Music Discovery: Exploiting TF-IDF to Boost Results in the Long Tail of the Tag Distribution

José Devezas jld@fe.up.pt
Filipe Coelho filipe.coelho@fe.up.pt
Sérgio Nunes ssn@fe.up.pt
Cristina Ribeiro mcr@fe.up.pt

Abstract

We tackle the problem of music discovery using a tag-based recommender system. We propose and analyze two recommendation algorithms, each with two parameters, to filter the number of tags considered, and assess the impact of changing each parameter individually for either algorithm. The goal is to improve the discovery impact by boosting results in the long tail of the tag distribution with the aid of the IDF metric. We automatically generate twenty playlists based on text search and use them as input to our recommendation algorithms. For each algorithm, we do seven different runs, with different combinations of parameters, and evaluate the results based on the user taste profile subset available as part of the Million Song Dataset. While there are clear challenges regarding the evaluation of the quality of a playlist and its discovery impact, we show that better rarity-driven results were achieved when taking advantage of the IDF metric.

1 Introduction

The way we listen to music and organize our playlists is a very personal action. The problem of generating good playlists for specific individuals, using a limited amount of data to characterize their listening behavior, is still an open challenge. Moreover, defining what a good playlist is and how it can be evaluated is a complex task. Playlists are usually generated with a given goal in mind, defined by the user and usually hard to predict. The objective of recommender systems is to enable the discovery of potentially interesting new items that the user hadn't previously considered. While discovery seems to be at the core of recommendation, many systems don't quite focus on this goal directly, but instead provide the users with similar music that they are bound to like but that lack a surprise factor. An example of a situation to avoid is the recommendation of more Beatles albums to someone who already likes Beatles songs. While, for particular contexts, such as marketing, this might be considered a successful and correct approach, when assessing the discovery impact of the suggestion, a user would easily indicate that the recommendation is redundant, as the consistency between Beatles albums is quite obvious and thus a simple search by the Beatles discography would easily achieve the same goal.

In this paper, we tackle the problem of playlist generation for music discovery, by analyzing the long tail of the tag distribution and proposing a different approach for harnessing the information given by the typical power law behavior of this distribution. We compare two recommendation methodologies based on the music folksonomy from the Million Song Dataset [1]. The first method uses a frequentist approach based on tag similarity, to obtain and rank a set of tracks according to the number of tags in common with the tracks in the input playlist; we filter tags according to their weight and at two different phases of the recommendation process. The second method replicates the same approach, but with an emphasis on the long tail, i.e. the least popular tags. However, instead of cutting the tag distribution into head and tail, we propose the application of the IDF (Inverse Document Frequency) to proportionally boost the least popular tags, instead of separately analyzing the most popular and the least popular tags.

2 Related Work

Playlist generation is a recommendation task that has been addressed in many forms, namely with social and content-based approaches. Indirectly, social behavior also establishes a context through music folksonomies. We explore the context provided by the long tail of the tag distribution to improve music discovery through the characterization of songs based on their most descriptive and overall less popular tags.

2.1 Music Recommendation: Playlist Generation

In his PhD thesis, Fields [4] has focused on the task of automated playlist generation for music recommendation. He proposed a novel multimodal similarity measure that integrates the social dimension with content-based features by taking advantage of the network of top friends for MySpace artists. He expanded this network by using songs as nodes and weighting the edges of this socially induced network with the similarity between songs using audio-based features. Fields pointed out that, while it is obvious that playlist generation highly depends on the relationships between songs, no previous work [prior to 2011] had acknowledged that fact. Fields also demonstrated a successful application of community detection methodologies to the artist-based song similarity network in order to cluster different musical genres successfully.

We also explore the task of automated playlist generation, but we focus on improving the music discovery impact of the recommended tracks.

2.2 Using Folksonomies for Recommendation

Sen et al. [11] presented a recommendation algorithm that took advantage of tags to improve movie suggestions. Their method, which they call a tagommender, was based on the automatic inference of tag preferences for a particular user, according to their interactions with tags and movies. The authors combined three signals for tag inference: (*i*) the tags a user applies to movies, (*ii*) the tags a user searches for, and (*iii*) the quality of the tags, given by the all-implicit inference algorithm, which is described in their previous work [10].

Knijf et al. [6] proposed a graph-based music recommendation algorithm supported by a bipartite graph, connecting artists via common tags. They used a probabilistic method, based on random walks with restart, to identify communities of related artists, as well as to discover the most probable paths between two given artists.

Firan et al. [5] evaluated the benefit of using tag-based profiles for music recommendation. They compared collaborative filtering with search-based techniques and experimented with different weighting schemes for tags, including the ITF (Inverse Tag Frequency) metric, aimed at diluting the bias of profiles towards highly used tags.

Similarly, in this work, we take advantage of a music folksonomy for recommendation and discovery, by focusing on the tags in the long tail to describe each track and enhance their uniqueness. We also consider the impact of tag popularity for the recommendation process, by taking advantage of a metric analogous to the IDF measure.

2.3 Music Discovery in the Long Tail

Using the long tail of popularity distributions to improve music discovery is not a novel approach. As we previously implied, other authors have already addressed this issue by cutting the distribution into head and tail, and by either using different methodologies for the two parts or by focusing solely on the tail.

Park and Tuzhilin [8] defined a cutting point α to divide the head from the tail in the item set distribution. They then used different recommendation methodologies for the head set and for the tail set: the *Each Item*, where they defined a custom data mining model for each item in the head set, and the *Clustered Tail*, where they defined a custom data mining model for each of the identified clusters in the tail.

Òscar Celma [2] has also focused on music recommendation by exploring the long tail of the song popularity distribution. The author has taken into consideration an important characteristic of music recommendation: discovery. A problem with recommender systems is that they tend to be biased towards suggesting popular content, which easily get a higher score simply due to the fact that most users like this content. Assuming that the process of discovery through a recommender engine is expected to result in higher gains than those obtained with traditional methods, and knowing that popular content is easily spread through traditional media and word-of-mouth, focusing on the items in the long tail should improve on the gain provided by a recommender system. Herrada proposed that music recommendation should simultaneously take advantage of the user-item similarity network and the popularity of the item, decreasing the score of the recommended item alongside its popularity, in order to increase the number of recommendations in the long tail.

This is also the approach we take in this paper, where we use the tag weight available in the Millions Song Dataset — a particular type of normalized Term Frequency (TF) — to measure tag relevance, and combine it with the tag IDF to boost tag relevance in the long tail. Given that our work is focused on a graph database implementation, which uses several Gremlin[1] traversal queries, we experiment with two parameters that limit the number of top tags at two different phases of the traversal. We analyze how these two parameters influence the outcome, and we also compare the long tail approach with a generic approach that disregards tag popularity.

3 Million Song Dataset

Our experiments are based on a selection of features from the Million Song Dataset (2011) [1, 7, 9]. The dataset is freely available and comprises a collection of audio features and metadata for a million contemporary popular music tracks. Additionally, the community has also contributed with several complementary datasets, including lyrics from musiXmatch, weighted tags from Last.fm, and a taste profile subset with over 48 million ⟨*user, song, play count*⟩ tuples. Table 1 characterizes the data available in the Million Song Dataset, focusing on the most relevant features for the scope of our work.

Given its dimension, this dataset is a perfect workbench to experiment with large-scale music information retrieval and recommendation. In the work we present here, we take advantage of the Last.fm tags to discover related songs based on an input playlist, generated using textual search. We then use the users' taste profile subset to evaluate the generated playlists.

[1]Gremlin is a DSL for graph traversal that works as a pipeline: http://markorodriguez.com/2011/08/03/on-the-nature-of-pipes/

Table 1: Million Song Dataset.

Size	280 GB
Songs	1,000,000
Artists	44,745
Releases	221,753
Last.fm Tags	522,366
Taste Profile Triplets	48,373,586

4 Graph-Based Recommendation

Juggle, our recommender system, uses a graph-based approach, implemented on top of the Neo4j graph database, which enables the creation of a knowledge model centered on the tracks and their features.

To prepare our database, the Million Song Dataset is first processed by generating a set of TSV (Tab-Separated Values) files: one containing the nodes and their properties, another one containing the relationships, their type and their properties, and several other files containing the properties to index for each of the types of nodes available. Each TSV file references the precomputed IDs for the nodes that will be inserted into an empty Neo4j graph database and stored using the graph property model, which distinguishes nodes, relationships (the edges) and properties (the data fields associated with the nodes and relationships). This enables us to run a batch import process and to quickly[2] obtain a graph-based knowledge model with nearly 3 million nodes, 60 million relationships and 60 million indexed properties.

After preparing the graph database, we use the Gremlin language to precompute some of the required properties, as well as to process the recommendation requests. Recommendation scripts are run through the REST API of the Neo4j server, using the provided Gremlin plugin.

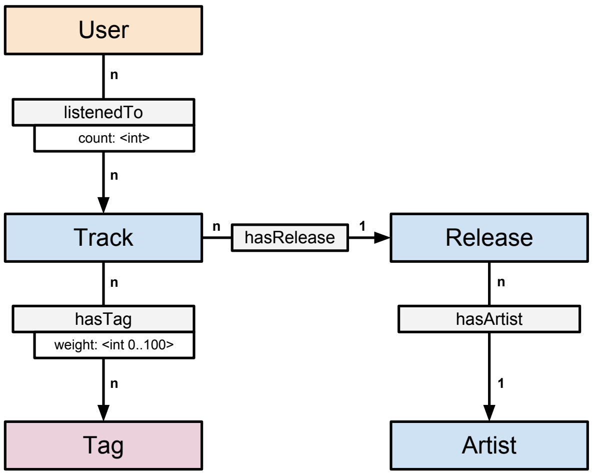

Figure 1: Juggle recommender system's knowledge model for the Million Song Dataset.

4.1 Knowledge Model

The knowledge model for Juggle is described in Figure 1. As shown in the figure, each song is represented by a track node and accompanied by a release node, which can be shared across several track nodes, and an artist node, which can be shared across many release nodes. Defining the release and album as nodes instead of properties of the track node is a way of enforcing consistency. An alternative would be to store this data in a separate database and to keep a property in each track that referenced the ID in the separate database. However, even though we currently don't use the release and the artist to filter recommendations, this is a valuable feature that we plan on using later, thus reinforcing the choice of storing them as nodes in the graph database. Finally, each track connects to several tags, with a weight that is based on a normalized term frequency value, and each user connects to several tracks, with a play count value representing the number of times the user listened to the specific track.

4.2 Data Preparation

For our experiment, we will need to calculate the TF-IDF value for a tag (our terms) in a given track (our documents). This process is done on-the-fly during the recommendation query, but requires the IDF to be precomputed for each tag. Equation 1 shows how the IDF is analogously calculated for tracks and tags. Let *tracks* be the set of tracks and $tags(t)$ the set of tags for track t:

$$IDF(tag, tracks) = log \frac{|tracks|}{|t \in tracks : tag \in tags(t)|} \tag{1}$$

IDF calculation is done by using the Gremlin language to traverse the graph, obtaining the value and storing it as a property of each tag node. For the knowledge graph g, we access the tracks index and do a Lucene query for all track IDs to obtain the total number of tracks, which we store in a variable $D = |tracks|$. Similarly, we do an index query to traverse all tag nodes and, for each tag, we obtain the number of in-links, i.e. the number of tracks tagged with the current tag. Finally, we calculate the IDF for a tag using the logarithm of the ratio between the total number of tracks D and the total number of tracks associated with the tag in the music collection.

4.3 Recommendation Queries

We describe the two recommender algorithms that are the base of this work. Although our recommender system, Juggle, uses a multimodal approach that currently combines context from the music folksonomy with user taste data for collaborative filtering, for the purpose of evaluation, we leave the user taste profile out of the experiment we present here and focus solely on the tag-based recommendation.

Algorithm 1 shows the generic recommendation strategy that we use, based on the tags and an associated metric representing the tag weight within a track. Algorithm 2 illustrates the traversal process that, given a list of track nodes, returns the top α tags for those tracks according to the selected metric. Algorithm 3 illustrates the traversal process that, given a list of tag nodes, returns the top β tracks represented by the given tags. While we use a functional approach to describe the Gremlin pipeline for the algorithm we implemented, we should point out that each non-scalar function returns an iterator instead of directly returning the

[2]The entire batch importing process, including the TSV export phase, takes less than two hours in a desktop machine with 4 GB of RAM and a 5,400 RPM hard disk.

Algorithm 1 Recommendation algorithm for a generic metric.

```
Input: $Q$ (query), $\alpha$ (top tags fraction), $\beta$ (top tracks fraction), $metric$ (TF or TF-IDF).
Output: A ranked list of recommended tracks.
  $tracks \Leftarrow$ SearchByTrackId($Q$)
  $tags \Leftarrow$ TopTags($tracks$, $\alpha$, by=$metric$)
  $recs \Leftarrow$ TopTracks($tags$, $\beta$, by=$metric$) $\setminus$ $tracks$

  $map \Leftarrow$ Dictionary with default zero values.
  for all $rec \in recs$ do
    Inc($map[rec]$)
  end for
  return DescendingSortByValue($map$).take(10)
```

Algorithm 2 TopTagsByMetric.

```
Input: $tracks$, $\alpha$, $metric$
Output: A list with the concatenation of the top $\alpha$ tags with highest weight for each of the input $tracks$.
  $tags \Leftarrow$ Empty list.
  for all $track \in tracks$ do
    $n_{tags} \Leftarrow$ CountTags($track$)
    $tags_{track} \Leftarrow$ HasTags($track$)
    $tags_{track} \Leftarrow$ DescSort($tags_{track}$, by=$metric$)
    $tags_{track} \Leftarrow$ Head($tags_{track}$, $\alpha \times n_{tags}$)
    $tags \Leftarrow$ Append($tags$, $tags_{track}$)
  end for
  return $tags$
```

Algorithm 3 TopTracksByMetric.

```
Input: $tags$, $\beta$, $metric$
Output: A list with the concatenation of the top $\beta$ tracks with highest weight for each of the input $tags$.
  $tracks \Leftarrow$ Empty list.
  for all $tag \in tags$ do
    $n_{tracks} \Leftarrow$ CountTracks($tag$)
    $tracks_{tag} \Leftarrow$ TracksHaveTag($tag$)
    $tracks_{tag} \Leftarrow$ DescSort($track_{tag}$, by=$metric$)
    $tracks_{tag} \Leftarrow$ Head($tracks_{tag}$, $\beta \times n_{tracks}$)
    $tracks \Leftarrow$ Append($tracks$, $tracks_{tag}$)
  end for
  return $tracks$
```

elements, which improves performance and memory usage.

Besides the metric selection parameter, there are three additional arguments that must be passed to our algorithm:

Q A Lucene query to obtain the set of track nodes that represent the input playlist.

α A float value between 0 and 1, representing the fraction of top tags to consider, according to either the TF, given by the tag weight on a track, or the TF-IDF of the tag within the input track.

β A float value between 0 and 1 representing the fraction of top tracks to consider, according to either their TF weight value or their TF-IDF value, for the traversed tags, i.e. for the tags of the input tracks.

metric The metric used for the ranking of the top tags and top tracks. This can either be the TF or the TF-IDF metric. While the TF is directly represented by the weight property in each track-tag relationship, the TF-IDF is calculated on-the-fly using the formula: $(weight(tag)/100) \times IDF(tag, tracks)$.

The value for the track-tag relationship weight represents the tag frequency within the associated track, and is normalized using the maximum tag frequency over all tags for the track, and then multiplied by 100. Our first recommendation algorithm uses this TF value as the metric for tags and tracks.

The second recommendation algorithm is based on tags and their TF-IDF relatively to a track. The difference to the previous algorithm is that now we combine the IDF with the tag weight to boost tags in the long tail. This means that we avoid track comparisons based on generic tags such as *rock*, *metal* or *electronic*.

In either algorithm, the method is to find the node representation for the input playlist tracks, traverse the graph through the tags of the input tracks, to arrive at new tracks, and count the number of times there was a traversal through each track node. We use this frequency to rank recommendation results and filter out tracks in the input playlist. Given the large-scale of the collection, it is important to develop filters that will spare system resources during processing. We do this in different ways for the two algorithms: either by selecting

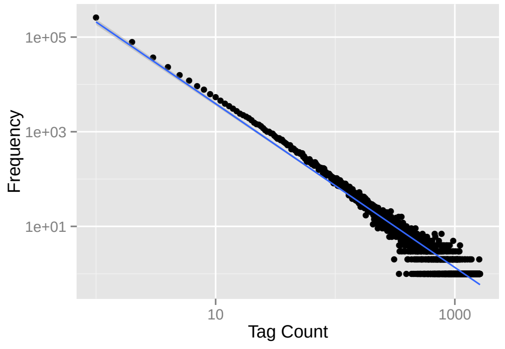

Figure 2: Tag count distribution in log-log scale.

the top tags using the tag weight (the TF), or by selecting the top tags using the TF-IDF of the tag within the respective track. While TF-IDF is currently calculated on-the-fly during the recommendation phase, this value could be precomputed and stored as a property for each *hasTag* relationship to speedup the process.

5 Experiment

As we can see from Figure 2, the distribution of tags in the Million Song Dataset closely fits a power law. This highlights the problem we are trying to address here, as it shows that a small number of tags have a high popularity and are widely used to characterize songs, while a large number of tags have a low popularity and are more sparse across the collection. Some of the tags in the head of the distribution are depicted next, ranked by frequency: *pop*, *indie*, *female vocalists*, *rock*, *alternative*, and *electronic*. And some of the tags in the tail, with frequency 1, are illustrated next: *Car ride sing a long*, *the word firecracker*, *guilty-pleasure*, *eliane*, *bossa bova*, and *Eliane elias - Segredos Secrets*.

Visibly, head tags tend to contain more generic terms, such as *pop* or *rock*, while the tail contains tags that are either more specific, like *guilty-pleasure*, a typo, like *bossa bova*, or something directly related with the artists or the release itself, like *eliane* or *Eliane elias - Segredos Secrets*. The problem we face is that, by giving a higher weight to elements in the head set, we will not obtain a set of tags that can successfully characterize each track in order to enable a suitable distinction from the remaining tags. While this can be done more successfully by using the tags in the tail set, most of them are so unique that they will prevent a successful clustering with similar tags, as there are no other instances of those tags. That is why we decided to take advantage of an established metric, the TF-IDF, that tackles this problem with a balanced approach, where both uniqueness over the whole collection and popularity are combined to provide an estimate of the appropriateness of a tag to characterize a given track, distinguishing the track from disparate tracks or clustering it with other similar tracks.

Our goal is to explore how the two parameters α and β of each of the previously defined algorithms influence the results, as well as to understand the differences between boosting results in the long tail using the IDF, versus singularly using the TF. In order to elaborate on this, we prepared an experiment where we generated twenty input playlists with ten tracks each, retrieved using a textual search process on top of an index with the fields title, release name, artist name, tags and lyrics, for the manually defined queries illustrated in Table 2. Each query aims at representing a different kind of taste and, based on each resulting playlist, we obtain a new playlist of recommended tracks. The playlist generation task is usually concerned with not only providing the best track recommendations, but also reordering the tracks by coherence to ensure a smooth transition, however we are not concerned at this point with the reordering task and thus disregard this factor in our experiments.

Using the TF and the TF-IDF as alternative metrics to rank the top tags and top tracks, we prepared seven runs for each of the input playlists, where we tested different values for the parameters, in order to understand how the results vary as those values are increased or decreased individually. The parameter combinations used in each run are shown in Table 3 in the α/β column. In total, we obtained 280 distinct playlists (2 *metrics* $\times$ 7 *runs* $\times$ 20 *input playlists*) with 30 recommended tracks each.

Table 2: Queries for the generation of the input playlists.

Playlist	Query
1	coldplay live
2	metallica slayer heavy metal
3	nirvana days of the new grunge alice in chains
4	jason mraz i'm yours
5	happy good vibe
6	sad depressing doom dark
7	britney spears rihanna madonna
8	norah jones diana krall jamie cullum
9	miles davis john coltrane classic jazz
10	frank sinatra new york
11	bob marley reggae summer happy positive
12	pop rock avril lavigne
13	indiana jones soundtrack
14	led zeppelin the who classic rock
15	rockabilly 50s elvis presley
16	country bluegrass bill monroe banjo
17	dubstep skrillex new beat
18	electronic aphex twin creative
19	house techno trance bestof
20	blues muddy waters robert johnson jimi hendrix

6 Evaluation

We did a comparison between corresponding resulting playlists in the first and second algorithms by calculating the intersection size for the pairs of recommended playlists. Figure 3 illustrates how different α/β parameter combinations influence the outcome of the algorithms, by analyzing the number of common tracks in the playlists generated with the TF and the TF-IDF metrics. As we can see, we consistently obtain the most different results for $\alpha = 0.2$ and $\beta = 0.5$. As expected, when the fraction of considered tags and tracks approximates 1, the algorithms tend to similar results. This happens due to the fact that information is being filtered by extracting a fraction of the top ranked items and as this fraction gets closer to 100%, the number of common items increases, thus reducing the difference between using the TF versus the TF-IDF.

We further analyzed the resulting playlists and assessed the discovery impact of the tag-based recommendations using an extension of the evaluation method described by Coelho et al. [3]. Given the lack of ground truth that would normally be used to evaluate the results using precision and recall,

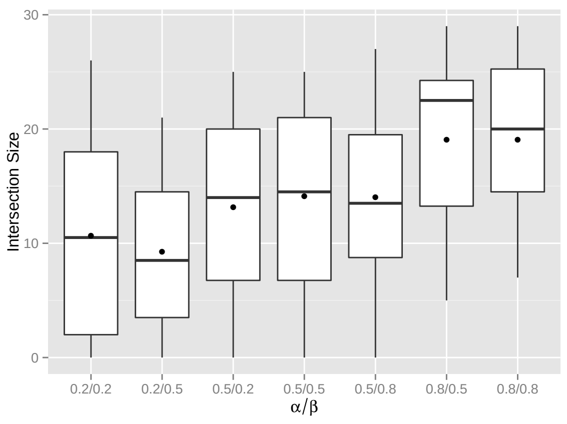

Figure 3: Box plot illustrating the number of common tracks in the corresponding recommended playlists that resulted from each method. The sequence of boxes illustrates how the α and β parameters influence the results. The mean value of the intersection size for each of the twenty playlists is shown as a point.

Table 3: Playlist average evaluation score for the 280 generated playlist recommendations.

α/β	TF method score		TF-IDF method score	
	Average	Std. Dev.	Average	Std. Dev.
0.2/0.2	0.00204	0.00295	0.00100	0.00182
0.2/0.5	0.00171	0.00329	0.00100	0.00248
0.5/0.2	0.00184	0.00240	0.00085	0.00172
0.5/0.5	0.00158	0.00224	0.00100	0.00224
0.5/0.8	0.00174	0.00223	0.00111	0.00322
0.8/0.5	0.00176	0.00253	0.00114	0.00197
0.8/0.8	0.00171	0.00235	0.00111	0.00211

the authors considered a pair of tracks in a recommended playlist as a true positive whenever it also appeared in a user's taste profile. In this paper, we extend the binary approach used by Coelho at al., where each pair of tracks would contribute only once to the evaluation score even if it appeared on more than one taste profile, to a frequentist approach, where a pair of tracks contributes as much to the evaluation score as the number of times it can be counted across individual taste profiles.

Thus, we defined our evaluation score as the sum over all pairs of songs, for the total number of users that listened to each individual pair of songs. This value can be normalized with the division by the maximum possible score, given by $\frac{1}{2} \times p \times (p-1) \times u$, where p is the number of tracks representing the playlist and u is the number of users. Additionally, we improved on this evaluation method by selecting the subset of tracks with a play count superior to the overall average play count (2.863) from the users' taste profiles. This has two consequences: (*i*) the removal of tracks with the lowest play count, solely focusing on tracks that the users potentially liked, in the sense that they listened to them more times than average, and (*ii*) a significant decrease of the evaluation time complexity.

The results of the assessment are shown in Table 3. As we can see, the recommendations provided by the second algorithm achieved a lower overall score, with similar, while slightly lower, standard deviation values, when compared to the first algorithm. Based on the pairwise frequency of the tracks from the recommended playlists in the users music libraries, there is a clear indication that a smaller number of users have listened to each pair of tracks that resulted from the TF-IDF based method. We could say that the discovery impact of the recommended playlist is potentially higher when using the IDF to boost the TF of the tags. This fact is congruent with our music discovery goal, where less popular tracks should have a higher probability of being included in the recommended playlist. While the evaluation of discovery cannot solely rely on popularity, we were still able to obtain playlists with less popular and potentially more surprising choices, whose generation process was supported by a solid knowledge base that modeled the relationships between tracks based on common tags.

7 Summary and Conclusions

We have compared two different methodologies based on graph traversal operations using the Gremlin language and a graph database backend to support our recommendation system. We tested each algorithm using a set of input playlists generated through textual search, obtaining several recommended playlists for representative pairs of parameters. We not only concluded that the IDF does in fact influence the outcome of the system in a positive way, regarding the discovery impact, but also that using a lower fraction of top tags and top tracks resulted in a stronger evidence of the differences introduced by using the TF-IDF versus the TF.

7.1 Future Work

As future work, we would like to include additional features, such as content-based features or genre, to improve the music discovery process. We would also like to work on the evaluation methodologies, considering factors such as popularity and discovery impact, but, more importantly, obtaining human feedback regarding the recommended playlists, to support our approach and its difference from randomly selecting the tracks for a playlist.

Acknowledgments

This work has been financed by Laboratório SAPO/U.Porto under a research grant for the Juggle project.

References

[1] Thierry Bertin-Mahieux, Daniel P W Ellis, Brian Whitman, and Paul Lemere. The Million Song Dataset. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011)*, pages 591–596, Miami, Florida, 2011.

[2] Òscar Celma Herrada. *Music recommendation and discovery in the long tail.* Phd thesis, Universitat Pompeu Fabra, 2009.

[3] Filipe Coelho, José Devezas, and Cristina Ribeiro. Large-scale Crossmedia Retrieval for Playlist Generation and Song Discovery. In *Proceedings of the 10th International Conference in the RIAO series (OAIR 2013)*, 2013.

[4] Benjamin Fields. *Contextualize your listening: The playlist as recommendation engine.* Phd thesis, Goldsmiths, University of London, 2011.

[5] CS Firan, Wolfgang Nejdl, and R Paiu. The benefit of using tag-based profiles. In *Web Conference, 2007. LA-WEB 2007. Latin American*, pages 32–41, 2007.

[6] Jeroen De Knijf, Anthony Liekens, and Bart Goethals. GaMuSo: graph base music recommendation in a social bookmarking service. In *Advances in Intelligent Data Analysis X*, 2011.

[7] Brian McFee, Thierry Bertin-Mahieux, Daniel PW Ellis, and Gert RG Lanckriet. The million song dataset challenge. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 909–916. ACM, 2012.

[8] Yoon-Joo Park and Alexander Tuzhilin. The long tail of recommender systems and how to leverage it. *Proceedings of the 2008 ACM conference on Recommender systems - RecSys '08*, page 11, 2008.

[9] Alexander Schindler, Rudolf Mayer, and Andreas Rauber. Facilitating comprehensive benchmarking experiments on the million song dataset. In *ISMIR*, pages 469–474, 2012.

[10] S Sen, Jesse Vig, and John Riedl. Learning to recognize valuable tags. In *Proceedings of the 14th international conference on Intelligent user interfaces (IUI 2009)*, pages 87–96, 2009.

[11] S Sen, Jesse Vig, and John Riedl. Tagommenders: connecting users to items through tags. In *Proceedings of the 18th international conference on World wide web*, pages 671–680, 2009.