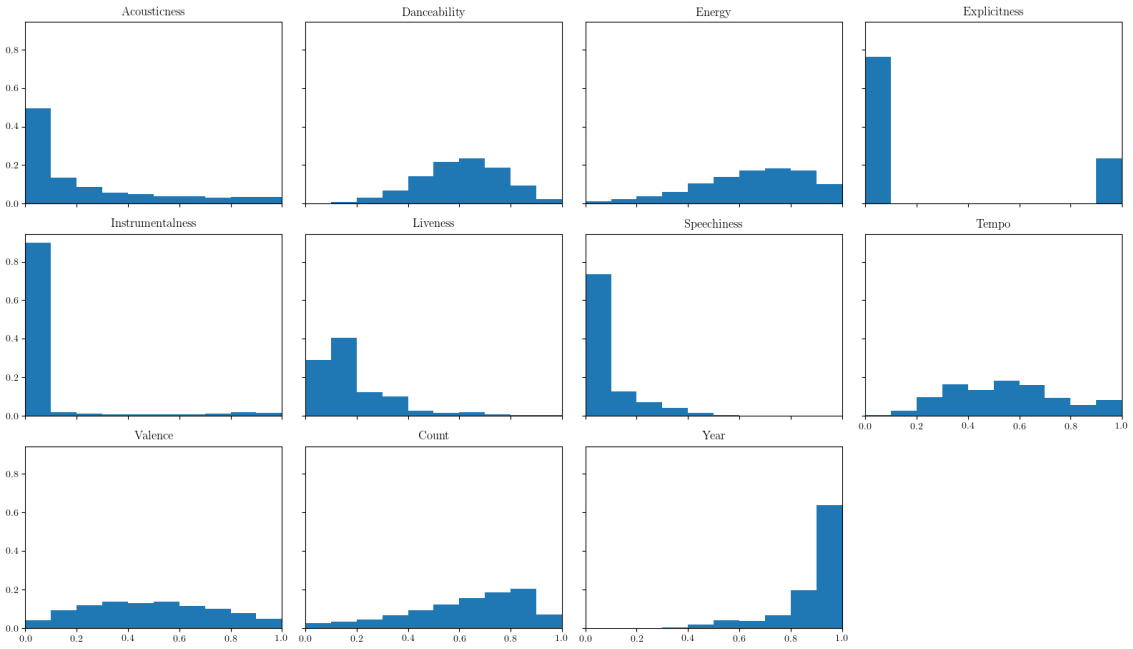


Figure 3.2: Feature distributions over the entire Million Playlist Dataset. The features are normalized to a range of 0.0 to 1.0 as described in Table 3.1 and grouped into ten bins. The y-axis shows the relative frequency of the values within each bin.

Feature	Description	Normalization
Acousticness	A score between 0.0 and 1.0 where 1.0 represents a high confidence that the track is acoustic [30].	
Count	Number of occurrences of the track in the Million Playlist Dataset.	$c(t_i) = \frac{\log(\deg(t_i))}{\log(\max_{t \in T}(\deg(t)))}$
Danceability	A score between 0.0 and 1.0 where 1.0 represents a high confidence that the track is danceable, based on a combination of musical elements [30].	
Energy	A measure between 0.0 and 1.0 representing perceptual intensity and activity [30].	
Explicitness	Boolean value indicating whether or not the track has explicit lyrics [29].	1.0 for explicit tracks, 0.0 otherwise.
Instrumentalness	A score between 0.0 and 1.0 where 1.0 represents a high confidence that the track contains no vocals and 0.0 a high confidence that there are vocals [30].	
Liveness	A score between 0.0 and 1.0 where 1.0 represents a high confidence that there is an audience present in the recording [30].	
Speechiness	A score between 0.0 and 1.0 where 1.0 represents a high confidence that the track is speech-like [30].	
Tempo	An estimate of the beats per minute (BPM) of a track [30].	Rescaled the range of $[50, 180]$ to $[0.0, 1.0]$, clipping values to 0.0 and 1.0 when outside the range.
Valence	A measure between 0.0 and 1.0 where low values represent a negative sound (e.g. sad) and high values represent a positive sound (e.g. happy) [30].	
Year	The year in which the track was released [29].	Rescaled the range of $[1920, 2020]$ to $[0.0, 1.0]$, tracks released before 1920 are treated as a missing value.

Table 3.1: List of features used in the playlist model and how they were normalized. $\deg(t)$ is the degree function as defined in Eq. (4.2), which is for a track $t \in T$ equivalent to the number of times it has been included in a playlist.

Chapter 4

Methods

In this chapter, we detail our methods for generating recommendations from the Million Playlist Dataset (MPD). In the first section, we start with a description of the graph structure that was used to model the playlist data, which we refer to as the *playlist graph*. The playlist graph is the main data structure used throughout this work, as track recommendations are the result of multiple random walks over this graph. The random walk algorithms are described in Section 4.2. We discuss how we processed the titles of playlists in Section 4.3. We use the playlist titles for two different extensions: a prefiltering approach (Section 4.3.2) and track retrieval based on the playlist title only (Section 4.3.3). Section 4.4 explains our methods for using the track metadata for modeling playlists. We use the playlist model for prefiltering the playlist graph as described in Section 4.4.2. In the final section, we show how we pruned the graph using the playlist model.

4.1 The Playlist Graph

The playlist graph is based on the Pinterest graph described by Eksombatchai et al. [5]. Both graphs are undirected and bipartite. The Pinterest graph connects pins on one side of the graph with boards on the other side. Similarly, Spotify's playlists can be represented using such a graph, connecting tracks with playlists. More formally, we define the playlist graph as an undirected biparite graph $G = (T, P, E)$ where T denotes the set of tracks, P denotes the set of playlists and E is a set of undirected edges (t, p) with $t \in T$ and $p \in P$. An edge exists between a track t and a playlist p only if the track is included in the playlist. In this thesis, we disregard track ordering information. We allow the existence of multiple edges between a playlist node p and a track t in order to model cases where a user has

included a track multiple times in their playlist.

For random-walk based algorithms, a large part of the runtime is spent sampling a neighbor to step to. It is therefore very important to use a data structure that allows efficient neighbor sampling. To this end, we use an implementation similar to that of Eksombatchai et al. [5], which is memory-efficient and implements neighbor sampling in constant time. We assign node IDs in the range of 0 to $|P|$ to the playlists, and IDs in the range of $|P|$ to $|P|+|T|$ to the unique tracks ($|P| = 1.000.000$, $|T| = 2.262.292$). We first assign empty adjacency lists to every one of the $|P|+|T|$ nodes and fill them by iterating over the MPD. We then concatenate all adjacency lists into a single adjacency vector $\mathbf{a}$. The indices at which to find the start of the adjacency list for a given node are stored in the offsets vector $\mathbf{o}$. The neighborhood operation is implemented in constant time with a single list slice following Eq. (4.1). Intuitively, the neighborhood of a playlist node is all tracks included in that playlist, and the neighborhood of a track node corresponds to all playlists in which that track was included. The degree of a node is computed using Eq. (4.2).

$$N_G(v) = \{a_i, a_{i+1}, \ldots, a_{j-1}\}, \text{ where } i = o_v, j = o_{v+1} \tag{4.1}$$

$$\deg(v) = o_{v+1} - o_v \tag{4.2}$$

The full graph containing 1.000.000 playlists and 2.262.292 unique tracks consumes roughly 1.8 GB of memory, and the entire playlist graph can therefore fit in the memory of a single machine.

4.2 Random Walks

The playlist graph described before gives a structure to the domain of playlists and tracks from which we wish to generate recommendations. Recommendations are often related to some sort of concept of similarity: a user wants to receive recommendations similar to their query. Intuitively, tracks that appear in the same playlists are likely similar. Likewise, playlists that have one or more tracks in common are likely similar. Random walks exploit these properties to find similar tracks to a query track by traversing the graph in random directions. In the following sections, we will describe the random walk methods used in this thesis.

4.2.1 Single Random Walk

The most basic variant of random walk algorithm is the single random walk, shown in Algorithm 4.1. This variant generates recommendations with a single query node as input. The transition probabilities are uniform. The random walk starts by initializing the visit counts for every track node to 0. The outer loop runs a number of walks until the maximum number of steps N is reached. At the start of each walk, the current node is reset to the query node q. Each step within the walk consists of three operations [5]:

1. Given the current track t_{curr} (initially set to the query track q), sample a playlist p_{curr} from its neighbors $N_G(t_{curr})$ using Eq. (4.1).
2. Given p_{curr}, sample a track t'_{curr} from its neighbors $N_G(p_{curr})$ using Eq. (4.1).
3. Update the current track t_{curr} to t'_{curr} and repeat.

In the inner loop, line 9 samples a random adjacent playlist and line 10 samples a random adjacent track node. The function RandomNeighbor simply returns a single random uniform sample of the neighborhood $N_G(v)$ of a given node v (Eq. (4.1)).

After every step, there is a non-zero probability of restarting at the query node. The restart probability is denoted with α. In the RandomGeometric function, we sample a walk length from the geometric distribution

$$P(k) = (1-\alpha)^{k-1}\alpha. \tag{4.3}$$

This distribution models the probability that the walk restarts at exactly the kth step and steps normally for the $k-1$ steps before, for a given restart probability α. As for the extreme cases, if $\alpha = 0$ the walk will never restart. If $\alpha = 1$ the walk will restart after a single step, which is equivalent to the Random-Random model described by Leskovec et al. [31].

The result of the single random walk is a dictionary of (node: visit count) pairs, where the nodes with highest visit count are most similar to the query node. This technique is very related to neighborhood formation in bipartite graphs [32], as the visit counts represent an approximation of the soft neighborhood of the query node.

4.2.2 Multiple Random Walks

To generate recommendations for a playlist, we run the single random walk for every query node and aggregate the results. This is shown in Algorithm 4.2, which is similar to the Pixie algorithm [5]. The differences are that the number of steps per

Algorithm 4.1 Single Random Walk with Restarts (based on [5])

function SINGLERANDOMWALK(q: Query Node, $G = (T, P, E)$: Playlist Graph, α: Restart Probability, N_q)
    $n \leftarrow 0$
    $V \leftarrow \{v : 0\}\ \forall_{v \in T}$
    repeat
        $t_{curr} \leftarrow q$
        $n_{curr} \leftarrow 0$
        $N_{curr} \leftarrow$ RANDOMGEOMETRIC(α) [Eq. (4.3)]
        repeat
            $p_{curr} \leftarrow$ RANDOMNEIGHBOR(G, t_{curr})
            $t_{curr} \leftarrow$ RANDOMNEIGHBOR(G, p_{curr})
            $V[t_{curr}] \leftarrow V[t_{curr}] + 1$
            $n_{curr} \leftarrow n_{curr} + 1$
        until $(n_{curr} \geq N_{curr}) \vee (n + n_{curr} \geq N_q)$
        $n \leftarrow n + n_{curr}$
    until $n \geq N_q$
    return V
end function

node N_q are assigned uniformly, and no multi-hit boosting is applied. Instead, to obtain a ranking over tracks based on the query playlist, we simply sum the visit counts for all query nodes.

Algorithm 4.2 Multiple Random Walks with Restarts (based on [5])

```
1: function MultipleRandomWalk(q: Query Tracks, G = (T, P, E)x: Playlist
   Graph, α: Restart Probability, N)
2:     N_q ← N/|q|
3:     V ← {v: 0} ∀_{v∈T}
4:     for all q ∈ q do
5:         V_q ←SingleRandomWalk(q, G, α, N_q)
6:         V ← V + V_q
7:     end for
8:     return V
9: end function
```

The single random walks are completely independent of each other. We can therefore parallelize the for-loop in Algorithm 4.2 to speed up computation time by utilizing multiple CPUs. Since both this algorithm and Algorithm 4.1 do not require write access to the playlist graph, it can reside in shared memory and no locking is required. Parallelization of the random walks leads to a speedup roughly by a factor of 2.3 when utilizing five processes.

4.3 Title Processing

The titles of playlists are considered to be an indicator of the intent of the playlist creator [2]. Indeed, playlists are often named after genres, artists, tracks, time period or the context in which a user might listen to it [7]. This is reflected in the MPD: some examples of the most occuring playlist titles are 'country' (10000 occurrences), 'chill' (10000), 'workout' (8481), 'christmas' (8015) and 'party' (6157). Pichl et al. [33] showed that there is a large increase in music recommendation quality when playlists are clustered into contextual clusters based on their titles and then using collaborative filtering per cluster. They apply this technique for recommending tracks to users directly. Instead, we use the playlist titles in an automatic playlist continuation setting. In the remainder of this section, we first describe how the titles were preprocessed. Then, we describe how the titles were used for a prefiltering approach, inspired by [33]. Finally, we consider the cold-start scenario in which only a title has been provided but no tracks yet, and develop a title-based retrieval system.

4.3.1 Preprocessing

As is common with short user-provided texts, there are many playlist titles that are noisy. In addition, playlist titles are often very short, usually only one or two words. We therefore design a preprocessing pipeline that allows us to match playlist titles as well as possible. We first convert the titles to lowercase and strip punctuation characters. Many playlist titles include emoji, and it is common to find playlist titles consisting only of emoji. Since these emoji may carry meaning in such a way that it reflects the contents of a playlist (e.g. 🤘 is related to rock music, ❤️ to love songs), we choose to include them as separate tokens. Single character repetitions are truncated to a maximum of two characters (e.g. 'happyyyy' is truncated to 'happyy') to reduce the number of variations of a word. We finally remove stopwords and stem using the Porter stemming algorithm [34]. The python package Whoosh[1] is used for stemming, indexing and retrieval.

4.3.2 Title-based Prefiltering

In order to generate high quality recommendations, we would like to recommend tracks that fit the target characteristics of a query playlist. We therefore prefer recommending tracks that appear in other playlists that share these characteristics. Since the playlist title is an indicator of its purpose [7], we propose to filter the playlist graph based on the title of a query playlist. Instead of running the random walk on the entire graph, we run it on the subgraph which only includes playlist nodes for which the playlist title is sufficiently similar to the title of the query playlist.

For a given query playlist, we apply the preprocessing steps as described in the previous section. We then find all other playlists whose titles contain at least all terms in the query playlist title. These nodes are then kept in the subgraph corresponding to the query playlist. We do not use a sophisticated ranking function for this part, since the playlist titles are short and recall is much more important than precision. It is more harmful to prune relevant playlists than it is to keep irrelevant playlists in the subgraph, since irrelevant playlists have a low probability of being visited by the random walk anyway. Finally, we only apply title-based prefiltering if the graph after filtering contains more than 150 playlist nodes. If there are very few matches for a playlist title, it is highly unlikely that it is informative. Title-based prefiltering is only useful for titles that imply some shared meaning, not when the title is only meaningful to the creator of the playlist.

[1] `https://pypi.org/project/Whoosh/`

4.3.3 Title-based Retrieval

We now consider the situation in which a user has determined a title for a playlist, but has not yet added any tracks. This setting is an additional step in helping the user build a playlist, and can be considered as a metadata-based query [2]. Given a sufficiently informative playlist title, a good starting point is again to retrieve tracks from playlists with the same title. We therefore use the same title preprocessing and indexing steps as before to retrieve a list of playlists ranked on the similarity to the query title. In contrast to the prefiltering approach, we only select the most relevant playlists, since partial matches are less likely to be relevant to the intent of the user. We use the BM25 function, which has been proven successful for many information retrieval tasks (see [35] for an overview), to rank the playlists in the MPD in terms of the title similarity. We retrieve the tracks for all playlists with a BM25 score of at least 5 and sort the tracks based on the number of occurrences in this set of playlists. The result is an ordered list that indicates which tracks are most commonly associated with the given title. We select the tracks that appear in at least 15% of the retrieved playlists and run the random walk with these tracks as input in order to expand the recommendations beyond the most common tracks for the given title. For efficiency, we limit the number of query tracks to 100.

4.4 Playlist Modeling

While the random walks compute similarity scores between tracks based on the local graph structure, they are agnostic of the content-based similarity between tracks or playlists. Methods that recommend solely based on community-provided ratings are not designed to take the target characteristics of playlists into account. In contrast, similarity-based recommendations select tracks that best fit the desired characteristics by recommending tracks with the lowest distance to the query. To improve the coherence of the recommendations with respect to the query playlist, we propose to model the contents of a playlist using content-based features and metadata for each track. We then define a distance metric between playlists, which is used for prefiltering approaches as described in Section 4.4.2. We also use the playlist models to clean the graph by pruning edges between high-degree tracks and playlists, which is described in Section 4.4.3.

4.4.1 Histogram Model

Throughout this work, we simplify the playlist representation by assuming that the order of the tracks does not matter. There has been evidence that listeners do not consider the ordering of playlist tracks important [12], and that models incorporating sequential information do not gain an advantage when many popular songs are present [13].

A track t is represented by the features described in Table 3.1:

$$t = \{f_1(t), f_2(t), \ldots, f_{|F|}(t)\} \tag{4.4}$$

where $f_i(t)$ is the feature value of feature i for track t and $|F|$ is the size of the feature set $F = \{\text{Acousticness, Count}, \ldots\}$. We can then create a playlist model as the set of distributions $P(f_i|p)$ for each feature dimension i. Since we represent the playlist as an unordered collection of tracks, a playlist is the result of repeatedly sampling i.i.d. feature values from $P(f_i|p)$ for all features. The distance between two playlists can then be expressed in terms of the distance between each feature distribution.

We represent a playlist using normalized histograms as an estimate of the continuous probability distribution function. We choose this model for its simplicity, and because it does not make any assumptions about the modality or skewness of the underlying distributions. The feature distribution for a feature i is estimated from its empirical distribution as

$$P(f_i \in b|p) = \frac{H(b)}{N_i} \tag{4.5}$$

where b is the bin in which the value f_i falls, $H(b)$ is the number of tracks in playlist p that fall in bin b and N_i is the total number of observations for feature i in playlist p, not counting missing values.

4.4.2 Feature-based Prefiltering

Similar to the reasoning behind title-based prefiltering described in Section 4.3.2, we argue that the playlist model encodes playlist purpose information. For example, based on these features we can distinguish playlists that only contain tracks from the 80's, playlists with high tempo, happy or sad playlists, energetic or calm playlists, and we can then adapt our recommendations to these characteristics. We therefore propose to use the playlist models to determine a subgraph of playlists that have a low distance to the query playlist, and remove the remaining playlists at runtime. This focuses the random walks to the playlists that are more likely to be relevant

based on the track features, and thus leads to recommended tracks that should fit the distribution of the query playlist.

We adopt the very general approach of pruning the graph based on the distance between two playlists. We use the ℓ_1 distance metric summed over features for comparing playlist histograms:

$$d(p_1, p_2) = \sum_{i=1}^{|F|} \sum_{b \in bins} \big| P(f_i \in b|p_1) - P(f_i \in b|p_2) \big|. \tag{4.6}$$

While there are distance measures that may have a better foundation theoretically, such as the Jensen-Shannon divergence or earth mover's distance, we find that these measures are too computationally intensive when comparing a query playlist with all 1M other playlists in the collection.

The histogram distribution of short playlists is sparse; there are many empty bins. For these playlists, we observed that there were often very few similar playlists. We solve this by introducing a smoothing parameter λ which controls the amount of Jelinek-Mercer smoothing [36] with the collection model $P(f_i|P)$, which is shown in Fig. 3.2. In other words, we replace the distributions $P(f_i|p)$ in Eq. (4.6) with an interpolation between the playlist and the collection as

$$P_\lambda(f_i|p) = (1 - \lambda)P(f_i|p) + \lambda P(f_i|P). \tag{4.7}$$

We set a threshold θ such that if the ℓ_1 distance is bigger than θ, the playlist is discarded from the graph. To reduce the risk of thinning the graph too much, we only apply feature-based prefiltering if the resulting graph contains at least 150 playlists. An additional criterion is that the query playlist contains at least five tracks, for the simple reason that the histogram model of a very short playlist is not informative.

For efficiency, we compute the collection model and playlist histograms for all playlists in the collection in advance. At runtime we only compute the query histogram and its distance to all other playlists based on Eq. (4.6) and Eq. (4.7).

4.4.3 Degree Pruning

Another addition to our methods is to clean the graph similar to the Pinterest graph pruning of Eksombatchai et al. [5]. They argue that the skewed heavy-tailed distribution of the Pinterest graph causes the random walk for high-degree nodes to be diffused among a very large number of nodes. In their work, they found that it was benificial both in terms of memory consumption and recommendation quality to systematically remove edges for high-degree nodes. Since we also observed a skewed

heavy-tailed distribution for tracks (see Fig. 3.1), we use a similar approach. Tracks are often included in playlists where they do not fit the rest of the songs. We therefore use the playlist histogram models to determine how related all tracks are to the rest of the songs for every playlist in which they are included, and discard edges between tracks and playlists where a track does not belong to the playlist based on the features described before. High-degree tracks are pruned more heavily than low-degree tracks due to an exponential pruning factor, described next.

The amount of edges removed is based on the pruning factor δ ($0 \leq \delta \leq 1$). The degree of every track node t is updated to $\deg(t)^{\delta}$, thus removing the track from a number of playlists in the graph. Note that high-degree tracks are pruned more heavily than less popular tracks. We rank the neighboring playlists based on the likelihood of drawing the track t from the playlist p. Intuitively, if there are many similar tracks to t in playlist p, the likelihood of drawing the same set of features from the playlist model is higher than when there are very few similar tracks. Since a track is defined as a set of independent feature values f_i for feature dimensions i, the track likelihood is

$$P(t|p) = \prod_{i=1}^{|F|} P(f_i|p). \tag{4.8}$$

We take the log of above equation to avoid underflowing errors, resulting in an equivalent ranking following the equation

$$\log P(t|p) = \sum_{i=1}^{|F|} \log P(f_i|p). \tag{4.9}$$

Whenever a feature value is missing for a track t, we simply skip that feature in the sum in above equation. The playlists are sorted based on the log-likelihood, and edges between t and the playlists with lowest log-likelihood are removed until the number of edges is equal to $\deg(t)^{\delta}$.

Chapter 5

Evaluation

In this chapter, we evaluate the methods proposed in the previous chapter. The evaluation is performed on a validation set that was built from the MPD. The validation set and additional setup for the experiments are described in Section 5.1, as well as the metrics for evaluation. The random walks with restart are evaluated in Section 5.3. We then evaluate the title-based retrieval in Section 5.4, the title-based prefiltering in Section 5.5 and the feature-based prefiltering in Section 5.6. We conclude the chapter in Section 4.4.3 with an analysis of the degree pruning. In the next chapter, we summarize the results of our evaluation on both the validation set and the challenge set.

5.1 Experimental Setup

In this section, we describe the setup used for running experiments and comparing different methods and settings. We create a validation set by selecting a number of playlists and splitting them into a query set and a held-out set. The validation playlists are selected following the structure of the challenge set. This ensures that the evaluation on the validation set is representative of the performance on the challenge set. As described in Section 3.2, the challenge set consists of ten categories of incomplete playlists of various lengths. We sample 1000 playlists from the MPD for every category, giving a total of 10,000 playlists in the validation set. The playlists are sampled such that there are enough held-out tracks when splitting the playlists. This is ensured using the following sampling steps:

1. Sample 2000 playlists with at least 150 tracks for category 9 and 10
2. Sample 2000 playlists with at least 38 tracks for category 7 and 8

3. Sample 2000 playlists with at least 20 tracks for category 5 and 6
4. Sample 4000 playlists with at least 10 tracks for category 1 - 4

This order was chosen to ensure that there are enough samples per category since at every step the required playlist length is reduced. Samples are drawn without replacement, so none of the playlists occur multiple times in the validation set. We seed the sampling so that the validation set is constant across experiments. Early experiments showed that changing the seed has little influence on the performance of the random walk, so we conclude that the sample size is large enough. For every playlist, the random walk is run with the query tracks as input. We ensure that the held-out tracks are hidden by disabling all edges from and to the query playlist node.

A set of 500 recommended tracks are generated from the query tracks and compared to the held-out tracks. The recommendations are scored based on three metrics, in accordance with the evaluation methods on the test set of the ACM RecSys Challenge 2018 [8]. These metrics will be described in the next section. Additionally, we analyze whether performance differs across different strata for different settings. Playlists containing mostly popular songs might require a different treatment than niche playlists. Similarly, the optimal settings may differ when presented with a short query playlist instead of a long one. If this is indeed the case, then a *switching hybrid* recommender [37] would lead to improved recommendation quality. Switching hybrid recommenders can combine the strengths of its constituents at the cost of additional parameters for the switching criteria [38]. Therefore, we stratify the performance on what we consider to be the two most distinct dimensions: the number of query tracks $(0, 1, 5, 10, 25, 100)$ and the mean of the Count feature of the query tracks, binned into ten bins $[0.0, 0.1), [0.1, 0.2), \ldots, [0.9, 1.0]$. The latter was chosen to be able to distinguish between niche playlists and playlists with mostly popular songs. For brevity, we only report stratified results when they lead to new insights.

5.2 Metrics

The following evaluation metrics are defined per playlist and averaged across the validation set when comparing different systems. In the following equations, G denotes the set of held-out tracks for the playlist and R denotes the set of recommended tracks. The first evaluation metric is *R-Precision*, defined as the number of relevant recommendations divided by the number of withheld tracks:

$$\text{R-precision} = \frac{|G \cap \{r_1, \ldots, r_{|G|}\}|}{|G|}. \tag{5.1}$$

The second evaluation metric is *Normalized Discounted Cumulative Gain* (NDCG). The NDCG is defined as the DCG divided by the ideal DCG:

$$\text{NDCG} = \frac{\text{DCG}}{\text{IDCG}}, \tag{5.2}$$

where the DCG is a measure rewarding relevant tracks more when they are more highly ranked. The DCG is in our case defined as

$$\text{DCG} = rel(r_1) + \sum_{i=2}^{|R|} \frac{rel(r_i)}{\log_2 i}. \tag{5.3}$$

The ideal DCG is the DCG for an ideal ranking, defined as

$$\text{IDCG} = 1 + \sum_{i=2}^{|G|} \frac{1}{\log_2 i}. \tag{5.4}$$

In above equations, the binary relevance function is defined as

$$rel(r) = \begin{cases} 1 & \text{if } r \in G \\ 0 & \text{if } r \notin G. \end{cases} \tag{5.5}$$

The third metric is the *recommended songs clicks* (RS Clicks), and is particular to the RecSys Challenge 2018 [8]. This metric stems from a feature in Spotify where users receive a set of ten recommendations, after which users can refresh the list to produce an additional 10 tracks. The recommended songs clicks metric is defined as the number of times a user would have to refresh before finding a relevant track:

$$\text{clicks} = \left\lfloor \frac{\text{argmin}_i(r_i : r_i \in G) - 1}{10} \right\rfloor. \tag{5.6}$$

If there is no relevant track in R, the recommended songs clicks is set to 51 since the maximum value is 50 clicks for a set of 500 recommended tracks.

5.3 Multiple Random Walks

We first evaluate the effectiveness of the multiple random walks with restart through the full playlist graph. The parameters of this algorithm (Algorithm 4.2) are the

restart probability α and the number of steps per playlist N. We tune these parameters by running the random walks with various settings on the validation set described in the previous section.

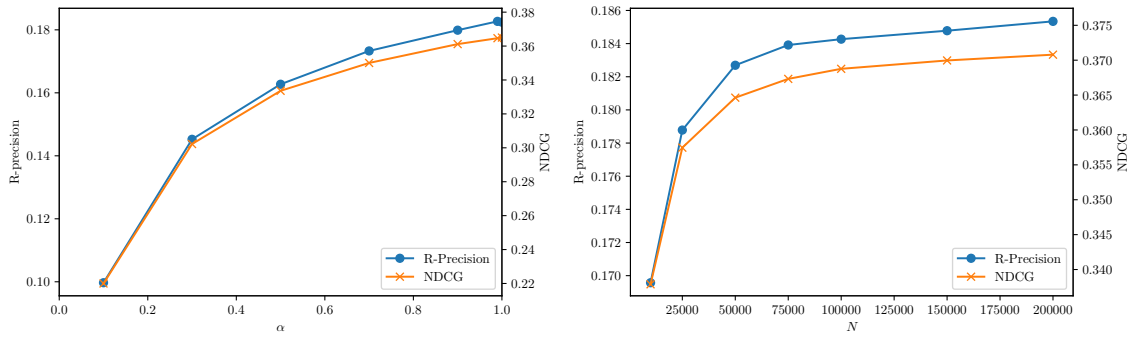
The restart probability was tuned first using $N = 50000$. The performance in terms of the R-Precision and NDCG are shown in Fig. 5.1a. Surprisingly, the performance increases drastically with a higher restart probability up to a value of 1.0. Recall that when the restart probability is 1.0, the random walk is reduced to a single step (track $\rightarrow$ playlist $\rightarrow$ track) from the starting node before restarting [31]. We find in these initial experiments that the performance is optimal with $\alpha = 1.0$ or slightly below 1.0. Fig. 5.1b shows the performance against the number of steps. As the number of steps increases, so does the recommendation performance. There is a trade-off between performance and runtime: the runtime of the random walks scales linearly with the number of steps [5]. The performance gains for more than 50000 steps are relatively small, so we run the rest of our experiments using $N = 50000$ to save computation time. The recommended songs clicks followed the same trend as the other metrics: a higher number of steps and higher values for α lead to a lower number of clicks. Note that low values are desirable for this metric, unlike R-Precision and NDCG.

The recommended songs clicks for the run with $\alpha = 0.99, N = 50000$ was 2.3682. Table 5.1 shows the performance of this run stratified on the number of query tracks. We see that performance is much better for larger query playlists. Since the N steps are distributed evenly over the query nodes, the runtime does not increase with more nodes. In fact, more query nodes allow for a larger amount of parallel execution, which leads to a shorter runtime. We did not notice any interesting patterns when comparing different values for α or N stratified on the number of query tracks or mean popularity of the query tracks.

5.4 Title-based Retrieval

As described in Section 3.2, one tenth of the challenge set consists of cold-start playlists with just the title given. It is to be expected that performance is much lower in this setting, since the playlist titles are noisy and provide very little information to use for building a recommender. In this section, we evaluate the title-based retrieval system described in Section 4.3.3. Since our validation set follows the same structure as the challenge set, we naturally use the subset of 1000 validation playlists corresponding to category 1 for evaluation.

After retrieving the most common tracks using title-based retrieval, we run the random walks starting at these tracks using $\alpha = 0.99, N = 50000$. The performance



(a) (b)

Figure 5.1: (a) Performance of the random walks against the restart probability α with $N = 50000$. (b) Performance of the random walks against the number of steps N with $\alpha = 0.99$. We omit the recommended songs clicks for visual clarity, this metric is discussed in the text.

# Query Tracks	R-Precision	NDCG	RS Clicks
0	0.1019	0.2130	11.5532
1	0.1366	0.2815	4.0090
5	0.1818	0.3618	2.0345
10	0.1961	0.3924	1.1400
25	0.2047	0.4068	0.6595
100	0.2115	0.4147	0.2445
Average	0.1827	0.3646	2.3682

Table 5.1: Performance stratified on the number of query tracks using $\alpha = 0.99$, $N = 50000$ and title-based retrieval for the first row. We consider this recommender to be the baseline approach.

of the title-based retrieval followed by random walks is shown in the first row of Table 5.1. To give an impression about the cases where title-based retrieval performs well, we show the top 20 in Table 5.2 based on the R-Precision metric. Playlist titles that correspond to soundtracks of movies, series, a specific artist or album perform very well. In these cases, the title provides a large amount of information about the desired characteristics, and there is only a very small set of tracks that fit these characteristics.

5.5 Title-based Prefiltering

We next evaluate whether the title-based prefiltering improves recommendation performance. As described in Section 4.3.2, we discard playlist nodes at runtime for which the title of the playlist is entirely dissimilar to the title of the query playlist. This is measured using the BM25 score. We use a similar experimental setup as Pichl et al. [7] and evaluate three different recommenders: (1) a baseline approach using the random walk described in the previous section, with title-based retrieval for the zero-length playlists, (2) the same recommender with title-based prefiltering applied for every query playlist, and (3) a hybrid or switching recommender. For the hybrid recommender, we first compute the R-precision, NDCG and RS Clicks of recommender (1) and (2) per title. Given a query playlist, we switch between these two recommenders based on majority voting: we choose the recommender that wins for at least two out of three metrics.

Table 5.3 shows a comparison of the performance of the three recommenders on the validation set. The baseline recommender (1) outperforms recommender (2) where prefiltering is applied to every playlist by a large margin for all three metrics. The hybrid recommender (3) outperforms both methods, but only by a small margin for the baseline recommender. By definition, the hybrid recommender always should perform at least equal or better than the other recommenders since it combines the best of the two. Nevertheless, we do not use the hybrid recommender in other experiments since the performance gain is small (1.7% for R-Precision, 0.7% for NDCG, 2.7% for RS Clicks) and we run the risk of overfitting. The performance of recommender (2) and (3) relative to recommender (1) was constant across the strata defined at the end of Section 5.1. Fig. 5.2 shows some examples of titles for which title-based pruning is beneficial and some for which it is detrimental to the performance.

Playlist Title	R-Precision
Frozen*	1.0000
Hamilton*	0.9857
Les Mis*	0.9545
SONS OF ANARCHY*	0.9333
Pirates of the Caribbean*	0.9298
Jon Bellion†	0.8923
MGMT†	0.8571
Coldplay†	0.8056
Kevin Gates†	0.7945
One Tree Hill*	0.7778
Chris Stapleton†	0.7121
janet‡	0.7059
One Direction – Made In The A.M. (Deluxe Edition)‡	0.7000
Kanye†	0.6975
disney*	0.6482
drake :)†	0.6349
Disney Songs*	0.6265
OutKast†	0.5775
Twilight*	0.5714
Disney ⋆*	0.5652

* Title is likely related to a movie or series

† Title is likely related to an artist

‡ Title is likely related to an album

Table 5.2: Top performing playlist titles in terms of R-Precision for the title-based retrieval and random walk method within the title-only category (category 1).

Recommender	R-Precision	NDCG	RS Clicks
(1) Baseline	0.1827	0.3646	2.3682
(2) All prefiltered	0.1650	0.3301	3.7993
(3) Hybrid	0.1858	0.3671	2.3042

Table 5.3: Performance of different recommenders: (1) random walk with title-based retrieval for 0-length query playlists, (2) the same recommender with title-based prefiltering for all playlists and (3) a hybrid recommender combining the best results of (1) and (2).

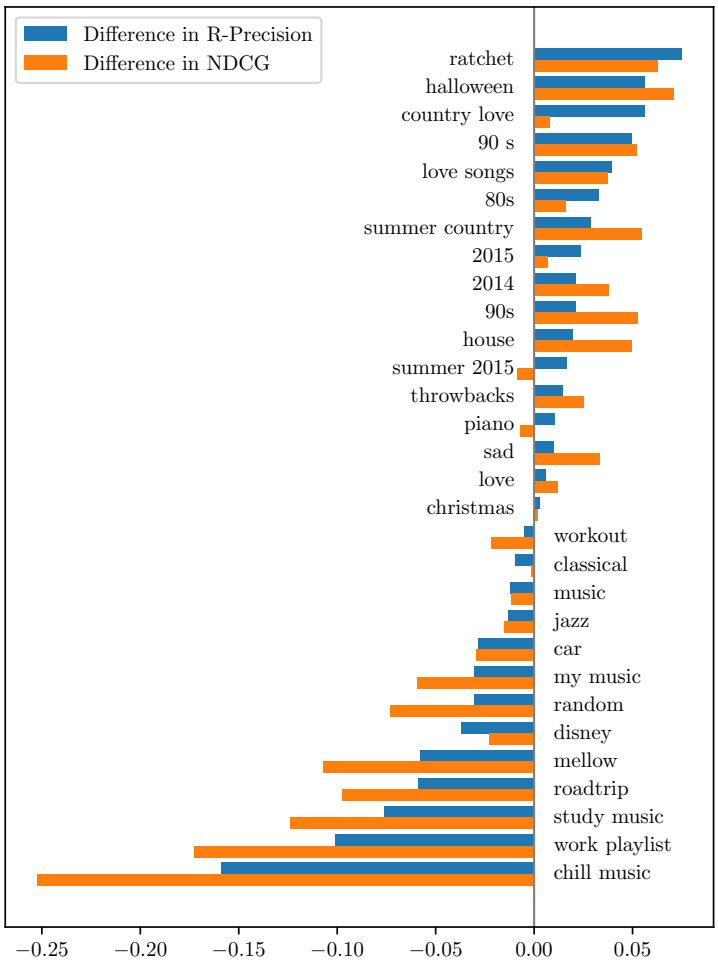


Figure 5.2: Performance gain when using the title-based recommender (2) versus the baseline recommender (1) described in Section 5.5 for a number of example titles. Positive values indicate an increased performance, negative values a decreased performance. Examples are sorted based on R-Precision. Recommended songs clicks is omitted for visual clarity.

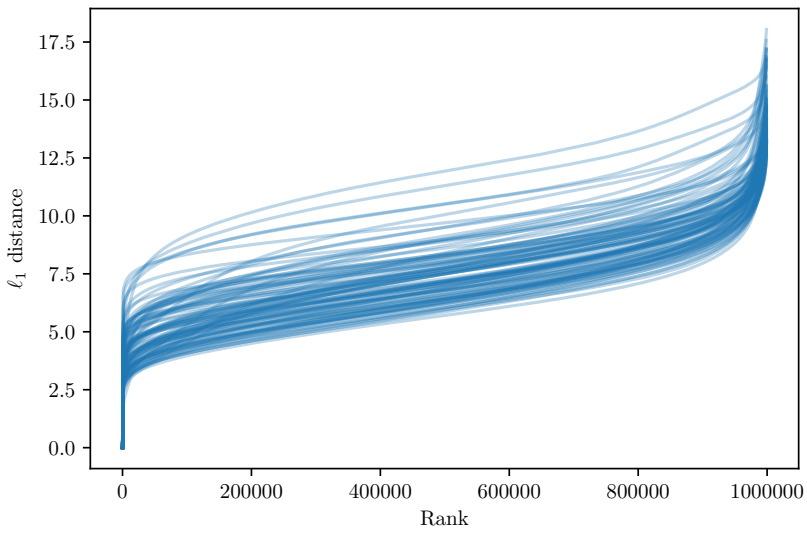


Figure 5.3: ℓ_1 distance between playlist histograms for 100 random sampled playlists versus all other playlists in the MPD. The x-axis corresponds to the remaining playlists in the MPD sorted by the distance to the random playlists. Overlapping lines are shown darker.

5.6 Feature-based Prefiltering

We now turn to an evaluation of the feature-based prefiltering method described in Section 4.4.2. Our feature-based prefiltering uses track metadata to build a histogram model for every playlist in the collection. We then filter playlists based on the ℓ_1 distance (Eq. (4.6)) between the collection playlists and the query playlists after applying Jelinek-Mercer smoothing (Eq. (4.7)). Our parameters are thus the amount of smoothing λ and the threshold θ above which playlists are removed from the graph. To simplify our analysis, we choose between $\lambda = 0$ and $\lambda = 0.25$ to determine whether smoothing is useful.

We first determine a suitable range of thresholds by sampling a set of 100 random playlists and ranking the rest of the playlists in the MPD based on the ℓ_1 distance between playlist histograms. This is shown in Fig. 5.3. We observe two knee points in all of the playlists, suggesting that the left knee point indicates a distance below which all playlists are very similar, and that the right knee point indicates a distance above which all remaining playlists are very dissimilar based on the features. Since

Recommender	R-Precision	NDCG	RS Clicks
Without smoothing			
$\theta = 4$	**0.1836**	**0.3655**	2.4047
$\theta = 5$	0.1825	0.3644	2.3593
$\theta = 7$	0.1812	0.3627	2.4428
$\theta = 10$	0.1826	0.3650	2.3695
With smoothing			
$\theta = 4$	0.1800	0.3596	2.3964
$\theta = 5$	0.1773	0.3552	2.5126
$\theta = 7$	0.1822	0.3640	2.4276
$\theta = 10$	0.1826	0.3647	2.3763
Baseline	0.1827	0.3646	**2.3682**

Table 5.4: Performance of recommenders that include feature-based prefiltering. Whenever Jelinek-Mercer smoothing is used, we set $\lambda = 0.25$. The threshold θ determines the amount of prefiltering. Note that prefiltering is only applied when the resulting graph contains at least 150 playlists.

most of the first knee points lie around a distance of 4.0, and the second around a distance of 10.0, we experiment with values between this range for the filtering threshold θ.

The results of feature-based prefiltering are shown in Table 5.4. The best performance in terms of R-Precision and NDCG is achieved without smoothing, $\theta = 4$. In general, smoothing does not lead to an increased performance compared to the recommenders without smoothing. We did not find any special cases where different settings for θ or λ were beneficial when stratifying on the number of query tracks or the mean count.

5.7 Degree Pruning

We finally evaluate the performance of graph cleaning by pruning edges from track nodes with high degree as described in Section 4.4.3. The amount of pruning is controlled with the pruning factor δ. Eksombatchai et al. [5] found that $\delta = 0.91$ is the optimal value for their work on Pinterest. Since they used a different distance measure, and the Pinterest graph structure is likely different from the playlist graph, we tune the δ to optimize for the present task.

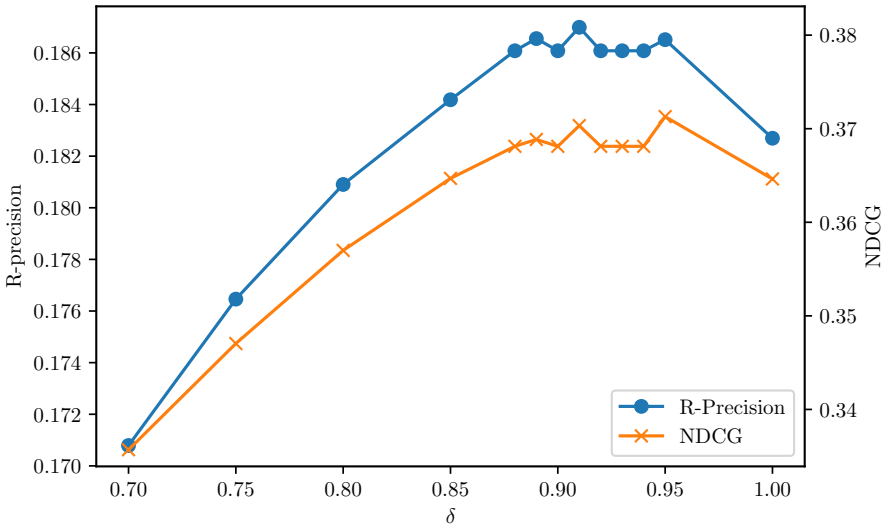


Figure 5.4: Performance of the random walks against the pruning factor δ.

Recommender	**R-Precision**	**NDCG**	**RS Clicks**
(1) Baseline	0.1827	0.3646	2.3682
(2) $\delta = 0.95$	0.1865	0.3712	2.3585
(3) Hybrid	0.1913	0.3770	2.3375

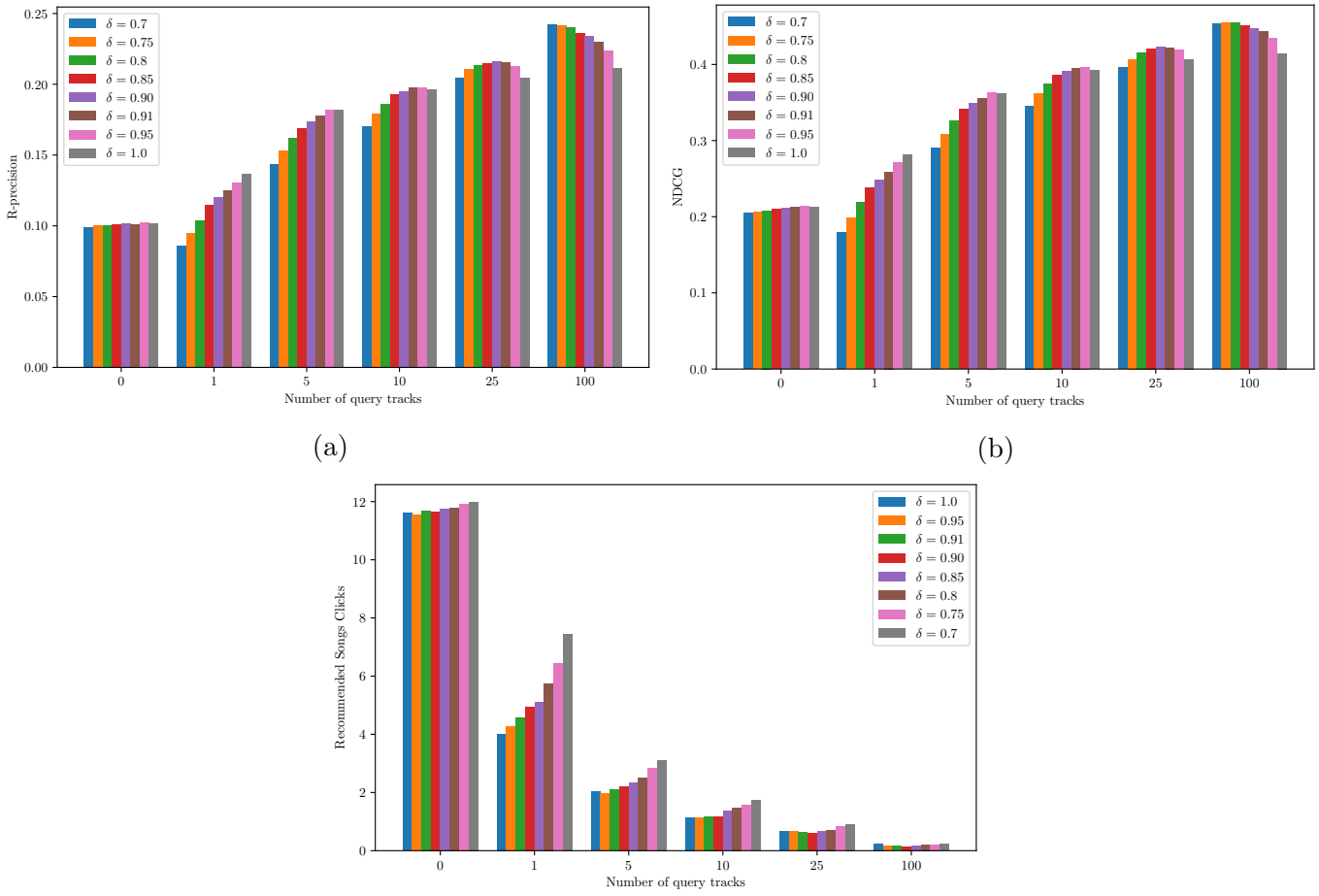
Table 5.5: Performance of different recommenders: (1) random walk with title-based retrieval for 0-length query playlists, (2) the same recommender with degree pruning factor $\delta = 0.95$ and (3) a hybrid recommender switching between δ's following Eq. (5.7).

Fig. 5.4 shows the performance of the random walk for different values. $\delta = 1.0$ corresponds to a random walk with no degree pruning, i.e. the baseline approach described previously. Lower values of δ correspond to a larger amount of edges pruned. We see from Fig. 5.4 that degree pruning leads to an improved performance in terms of R-Precision and NDCG. For R-Precision, the score peaks at $\delta = 0.91$ and for NDCG at $\delta = 0.95$. The RS Clicks metric is lowest at $\delta = 0.95$ with a value of 2.3585, which is an improvement upon the baseline (RS Clicks = 2.3682).

While the relative performance of different settings for δ did not change for niche or popular playlists, we found that there is an interesting pattern when stratifying on the number of query tracks. The performance of the random walk against the number of query tracks for different settings of δ is shown in Fig. 5.5. Degree pruning only leads to an improved performance when the set of query tracks is sufficiently large. For smaller query playlists, it is useful to prune less or not at all. Based on these findings, we propose a hybrid switching recommender that switches between different values of δ based on the number of query tracks $|\mathbf{q}|$:

$$\delta = \begin{cases} 0.75 & \text{if } |\mathbf{q}| \geq 100 \\ 0.9 & \text{if } 25 \leq |\mathbf{q}| < 100 \\ 0.95 & \text{if } 5 \leq |\mathbf{q}| < 25 \\ 1.0 & \text{if } |\mathbf{q}| < 5 \end{cases} \tag{5.7}$$

The switching criteria were chosen based on the optimal performance for different query playlist lengths shown in Fig. 5.5. Table 5.5 summarizes the results of degree pruning. The hybrid switching recommender improves upon both the baseline approach and the recommender with degree pruning using a single value for the pruning factor, $\delta = 0.95$.



(a) (b)

(c)

Figure 5.5: Performance of the random walk using various settings for graph pruning factor δ stratified on the number of query tracks. (a) Performance in terms of R-Precision. (b) Performance in terms of NDCG. (c) Performance in terms of Recommended Songs Clicks.

Chapter 6

Results

We briefly summarize the results of our evaluation in this chapter. Table 6.1 shows the performance of the best performing recommenders that were presented in the previous sections. We test the significance of the difference in performance between the baseline and the other recommenders with a Wilcoxon signed-rank test since the scores were non-normal. All proposed extensions lead to an increase in performance over the baseline system in terms of R-Precision and NDCG. The same holds for RS Clicks, except for feature-based prefiltering. For the first two metrics all differences were significant. The hybrid degree pruning recommender gave the biggest increase in performance compared to the baseline.

Jannach et al. [39] note in their research that many playlist generation systems suffer from biases, most notably popularity bias. Based on their research, we show the magnitude of which our systems are biased for the features defined in Section 3.3 in Fig. 6.1. We calculate bias for all features as the difference between the mean of the recommendations and the mean of the query tracks. Indeed, most systems suffer from a number of biases. Besides observing a strong popularity bias in the Count feature, we also see a bias towards explicit tracks. Small negative biases are observed towards acoustic tracks (preferring electronic music) and instrumental tracks (preferring tracks with vocals). We also see that danceable tracks are slightly favored above tracks without danceable features. The worst offender is a system with restart probability α lowered to 0.5, which confirms that longer walks tend to drift further away from the target characteristics of the input playlist. All proposed extensions reduce biases, and especially degree pruning leads to recommendations that are in general much more consistent with the query playlist.

We show the performance on the challenge set in Table 6.2. The recommenders used the same parameters as the runs reported in Table 6.1, except the number of

Recommender	R-Precision	NDCG	RS Clicks
Baseline	0.1827	0.3646	2.3682
Hybrid title-based pf.	0.1858†	0.3671†	**2.3042**
Feature-based pf.	0.1836*	0.3655†	2.4047
Degree pruning	0.1865†	0.3712†	2.3585*
Hybrid degree pruning	**0.1913**†	**0.3770**†	2.3375

* Significant with $p < 0.05$

† Significant with $p < 0.01$

Table 6.1: Summary of the performance of different recommenders on the validation set. Significance of the difference between the baseline performance and others is tested using a Wilcoxon signed-rank test.

Recommender	R-Precision	NDCG	RS Clicks
Baseline (50%)	0.1956	0.3535	**2.2566**
Degree pruning (50%)	0.1982	0.3569	2.3442
Hybrid degree pruning (50%)	**0.1985**	**0.3571**	2.3328
Hybrid degree pruning (100%)	TBD	TBD	TBD

Table 6.2: Summary of the performance of different recommenders on the challenge set. The number of steps N is doubled to 100,000 to improve performance for the challenge. Due to the setup of the RecSys Challenge, the baseline and degree pruning recommenders were only evaluated on 50% of the challenge set. Only the hybrid degree pruning recommender was evaluated on both the 50% challenge set and the full challenge set.

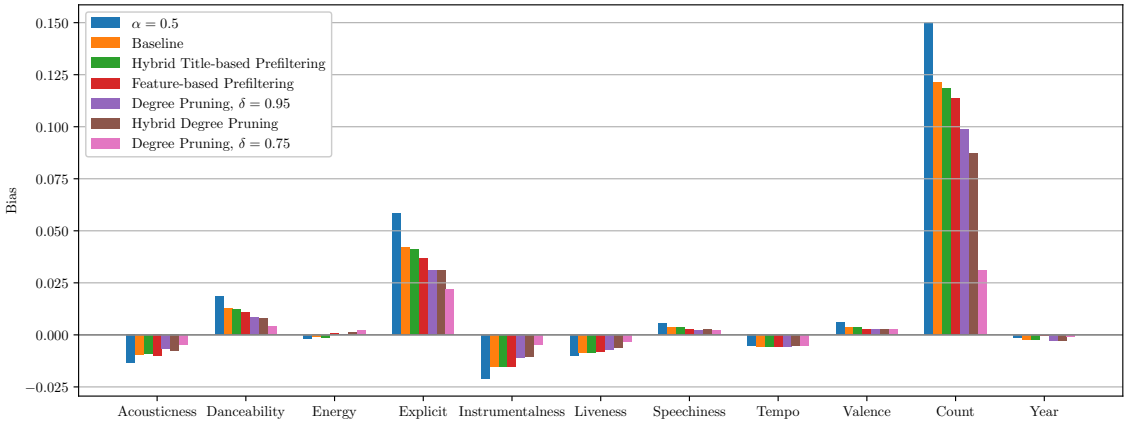


Figure 6.1: Recommendation biases over all features introduced by different recommenders. The bias is calculated for all features as the difference between the mean of the recommendations and the mean of the query tracks.

steps N which was doubled to 100,000. Due to time constraints, we were unable to evaluate all systems on the challenge set. We therefore chose the baseline approach as well as the degree pruning and hybrid degree pruning recommender. The last two recommenders were chosen because they were most promising with respect to the scores on the validation set. Additionally, the setup of the RecSys Challenge 2018 caused that only the hybrid degree pruning recommender could be evaluated on the full challenge set. The scores reported are the result of an online submission system, which used 50% of the challenge set for evaluation. As a precaution against overfitting on the challenge set, the score of the latest submission was recalculated on the full test set after the challenge closed. We therefore advise the reader to consider the scores on the challenge set as indicative only. In our case, we selected the hybrid degree pruning recommender as our contender.

The hybrid degree pruning recommender performed best on the 50% challenge set out of the three submitted systems in terms of the R-Precision and NDCG. The baseline system outperformed both degree pruning recommenders in terms of the Recommended Songs Clicks.

Chapter 7

Discussion

In this chapter, we discuss the findings described in the previous chapter as well as the shortcomings of our approach. We first interpret the results of the multiple random walks through the playlist graph. The proposed extensions are then discussed: title-based retrieval, title-based prefiltering, feature-based prefiltering and degree pruning.

In general, the proposed method based on random walk with restart has some nice properties that make it an interesting choice for automatic playlist continuation. The first main strength of the methods proposed is the scalability. Eksombatchai et al. [5] show that the runtime of random walk with restart is linearly related to the number of query tracks, but independent of the graph size. These properties translate to our work, and we further improve the runtime by parallelizing multiple walks. The second strength of our system is the flexibility and extensibility, as evidenced by the numerous additions proposed in this work. The methods can be applied to any set of tracks that are available in the playlist graph, or even an empty playlist with only a title. The final strength is the performance of the system, which we believe to be highly competitive based on the results of the RecSys Challenge 2018.

We based our multiple random walk approach on the Pixie recommender for Pinterest [5]. The graph representation and random walk algorithms were easily adapted for this work. We left out step allocation based on node degree and multi-hit boosting since these methods led to a worse performance in all preliminary experiments. Our baseline recommender performed well on the 50% challenge set, and most extensions only led to small improvements.

Perhaps our most surprising finding was that very high values for the restart probability α lead to the best performance. Note that Eksombatchai et al. [5] do not report the optimal restart probability. For a restart probability of 1.0, our single random walk algorithm is equivalent to the random-random model [31], which has

been shown to perform well on a link prediction task within social networks. We argue based on our results in Fig. 6.1 that long random walks are likely to diverge very quickly, and lead to a strong popularity bias since high-degree nodes are much more likely to be visited. The other biases stem from the distribution of the high-degree nodes: in our dataset, many popular tracks contain explicit lyrics and feature strong electronic beats.

The title-based retrieval on category 1 (no tracks, only title) consistently gave worse performance than all other categories. Nevertheless, we believe that our methods are promising, and that the difference in performance is largely due to the difficulty of the task given so little information. Title-based retrieval is most useful when the titles indicate a very specific set of relevant tracks. We observed that this is often the case when the title directly refers to an artist, album or the soundtrack to a movie or series.

We additionally conducted experiments with title-based prefiltering. Prefiltering on all titles led to a decreased performance. The hybrid system did lead to an improvement upon the baseline, which is expected due to its definition. The titles shown in Fig. 5.2 support the findings of Pichl et al. [7]. There are many playlist names which are not valuable because they contain no information about the playlist contents (e.g. 'my music', 'random'). Additionally, many titles do not share a common sense (e.g. 'chill music', 'work playlist', 'roadtrip'), while some titles do (e.g. 'halloween', 'love songs', '90s'). Other than this, it is difficult to compare our work with that of Pichl et al. [7] since they use a different dataset, different recommendation strategies, and different title processing techniques. The fact that we observed only a very small increase in performance for the hybrid recommender suggests that a large majority of the playlist titles are uninformative or do not share a common sense. This is also likely the reason for a drop in performance when prefiltering for all titles: for these cases there is likely no simple relation between title similarity as scored using BM25 and playlist contents.

Another proposed extension was the feature-based prefiltering system. This extension led to a negligible increase in performance when the threshold θ was set to a relatively low value of 4 and no smoothing was applied. We noticed in our experiments that when the threshold is this low, the criterion that at least 150 playlists need to remain in the graph is very rarely met. This explains why feature-based prefiltering only leads to a small reduction in biases, as shown in Fig. 6.1. For those playlists, the random walks were run as in the baseline recommender. Our results suggest that queries that share very specific characteristics with a large number of playlists, feature-based prefiltering is beneficial. We therefore think that our proposed feature-based prefiltering is still promising in some specific cases, and future

research is required to determine when.

The most useful extension turned out to be the degree pruning. We conclude that the graph cleaning methods of Eksombatchai et al. [5] were also useful in the APC setting, even though our playlist representation and pruning methods are very different. Eksombatchai et al. proposed graph pruning with the reasoning that many pins are mis-categorized by users, and that high degree nodes diffuse the random walk. Our results indicate that the same holds for playlists: we found that degree pruning reduces popularity bias immensely, as shown in Fig. 6.1. This supports the theory that high degree nodes cause diffusion in the random walk. An additional result is that we found that larger query playlists benefit from more pruning, while for smaller query playlists it is better to prune very little or not at all. We believe that a harsh pruning regime causes too many edges to be removed from the neighborhood for single-node queries, but summing visit counts over a large number of input tracks compensates for the resulting sparsity.

We now note some shortcomings of this work. Even though the runtime of the random walk is constant with respect to the graph size, a single run over the validation set still took approximately 8-9 hours. To save computation time, we opted to use a relatively simple playlist model and distance metric, while more sophisticated models might lead to further improvements. Additionally, some combinations of parameters were left untested. Specifically, we were unable to combine the different recommenders into a system that incorporates all extensions. Additionally, we did not evaluate all systems on the challenge set for the same reasons as well as limitations imposed by the RecSys Challenge 2018. However, in spite of the limitations recognized, our experiments should provide a solid understanding of the challenges and possible solutions for automatic playlist continuation.

Chapter 8

Conclusion

In this work we presented a random-walk-based method for automatic playlist continuation (APC). Our methods were developed as part of the RecSys Challenge 2018, for which a dataset of 1,000,000 playlists was released: the Million Playlist Dataset.

We defined the bipartite playlist graph, an efficient data structure for representing a large collection of playlists and tracks. We adapted the multiple random walk with restart to the setting of APC. The performance for the random walk with restart was evaluated on the Million Playlist Dataset, as well as the challenge set corresponding to the RecSys Challenge 2018. We found that extremely short walks with only two steps on average performed best. This leads us to believe that random walks tend to diverge very quickly in the case of the playlist graph. Preliminary scores on the challenge set showed that the random walk with restart, which we used as a baseline, already achieves a competitive performance in the context of the challenge. Additionally, our methods are highly scalable since the multiple random walks run in constant time with respect to the graph size and are easily parallelizable. Another advantage is that our recommenders are very flexible and extensible.

To account for the scenario in which a user has not yet added tracks to a playlist, but has given a name, we proposed a title-based retrieval method. As expected, this task is more difficult than the case where some query tracks are given. Nevertheless, we identified a number of scenarios for which title-based retrieval can give excellent performance. We investigated whether prefiltering based on the playlist title or track features led to higher quality recommendations. Both methods only led to small improvements upon the baseline. We proposed a recommender in which edges between high degree tracks and playlists were removed based on the distance to the remainder of the playlist. The degree pruning recommender yielded a sizable increase in performance above the baseline. We additionally observed that for short query

playlists, little to no pruning is better, while long playlists benefited from heavy pruning. The best performance reported in this work was achieved by combining multiple levels of pruning into a hybrid recommender; this system achieved an R-Precision of [TBD], an NCDG of [TBD] and a Recommended Songs Clicks of [TBD] on the challenge set.

Some directions for future work include combining the different extensions proposed, in order to see if performance can be improved further. Additionally, one can think of many more extensions that could combine the graph structure with metadata and audio features, such as biasing the random walk using weighted transitions, or adding new edges between playlists based on the track features or titles. A major hindrance for random walks is popularity bias. While we have proposed methods that reduce this bias, we believe that more insight can be gained using user-centric metrics or metrics that measure the coherence of recommendations. Our results show that random walk with restart using query-specific adaptations is a promising new direction for the domain of automatic playlist continuation.

The code for this thesis can be found at `https://github.com/TimovNiedek/recsys-random-walk`.

Bibliography

[1] "Announcing MIDiA's State Of The Streaming Nation 2 Report | MIDiA Research," https://www.midiaresearch.com/blog/announcing-midias-state-of-the-streaming-nation-2-report, accessed 2018-03-05.

[2] M. Schedl, H. Zamani, C.-W. Chen, Y. Deldjoo, and M. Elahi, "Current challenges and visions in music recommender systems research," *International Journal of Multimedia Information Retrieval*, vol. 7, no. 2, pp. 95–116, 2018.

[3] S. J. Cunningham, D. Bainbridge, and A. Falconer, "'More of an art than a science': Supporting the creation of playlists and mixes," 2006.

[4] J. H. Lee, B. Bare, and G. Meek, "How similar is too similar?: Exploring users' perceptions of similarity in playlist evaluation." in *ISMIR*. Citeseer, 2011, pp. 109–114.

[5] C. Eksombatchai, P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec, "Pixie: A system for recommending 3+ billion items to 200+ million users in real-time," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2018, pp. 1775–1784.

[6] "Pinterest," https://www.pinterest.com/, accessed 2018-03-06.

[7] M. Pichl, E. Zangerle, and G. Specht, "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, 2015, pp. 1360–1365.

[8] "RecSys Challenge 2018, the ACM Conference Series on Recommender Systems," https://recsys.acm.org/recsys18/challenge/, accessed 2018-03-05.

[9] "Spotify homepage," https://www.spotify.com, accessed 2018-03-05.

[10] "Million playlist dataset," official website hosted at https://recsys-challenge.spotify.com, accessed 2018-03-05.

[11] G. Bonnin and D. Jannach, "Automated generation of music playlists: Survey and experiments," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, p. 26, 2015.

[12] N. Tintarev, C. Lofi, and C. Liem, "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, 2017, pp. 391–392.

[13] A. Vall, M. Schedl, G. Widmer, M. Quadrana, and P. Cremonesi, "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, 2017.

[14] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[15] G. Jeh and J. Widom, "Scaling personalized web search," in *Proceedings of the 12th international conference on World Wide Web*. Acm, 2003, pp. 271–279.

[16] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized pagerank," *Proceedings of the VLDB Endowment*, vol. 4, no. 3, pp. 173–184, 2010.

[17] T. H. Haveliwala, "Topic-sensitive pagerank," in *Proceedings of the 11th international conference on World Wide Web*. ACM, 2002, pp. 517–526.

[18] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu, "Automatic multimedia cross-modal correlation discovery," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 653–658.

[19] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," 2006.

[20] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in social networks," in *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 2011, pp. 635–644.

[21] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh, "Wtf: The who to follow service at twitter," in *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 505–514.

[22] R. Lempel and S. Moran, "Salsa: the stochastic approach for link-structure analysis," *ACM Transactions on Information Systems (TOIS)*, vol. 19, no. 2, pp. 131–160, 2001.

[23] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós, "Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments," *Internet Mathematics*, vol. 2, no. 3, pp. 333–358, 2005.

[24] G. Adomavicius and A. Tuzhilin, "Context-aware recommender systems," in *Recommender systems handbook*. Springer, 2011, pp. 217–253.

[25] J.-J. Aucouturier and F. Pachet, "Scaling up music playlist generation," in *Multimedia and Expo, 2002. ICME'02. Proceedings. 2002 IEEE International Conference on*, vol. 1. IEEE, 2002, pp. 105–108.

[26] "Web API Reference | Spotify for Developers," https://beta.developer.spotify.com/documentation/web-api/reference/, accessed 2018-03-06.

[27] "Spotify - RecSys Challenge 2018," https://recsys-challenge.spotify.com/dataset, accessed 2018-04-04.

[28] "Challenge Set," https://recsys-challenge.spotify.com/challenge_readme, accessed 2018-06-27.

[29] "Get Several Tracks | Spotify for Developers," https://developer.spotify.com/documentation/web-api/reference/tracks/get-several-tracks/, accessed 2018-06-12.

[30] "Get Audio Features for a Track | Spotify Developer," https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/, accessed 2018-06-12.

[31] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins, "Microscopic evolution of social networks," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 462–470.

[32] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos, "Neighborhood formation and anomaly detection in bipartite graphs," in *Data Mining, Fifth IEEE International Conference on*. IEEE, 2005, pp. 8–pp.

[33] M. Pichl, E. Zangerle, and G. Specht, "Improving context-aware music recommender systems: Beyond the pre-filtering approach," in *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*. ACM, 2017, pp. 201–208.

[34] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[35] S. Robertson, H. Zaragoza *et al.*, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[36] F. Jelinek and R. Mercer, "Interpolated estimation of markov source parameters from sparse data." pp. 381–397, 401, 01 1980.

[37] R. Burke, "Hybrid recommender systems: Survey and experiments," *User modeling and user-adapted interaction*, vol. 12, no. 4, pp. 331–370, 2002.

[38] F. Isinkaye, Y. Folajimi, and B. Ojokoh, "Recommendation systems: Principles, methods and evaluation," *Egyptian Informatics Journal*, vol. 16, no. 3, pp. 261–273, 2015.

[39] D. Jannach, I. Kamehkhosh, and G. Bonnin, "Biases in automated music playlist generation: A comparison of next-track recommending techniques," in *Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization*. ACM, 2016, pp. 281–285.