LEARNING TO EMBED SONGS AND TAGS FOR PLAYLIST PREDICTION

Joshua L. Moore, Shuo Chen, Thorsten Joachims
Cornell University, Dept. of Computer Science
{jlmo|shuochen|tj}@cs.cornell.edu

Douglas Turnbull
Ithaca College, Dept. of Computer Science
dturnbull@ithaca.edu

ABSTRACT

Automatically generated playlists have become an important medium for accessing and exploring large collections of music. In this paper, we present a probabilistic model for generating coherent playlists by embedding songs and social tags in a unified metric space. We show how the embedding can be learned from example playlists, providing the metric space with a probabilistic meaning for song/song, song/tag, and tag/tag distances. This enables at least three types of inference. First, our models can generate new playlists, outperforming conventional n-gram models in terms of predictive likelihood by orders of magnitude. Second, the learned tag embeddings provide a generalizing representation for embedding new songs, allowing it to create playlists even for songs it has never observed in training. Third, we show that the embedding space provides an effective metric for matching songs to natural-language queries, even if tags for a large fraction of the songs are missing.

1. INTRODUCTION

Music consumers can store thousands of songs on their computer or smart phone. In addition, cloud-based services like Rhapsody or Spotify give instant and on-demand access to millions of songs. While these technologies provide powerful new ways to access music, they can also overwhelm users by giving them too much choice [15].

This has created substantial interest in automatic playlist algorithms that can help consumers explore large collections of music. Companies like Apple and Pandora have developed successful commercial playlist algorithms, but relatively little is known about how these algorithms work and how well they perform in rigorous evaluations. Comparably little scholarly work has been done on automated methods for playlist generation (e.g., [1, 4, 10, 12, 14]), and the results to date indicate that it is far from trivial to operationally define what makes a playlist coherent.

Most approaches to automatic playlist creation rely on computing some notion of music similarity between pairs of songs. Numerous similarity functions have been proposed and are often based on the analysis of audio content [9, 13], social tag information [8], web document mining [6], preference-based ratings data [11], or some combination of these data sources. Given a music similarity algorithm, a playlist is created by finding the most similar songs to a given seed song or set of seed songs.

In this paper, we explore the idea of learning a playlist model that does not require an external similarity measure and that is trained directly on the data of interest, namely historical playlists. In particular, we extend the *Logistic Markov Embedding* (LME) [3] approach to probabilistic sequence modeling to incorporate social tags, unifying song and tag embeddings in a single Euclidean space. This provides a probabilistically well-founded and constructive way to compute meaningful distances between pairs of songs, pairs of tags, and songs and tags. We show that this joint embedding is useful not only for probabilistically sound playlist generation, but also for a variety of other music information retrieval tasks such as corpus visualization, automatic tagging, and keyword-based music retrieval.

An efficient C implementation, a demo, and data are available at `http://lme.joachims.org`.

2. RELATED WORK

Automatically generated playlists are a key component in several commercial systems. For example, Pandora relies on content-based music analysis by human experts [16] while Apple iTunes Genius relies on preference ratings and collaborative filtering [2]. What is not known is the mechanism by which the playlist algorithms are used to *order* the set of relevant songs, nor is it known how well these playlist algorithms perform in rigorous evaluations.

In the scholarly literature, two recent papers address the topic of playlist prediction. First, Maillet et al. [10] formulate the playlist ordering problem as a supervised binary classification problem that is trained discriminatively. Positive examples are pairs of songs that appeared in this order in the training playlists, and negative examples are pairs of songs selected at random which do not appear together in order in historical data. Second, McFee and Lanckriet [12] take a generative approach by modeling historical playlists as a Markov chain. That is, the probability of the next song in a playlist is determined only by acoustic and/or social-tag similarly to the current song. Our approach is substantially different from both [10] and [12], since we do not require any acoustic or semantic information about the songs.

While relatively little work has been done on explicitly modeling playlists, considerably more research has focused on embedding songs (or artists) into a similarity-based music space (e.g., [4, 9, 14, 18].) For example, Platt et al. use semantic tags to learn a Gaussian process kernel function between pairs of songs [14]. More recently, Weston et al. learn an embedding over a joint semantic space of audio features, tags and artists by optimizing performance metrics for various music retrieval tasks [18]. Our approach, however, differs substantially from these existing methods, since it explicitly models the sequential nature of playlists in the embedding. Recently and independently, [1] also proposed a sequential embedding model. However, their model does not include tags.

Modeling playlists as a Markov chain connects to a large body of work on sequence modeling in natural language processing (NLP) and speech recognition. Smoothed n-gram models (see e.g. [5]) are the most commonly used method in language modeling, and we will compare against such models in our experiments.

3. PROBABILISTIC EMBEDDING OF PLAYLISTS

Our goal is to estimate a generative model of coherent playlists, which will enable us to efficiently sample new playlists. More formally, given a collection $\mathcal{S} = \{s_1, ..., s_{|\mathcal{S}|}\}$ of songs s_i, we would like to estimate the distribution $\Pr(p)$ of coherent playlists $p = (p^{[1]}, ..., p^{[k_p]})$. Each element $p^{[i]}$ of a playlist refers to one song from S.

A natural approach is to model playlists as a Markov chain, where the probability of a playlist $p = (p^{[1]}, ..., p^{[k_p]})$ is decomposed into the product of transition probabilities $\Pr(p^{[i]}|p^{[i-1]})$ between adjacent songs $p^{[i-1]}$ and $p^{[i]}$.

$$\Pr(p) = \prod_{i=1}^{k_p} \Pr(p^{[i]}|p^{[i-1]}) \tag{1}$$

For ease of notation, we assume that $p^{[0]}$ is a dedicated start symbol. Such bi-gram (or, more generally, n-gram) models have been widely used in language modeling for speech recognition and machine translation with great success [5]. In these applications, the $O(|\mathcal{S}|^n)$ transition probabilities $\Pr(p^{[i]}|p^{[i-1]})$ are estimated from a large corpus of text using sophisticated smoothing methods.

While such n-gram approaches can be applied to playlist prediction in principle, there are fundamental differences between playlists and language. First, playlists are less constrained than language, so that transition probabilities between songs are closer to uniform. This means that we need a substantially larger training corpus to observe all of the (relatively) high-probability transitions even once. Second, and in contrast to this, we have orders of magnitude less playlist data to train from than we have written text.

To overcome these problems, we propose a Markov-chain sequence model that produces a *generalizing representation* of songs, song sequences, and social tags. Unlike n-gram models that treat words as atomic units without metric relationships between each other, our approach seeks to model coherent playlists as paths through a latent space. In particular, songs are embedded as points in this space so that Euclidean distance between songs reflects the transition probabilities. Similarly, each social tag is represented as a point in this space, summarizing the average location of songs with that tag. The key learning problem is to determine the location of each song and tag using existing playlists as training data. Once songs and tags are embedded, our model can assign meaningful transition probabilities even to those transitions that were not seen in the training data, and it can also reason about tagged songs that were never seen before.

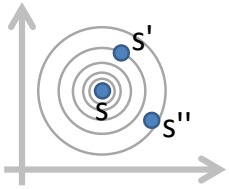
In the following we start by reviewing the basic LME model of $\Pr(p)$, and then extend this model to incorporate social tags.

3.1 Embedding Model for Songs

The basic LME model [3] represents each song s as a single vector $X(s)$ in d-dimensional Euclidean space $\mathcal{M}$. The key assumption of our model is that the transition probabilities $\Pr(p^{[i]}|p^{[i-1]})$ are related to the Euclidean distance $||X(p^{[i]}) - X(p^{[i-1]})||_2$ between $p^{[i-1]}$ and $p^{[i]}$ in $\mathcal{M}$ through the following logistic model:

$$\Pr(p^{[i]}|p^{[i-1]}) = \frac{e^{-||X(p^{[i]})-X(p^{[i-1]})||_2^2}}{\sum_{j=1}^{|S|} e^{-||X(s_j)-X(p^{[i-1]})||_2^2}} \tag{2}$$

This is illustrated in the figure to the right, showing that transitioning from s to a nearby point s' is more likely than transitioning to a point s'' that is further away. We will typically abbreviate the partition function in the denominator as $Z(p^{[i-1]})$, and the distance $||X(s) - X(s')||_2$ between two songs in embedding space as $\Delta(s, s')$ for brevity. Using a Markov model with this transition distribution, we can now define the probability of an entire playlist of a given length k as



$$\Pr(p) = \prod_{i=1}^{k_p} \Pr(p^{[i]}|p^{[i-1]}) = \prod_{i=1}^{k_p} \frac{e^{-\Delta(p^{[i]},p^{[i-1]})^2}}{Z(p^{[i-1]})}. \tag{3}$$

The LME seeks to discover an embedding of the songs into this latent space which causes "good" playlists to have high probability of being generated by this process. This is inspired by collaborative filtering methods such as [7], which similarly embed users and items into a latent space to predict users' ratings of items. However, our approach differs from these methods in that we wish to predict paths through the space, as opposed to independent item ratings.

In order to learn the embedding of songs, we use a sample $D = (p_1, ..., p_n)$ of existing playlists as training data and take a maximum a posteriori (MAP) approach to learning. Denoting with X the matrix of embedding vectors for all songs in the collection $\mathcal{S}$, this leads to the following training problem

$$X = \underset{X \in \Re^{|\mathcal{S}| \times d}}{\operatorname{argmax}} \prod_{p \in D} \prod_{i=1}^{k_p} \frac{e^{-\Delta(p^{[i]},p^{[i-1]})^2}}{Z(p^{[i-1]})} \cdot \prod_{i=1}^{|S|} e^{-\lambda||X(s_i)||_2^2}, \tag{4}$$

where we also added a zero-mean Normal prior as regularizer to control overfitting (see term after the dot). The parameter λ controls how heavily the embedding is regularized. While the optimization problem is not concave, we have already shown in [3] how to efficiently and robustly find good optima using a stochastic gradient approach.

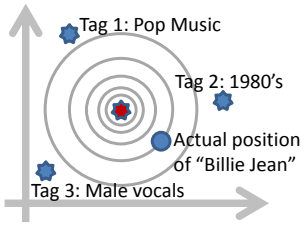
3.2 Embedding Model for Songs and Tags

The previous model is very general in that it does not require any features that describe songs. However, this is also a shortcoming, since it may ignore available information. We therefore now extend the LME to include tags as prior information. The new model will provide reasonable embeddings even for songs it was not trained on, and it will define a unified metric space for music retrieval based on query tags.

The key idea behind the new model is that the tags $T(s)$ of song s inform the prior distribution of its embedding location $X(s)$. In particular, each tag t is associated with a Normal distribution $\mathcal{N}(M(t), \frac{1}{2\lambda} I_d)$ with mean $M(t)$. Here, I_d is the d by d identity matrix and we will see soon that λ again behaves like a regularization parameter. For a song with multiple tags, we model the prior distribution of its embedding as the average of the Normal distribution of its tags $T(s)$, while keeping the variance constant.

$$\Pr(X(s)|T(s)) = \mathcal{N}\left(\frac{1}{|T(s)|}\sum_{t\in T(s)} M(t), \frac{1}{2\lambda} I_d\right) \quad (5)$$

Note that this definition of $\Pr(X(s)|T(s))$ nicely generalizes the regularizer in (4), which corresponds to an "uninformed" Normal prior $\Pr(X(s)) = \mathcal{N}(0, \frac{1}{2\lambda} I_d)$ centered at the origin of the embedding space. The tag-based prior distribution is illustrated in the figure to the right. In this example, the song "Billie Jean" has the three tags "pop music", "male vocals" and "1980s". Each tag has a mean $M(t)$ as depicted, and $\Pr(X(s)|T(s))$ is centered at the average of the tag means, providing the prior for the embedding of "Billie Jean". Without any training data, the most likely location is the center of the prior, but with more observed training data the embedding may move further away as necessary.



Let M be the matrix of all tag means, we obtain the following maximum a posteriori estimate for the tag-based LME analogous to the basic LME model:

$$(X, M) = \underset{X,M}{\operatorname{argmax}} \Pr(D|X) \cdot \Pr(X|M) \quad (6)$$

$$= \underset{X,M}{\operatorname{argmax}} \prod_{p\in D} \prod_{i=1}^{k_p} \frac{e^{-\Delta(p^{[i]}, p^{[i-1]})^2}}{Z(p^{[i-1]})} \cdot \prod_{i=1}^{|S|} e^{-\lambda||X(s) - \frac{\sum_{t\in T(s)} M(t)}{|T(s)|}||_2^2}$$

Note that we now optimize jointly over the song locations $X(s)$ and tag locations $M(t)$. In this way, the tag-based LME model yields a meaningful probabilistic interpretation of distances not only among songs, but also among songs and tags. The following experiments exploit this for locating new songs and for tag-based music retrieval.

	yes_small	*yes_big*
Appearance Threshold	20	5
Num of Songs	3,168	9,775
Num of Train Trans	134,431	172,510
Num of Test Trans	1,191,279	1,602,079

Table 1: Statistics of the playlists datasets.

4. EXPERIMENTS

The playlists and tag data we used for our experiments are respectively crawled from *Yes.com* and *Last.fm*.

Yes.com is a website that provides radio playlists from hundreds of radio stations in the United States. By using the web based API [1], one can retrieve the playlist record of a specified station for the last 7 days. We collected as many playlists as possible by specifying all possible genres and getting playlists from all possible stations. The collection lasted from December 2010 to May 2011. This lead to a dataset of 75,262 songs and 2,840,553 transitions. To get datasets of various sizes, we pruned the raw data so that only the songs with a number of appearances above a certain threshold are kept. We then divide the pruned set into a training set and a testing set, making sure that each song has appeared at least once in the training set. We report results for two datasets, namely *yes_small* and *yes_big*, whose basic statistics are shown in Table 1.

Last.fm provides tag information for songs, artists and albums that is contributed by its millions of users. For each of the songs in our playlists dataset, we query the Last.fm API [2] for the name of the artist and the song, retrieving the top tags. We then prune the tag set by only keeping the top 250 tags with the most appearances across songs. Note that Last.fm did not provide any tags for about 20% of songs.

Unless noted otherwise, experiments use the following setup. Any model (either the LME or the baseline model) is first trained on the training set and then tested on the test set. We evaluate test performance using average log-likelihood as our metric. It is defined as $\log(\Pr(D_{\text{test}}))/N_{\text{test}}$, where N_{test} is the number of transitions in test set.

4.1 What does the embedding space look like?

Before starting the quantitative evaluation of our method, we first want to give a qualitative impression of the embeddings it produces. Figure 1 shows the two-dimensional embedding of songs and tags according to (6) for the *yes_small* dataset. The top 50 genre tags are labeled, and the lighter points represent songs.

Overall, the embedding matches our intuition of what a semantic music space should look like. The most salient

[1] http://api.yes.com
[2] http://www.last.fm/api

Figure 1: 2D embedding for *yes_small*. The top 50 genre tags are labeled; lighter points represent songs.

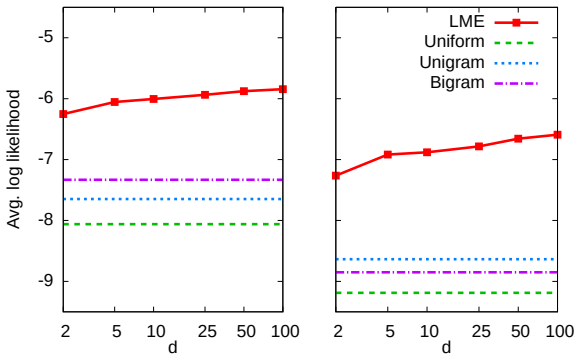


Figure 2: Log-likelihood on the test set for the LME and the baselines on *yes_small* (left) and *yes_big* (right).

observation is that the embedding of songs does not uniformly cover the space, but forms clusters as expected. The location of the tags provides interesting insight into the semantics of these clusters. Note that semantically synonymous tags are typically close in embedding space (e.g. "christian rock" and "christian", "metal rock" and "heavy metal"). Furthermore, location in embedding space generally interpolates smoothly between related genres (e.g. "rock" and "metal"). Note that some tags lie outside the support of the song distribution. The reason for this is twofold. First, we will see below that a higher-dimensional embedding is necessary to accurately represent the data. Second, many tags are rarely used in isolation, so that some tags may often simply modify the average prior for songs.

To evaluate our method and the embeddings it produces more objectively and in higher dimensions, we now turn to quantitative experiments.

4.2 How does the LME compare to n-gram models?

Our first quantitive experiment explores how the generalization accuracy of the LME compares to that of traditional n-gram models from natural language processing (NLP). The simplest NLP model is the **Unigram Model**, where

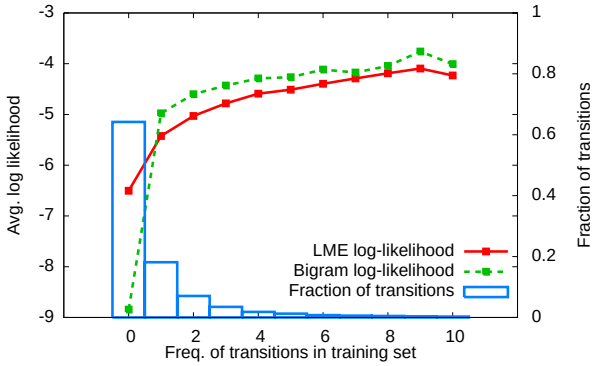


Figure 3: Log-likelihood on testing transitions with respect to their frequencies in the training set for *yes_small*.

the next song is sampled independently of the previous songs. The probability $p(s_i)$ of each song s_i is estimated from the training set as $p(s_i) = \frac{n_i}{\sum_j n_j}$, where n_i is the number of appearances of s_i.

The **Bigram Model** conditions the probability of the next song on the previous song similar to our LME model. However, the transition probabilities $p(s_j|s_i)$ of each song pair are estimated separately, not in a generalizing model as in the LME. To address the the issue of data sparsity when estimating $p(s_j|s_i)$, we use Witten-Bell smoothing (see [5]) as commonly done in language modeling.

As a reference, we also report the results for the **Uniform Model**, where each song has equal probability $1/|\mathcal{S}|$.

Figure 2 compares the log-likelihood on the test set of the basic LME model to that of the baselines. The x-axis shows the dimensionality d of the embedding space. For the sake of simplicity and brevity, we only report the results for the model from Section 3.1 trained without regularization (i.e. $\lambda = 0$). Over the full range of d the LME outperforms the baselines by at least two orders of magnitude in terms of likelihood. While the likelihoods on the big dataset are lower as expected (i.e. there are more songs to choose from), the relative gain of the LME over the baselines is even larger for *yes_big*.

The tag-based model from Section 3.2 performs comparably to the results in Figure 2. For datasets with less training data per song, however, we find that the tag-based model is preferable. We explore the most extreme case, namely songs without any training data, in Section 4.4.

Among the conventional sequence models, the bigram model performs best on *yes_small*. However, it fails to beat the unigram model on *yes_big* (which contains roughly 3 times the number of songs), since it cannot reliably estimate the huge number of parameters it entails. Note that the number of parameters in the bigram model scales quadratically with the number of songs, while it scales only linearly in the LME model. The following section analyzes in more detail where the conventional bigram model fails, while the LME shows no signs of overfitting.

4.3 Where does the LME win over the n-gram model?

We now analyze why the LME beats the conventional bigram model. In particular, we explore to what extent

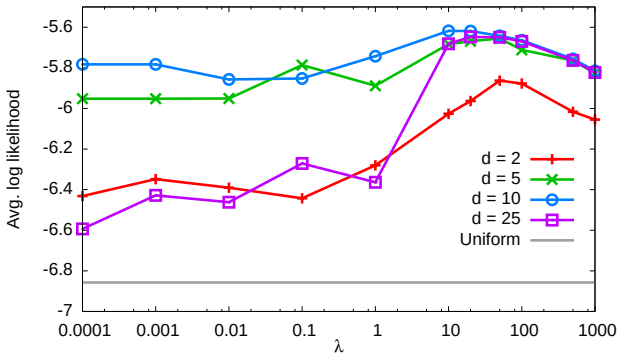


Figure 4: Log-likelihood of predicting transitions for new songs for different d and λ.

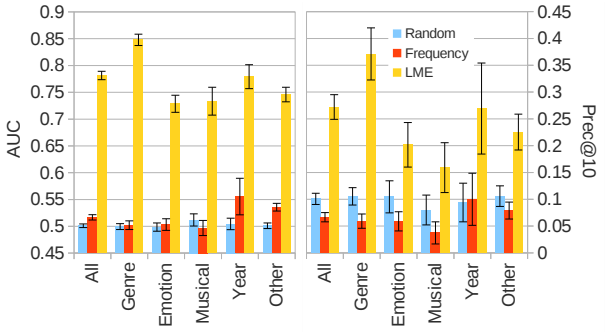


Figure 5: Average AUC (left) and precision at 10 (right) across tag categories for random and frequency baselines and LME. Error bars indicate +/- 1 standard error.

the generalization performance of the methods depends on whether (and how often) a test transition was observed in the training set. The ability to produce reasonable probability estimates even for transitions that were never observed is important, since even in *yes_small* about 64% of test transitions were not at all observed in our training set.

For both the LME and the bigram model, the lines in Figure 3 show the log-likelihood of the test transitions conditioned on how often that transition was observed in the training set of *yes_small*. The bar graph illustrates what percentage of test transitions had that given number of occurences in the training set (i.e. 64% for zero). It can be seen that the LME performs comparably to the bigram model for transitions that were seen in the training set at least once, but it performs substantially better on previously unseen transitions. This is a key advantage of the generalizing representation that the LME provides, since it provides an informed way of assigning transition probabilities to all pairs of songs.

4.4 Can the tag model coldstart new songs?

Any playlist generator will encounter new songs it has not been trained on. Fortunately, it is easy to impute an embedding for new songs in our tag-based LME model. Given a new song s with tags $T(s)$, the most likely embedding location according our probabilistic model is

$$X(s) = \frac{1}{|T(s)|} \sum_{t \in T(s)} M(t). \quad (7)$$

To evaluate performance on new songs, we take the *yes_small* dataset and randomly withhold a subset of 30% (951) of the songs which have at least one tag each. We test on these songs and train the tag-based LME on the remaining songs. In particular, we test on transitions from training to test songs, having our model predict based on the imputed test-song location which one of the 951 songs was played.

The only valid baseline for this experiment is the uniform model, since we have no history for the testing songs. The results are shown in Figure 4 for various dimensionalities and regularization parameters λ. Over all parameter settings, the LME outperforms the baseline substantially. Comparing Figure 4 with Figure 2, the gain over uniform for new songs is still roughly half of that for songs that the LME was trained on. This demonstrates that the embedding of the tags captures a substantial amount of the playlist semantics, generalizing well even for new songs.

4.5 Can the embedding space be used for retrieval?

As already demonstrated in the previous section, a powerful property of our model is that it results in a similarity metric that unifies tags and songs – namely, the Euclidean distance of the corresponding points in the embedding. This leads to a natural method for retrieval of songs based on query tags: rank songs by their Euclidian distance to the query tag(s). Note that this method can retrieve songs even though they are not manually tagged with any of the query tags.

To evaluate the effectiveness of the embedding space for retrieval, we now evaluate how well **untagged** songs can be retrieved using queries that consist of a single tag. The experiment is set up as follows. We pooled the train and test partitions of the *yes_small* dataset and then randomly split all songs with at least one tag into 5 partitions. Following a 5-fold cross-validation setup, we removed the tags from the songs in one of the partitions, trained the tag-based LME on the now untagged songs plus the tagged songs from the other 4 partitions, and then computed the query-tag rankings over the untagged songs. For each query tag, we computed the average (over folds) ROC Area (AUC) and Precision@10.

Figure 5 shows the results for the LME and for two baselines: a random ranking of all held-out songs and a ranking of the held-out songs in order of decreasing frequency of appearance in the data set. We separated (by hand) each of the 250 query tags into one of five categories: genre tags (91 tags like rock, hip hop, etc.), emotion tags (35 tags: sad, happy, dark, upbeat etc.), musical and instrumental tags (23 tags: male vocalist, guitar, major key tonality...), years and decades (17 tags), and other tags (84 tags including awesome, loved, catchy, and favorites). For brevity, we only report results for a model with dimension 25 and $\lambda = 10$. However, similar to the results in Figure 4, we find that the exact choice of these parameters is not crucial. For example, the best unregularized model was never more than 4 percentage points worse in AUC than the best regularized model (though naturally for higher dimensions

regularization becomes more important).

Our method significantly and substantially outperforms both baselines in every category. Matching our intuition, it does the best for genre queries, with an AUC of nearly 0.85 and Precision@10 of about 37%. The emotion and musical categories prove the most difficult, while the year and other categories are the easiest after genre.

Note that the performance values reported in Figure 5 are extremely conservative estimates of the actual retrieval quality of our method. This is for three reasons: First, social tags can be noisy since they result from ad-hoc labeling practices by non-experts [17]. Second, we made no attempt to identify lexicographically similar tags as the same. For example, consider the following ranking that our method produces for the tag-query "male vocals", with a relevant subset of the tags given for each song:

Daughtry - Home: male vocalists, male vocalist, male
Allen - Live Like We're Dying: male vocalists, male vocalist
The Fray - How To Save A Life: male vocalists, male vocalist
Aerosmith - Rag Doll: male vocalist, malesinger
Lifehouse - Hanging By A Moment: male vocalists, male vocalist, male vocals

Here, all five songs are clearly relevant to the query, but only the last song was considered relevant for the purposes of our experiments. Third, we only test our method on songs for which *no tags at all* were seen during training. For these reasons, it is important to keep in mind that the results we report are strict lower bounds on the actual retrieval performance of our method.

5. CONCLUSIONS

We presented a method for learning to predict playlists through an embedding of songs and tags in Euclidian space. The method not only provides a well-founded probabilistic model for playlist generation, it also produces a distance metric with a probabilistic meaning for song/song, song/tag, and tag/tag distances. We show that the method substantially outperforms conventional sequence models from NLP, that it can sensibly impute the location of previously unseen songs, and that its distance metric is effective for music retrieval even of untagged songs.

The flexibility of the LME approach provides exciting opportunities for future work, since the model leaves open the possibility of more complex representations of songs. For example, instead of representing each song as a single $X(s)$, one can use two embedding vectors $U(s)$ and $V(s)$ to model the beginning and ending of a song respectively. This allows modeling that the ending of song s is compatible with the beginning of song s', but that the reverse may not be the case. Another interesting direction for future work is the modeling of long-range dependencies in playlists. Such long-range dependencies could capture the amount of redundancy/repetition that a user may seek, versus how much a playlist provides variety and explores new music.

This research was supported in part by NSF Awards IIS-1217686, IIS-0812091 and IIS-0905467.

6. REFERENCES

[1] N. Aizenberg, Y. Koren, and O. Somekh. Build your own music recommender by modeling internet radio streams. In *WWW*, 2012.

[2] L. Barrington, R. Oda, and G. Lanckriet. Smarter than genius? human evaluation of music recommender systems. *ISMIR*, 2009.

[3] Shuo Chen, J. L. Moore, D. Turnbull, and T. Joachims. Playlist prediction via metric embedding. In *SIGKDD*, 2012.

[4] D. F. Gleich, L. Zhukov, M. Rasmussen, and K. Lang. The World of Music: SDP embedding of high dimensional data. In *Information Visualization 2005*, 2005.

[5] D. Jurafsky and J.H. Martin. Speech and language processing, 2008.

[6] P. Knees, T. Pohle, M. Schedl, D. Schnitzer, and K. Seyerlehner. A document-centered approach to a natural language music search engine. In *ECIR*, 2008.

[7] Y. Koren, R. M. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[8] M. Levy and M. Sandler. A semantic space for music derived from social tags. In *ISMIR*, 2007.

[9] B. Logan. Content-based playlist generation: exploratory ex- periments. *ISMIR*, 2002.

[10] F. Maillet, D. Eck, G. Desjardins, and P. Lamere. Steerable playlist generation by learning song similarity from radio station playlists. In *ISMIR*, 2009.

[11] B. McFee, L. Barrington, and G. Lanckriet. Learning content similarity for music recommendation. *IEEE TASLP*, 2012.

[12] B. McFee and G. R. G. Lanckriet. The natural language of playlists. In *ISMIR*, 2011.

[13] E. Pampalk. *Computational Models of Music Similarity and their Application in Music Information Retrieval*. PhD thesis, TU Wien, Vienna, Austria, 2006.

[14] J. C. Platt. Fast embedding of sparse music similarity graphs. In *NIPS*, 2003.

[15] B. Schwartz. *The Paradox of Choice: Why More is Less*. Ecco, 2003.

[16] D. Tingle, Y. Kim, and D. Turnbull. Exploring automatic music annotation with "acoustically-objective" tags. In *ICMR*, 2010.

[17] D. Turnbull, L. Barrington, and G. Lanckriet. Five approaches to collecting tags for music. In *ISMIR*, 2008.

[18] J. Weston, S. Bengio, and P. Hamel. Multi-tasking with joint semantic spaces for large-scale music annotation and retrieval. *Journal of New Music Research*, 2011.