Playlist Prediction via Metric Embedding

Shuo Chen
Cornell University
Dept. of Computer Science
Ithaca, NY, USA
shuochen@cs.cornell.edu

Joshua L. Moore
Cornell University
Dept. of Computer Science
Ithaca, NY, USA
jlmo@cs.cornell.edu

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

Thorsten Joachims
Cornell University
Dept. of Computer Science
Ithaca, NY, USA
tj@cs.cornell.edu

ABSTRACT

Digital storage of personal music collections and cloud-based music services (e.g. Pandora, Spotify) have fundamentally changed how music is consumed. In particular, automatically generated playlists have become an important mode of accessing large music collections. The key goal of automated playlist generation is to provide the user with a *coherent* listening experience. In this paper, we present Latent Markov Embedding (LME), a machine learning algorithm for generating such playlists. In analogy to matrix factorization methods for collaborative filtering, the algorithm does not require songs to be described by features a priori, but it learns a representation from example playlists. We formulate this problem as a regularized maximum-likelihood embedding of Markov chains in Euclidian space, and show how the resulting optimization problem can be solved efficiently. An empirical evaluation shows that the LME is substantially more accurate than adaptations of smoothed n-gram models commonly used in natural language processing.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.5.1 [**Pattern Recognition**]: Models

General Terms

Algorithms, Experimentation, Human Factors

Keywords

Music Playlists, Recommendation, User Modeling, Sequences

1. INTRODUCTION

A music consumer can store thousands of songs on his or her computer, portable music player, or smart phone. In addition, when using a cloud-based service like Rhapsody or Spotify, the consumer has instant on-demand access to millions of songs. 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.

Despite the large commercial demand, comparably little scholarly work has been done on automated methods for playlist generation (e.g., [13, 4, 9, 11]), and the results to date indicate that it is far from trivial to operationally define what makes a playlist coherent. The most comprehensive study was done by [11]. Working under a model where a coherent playlist is defined by a Markov chain with transition probabilities reflecting similarity of songs, they find that neither audio-signal similarity nor social-tag-based similarity naturally reflect manually constructed playlists.

In this paper, we therefore take an approach to playlist prediction that does not rely on content-based features, and that is analogous to matrix decomposition methods in collaborative filtering [7]. Playlists are treated as Markov chains in some latent space, and our algorithm – called Logistic Markov Embedding (LME) – learns to represent each song as one (or multiple) points in this space. Training data for the algorithm consists of existing playlists, which are widely available on the web. Unlike other collaborative filtering approaches to music recommendation like [13, 4, 19], ours is among the first (also see [1]) to directly model the sequential and directed nature of playlists, and that includes the ability to sample playlists in a well-founded and efficient way.

In empirical evaluations, the LME algorithm substantially outperforms traditional n-gram sequence modeling methods from natural language processing. Unlike such methods, the LME algorithm does not treat sequence elements as atomic units without metric properties, but instead provides a generalizing representation of songs in Euclidean space. Technically, it can be viewed as a multi-dimensional scaling problem [3], where the algorithm infers the metric from a stochastic sequence model. While we focus exclusively on playlist prediction in this paper, the LME algorithm also provides interesting opportunities for other sequence prediction problems (e.g. language modeling).

2. RELATED WORK

Personalized Internet radio has become a popular way of listening to music. A user seeds a new stream of music by specifying a favorite artist, a specific song, or a semantic tag (e.g., genre, emotion, instrument.) A backend playlist algorithm then generates a sequence of songs that is related to the seed concept. While the exact implementation details of various commercial systems are trade secrets, different companies use different forms of music metadata to identify relevant songs. For example, Pandora relies on the content-based music analysis by human experts [17] 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. [9] 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 [11] 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. We take a similar Markov chain approach, but 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., [8, 13, 4, 19].) Our work is most closely related to research that involves automatically *learning* the music embedding. For example, Platt et al. use semantic tags to learn a Gaussian process kernel function between pairs of songs [13]. More recently, Weston et al. learn an embedding over a joint semantic space of audio features, tags and artists by optimizing an evaluation metric (Precision at k) for various music retrieval tasks [19]. Our approach, however, is substantially different from these existing methods, since it explicitly models the sequential nature of playlists.

Modeling playlists as a Markov chain connects to a large body of work on sequence modeling in natural language processing and speech recognition. In those applications, a language model of the target language is used to disambiguate uncertainty in the acoustic signal or the translation model. Smoothed n-gram models (see e.g. [6]) are the most commonly used method in language modeling, and we will compare against such models in our experiments. However, in natural language processing and speech recognition n-grams are typically used as part of a Hidden Markov Model (HMM)[14], not in a plain Markov Model as in our paper. In the HMM model, each observation in sequence is governed by an hidden state that evolves in Markovian fashion. The goal for learning to estimate the transition probability between hidden states as well as the probability of the observations conditioned on the hidden states. Using singular value decomposition, recent works on embedding the HMM distribution into a reproducing kernel Hilbert space [16, 5] circumvent the inference of the hidden states and make the model usable as long as kernel can be defined on the domain of observation. While both this work and our work make use of embeddings in the context of Markov chains, the two approaches solve very different problems.

Sequenced prediction also has important applications and related work in other domains. For example, Rendle et al. [15] consider the problem of predicting what a customer would have in his next basket of online purchasing. They model the transition probabilities between items in two consecutive baskets, and the tensor decomposition technique they use can be viewed as embedding in a way. While both are sequence prediction problems, the precise modeling problems are different.

Independent of and concurrent with our work, Aizenberg et al. [1] developed a model related to ours. The major difference lies in two aspects. First, they focus less on the sequential aspect of playlists, but more on using radio playlists as proxies for user preference data. Second, their model is based on inner products, while we embed using Euclidean distance. Euclidian distance seems a more natural choice for rendering an easy-to-understand visualization from the embeddings. Related is also work by Zheleva et al. [21]. Their model, however, is different from ours. They use a Latent Dirichlet Allocation-like graphical model to capture the hidden taste and mood of songs, which is different from our focus.

3. METRIC MODEL 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 bigram (or n-gram models more generally) have been widely used in language modeling for speech recognition and machine translation with great success [6]. 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 difference 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 and song sequences. Unlike n-gram models that treat words as atomic units without metric relationships between each other, our approach seeks to model coherent

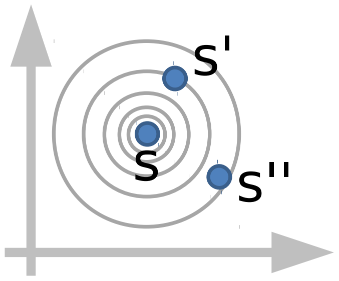

Figure 1: Illustration of the Single-Point Model. The probability of some other song following s depends on its Euclidean distance to s.

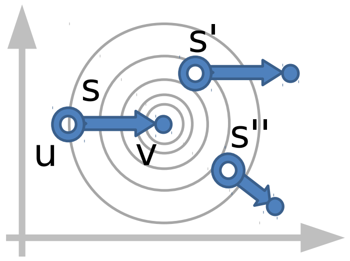

Figure 2: Illustration of the Dual-Point Model. The probability of some other song following s depends on the Euclidean distance from the exit vector $V(s)$ of s to the target song's entry vector $U(\cdot)$.

playlists as paths through a latent space. In particular, songs are embedded as points (or multiple points) in this space so that Euclidean distance between songs reflects the transition probabilities. The key learning problem is to determine the location of each song using existing playlists as training data. Once each song is embedded, our model can assign meaningful transition probabilities even to those transitions that were not seen in the training data.

Note that our approach does not rely on explicit features describing songs. However, explicit song features can easily be added to our transition model as outlined below. We will now introduce two approaches to modeling $\Pr(p)$ that both create an embedding of playlists in Euclidean space.

3.1 Single-Point Model

In the simplest model as illustrated in Figure 1, we represent 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:

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

We will typically abbreviate the partition function in the denominator as $Z(p^{[i-1]})$ and the distance $||X(s)-X(s')||_2$ as $\Delta(s,s')$ for brevity. Using a Markov model with this transition distribution, we can now define the probability the of an entire playlist of a given length k_p as

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

Our method 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, 18], 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 likelihood approach. Denoting with X the matrix of feature vectors describing all songs in the collection $\mathcal{S}$, this leads to the following training problem:

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

Equivalently, we can maximize the log-likelihood

$$L(D|X) = \sum_{p\in D}\sum_{i=1}^{k_p} -\Delta(p^{[i]},p^{[i-1]})^2 - \log(Z(p^{[i-1]})). \tag{5}$$

In Section 5, we describe how to solve this optimization problem efficiently, and we explore various methods for avoiding overfitting through regularization in Section 3.3. First, however, we extend the basic single-point model to a model that represents each song through a pair of points.

3.2 Dual-Point Model

Representing each song using a single point $X(s)$ as in the previous section has at least two limitations. First, the Euclidean metric $||X(s)-X(s')||_2$ that determines the transition distribution is symmetric, even though the end of a song may be drastically different from its beginning. In this case, the beginning of song s may be incompatible with song s' altogether, and a transition in the opposite direction – from s' to s – should be avoided. Second, some songs may be good transitions between genres, taking a playlist on a trajectory away from the current location in latent space.

To address these limitations, we now propose to model each song s using a pair $(U(s), V(s))$ of points. We call $U(s)$ the "entry vector" of song s, and $V(s)$ the "exit vector". An illustration of this model is shown in Figure 2. Each song s is depicted as an arrow connecting $U(s)$ to $V(s)$. The "entry vector" $U(s)$ models the interface to the previous song in the playlist, while the "exit vector" $V(s)$ models the interface to the next song. The transition from song s to s' is then described by a logistic model relating the exit vector $V(s)$ of song s to the entry vector $U(s')$ of song s'. Adapting our notation for this setting by representing the asymmetric song divergence $||V(s)-U(s')||_2$ as $\Delta_2(s,s')$ and the corresponding dual-point partition function as $Z_2(s)$, we obtain the following probabilistic model of a playlist.

$$\Pr(p) = \prod_{i=1}^{k_p} \Pr(p^{[i]}|p^{[i-1]}) = \prod_{i=1}^{k_p} \frac{e^{-\Delta_2(p^{[i]},p^{[i-1]})^2}}{Z_2(p^{[i-1]})} \tag{6}$$

Similar to Eq. (4), computing the embedding vectors $(U(s), V(s))$ for each song can be phrased as a maximum-likelihood problem for a given training sample of playlists $D = (p_1, ..., p_n)$, where V and U are the matrices containing the respective entry and exit vectors for all songs.

$$(V, U) = \operatorname*{argmax}_{V,U \in \Re^{|\mathcal{S}| \times d}} \prod_{p \in D} \prod_{i=1}^{k_p} \frac{e^{-\Delta_2(p^{[i]}, p^{[i-1]})^2}}{Z_2(p^{[i-1]})} \quad (7)$$

As in the single-point case, it is again equivalent to maximize the log-likelihood:

$$L(D|V, U) = \sum_{p \in D} \sum_{i=1}^{k_p} -\Delta_2(p^{[i]}, p^{[i-1]})^2 - \log(Z_2(p^{[i-1]})) \quad (8)$$

3.3 Regularization

While the choice of dimensionality d of the latent space $\mathcal{M}$ provides some control of overfitting, it is desirable to have more fine-grained control. We therefore introduce the following norm-based regularizers that get added to the log-likelihood objective.

The first regularizer penalizes the Frobenius norm of the matrix of feature vectors, leading to

$$X = \operatorname*{argmax}_{X \in \Re^{|\mathcal{S}| \times d}} L(D|X) - \lambda ||X||_F^2 \quad (9)$$

for the single point model, and

$$(V, U) = \operatorname*{argmax}_{V,U \in \Re^{|\mathcal{S}| \times d}} L(D|V, U) - \lambda(||V||_F^2 + ||U||_F^2) \quad (10)$$

for the dual point model. λ is the regularization parameter which we will set by cross-validation. For increasing values of λ, this regularizer encourages vectors to stay closer to the origin. This leads to transition distributions $\Pr(p^{[i]}|p^{[i-1]})$ that are closer to uniform.

For the dual-point model, it also makes sense to regularize by the distance between the entry and exit vector of each song. For most songs, these two vectors should be close. This leads to the following formulation,

$$(V, U) = \operatorname*{argmax}_{V,U \in \Re^{|\mathcal{S}| \times d}} L(D|V, U) - \lambda(||V||_F^2 + ||U||_F^2) \quad (11)$$
$$-\nu \sum_{s \in \mathcal{S}} \Delta_2(s, s)^2$$

where ν is a second regularization parameter.

3.4 Extending the Model

The basic LME model can be extended in a variety of ways. We have already seen how the dual-point model can account for the directionality of playlists. To further demonstrate its modeling flexibility, consider the following extensions to the single-point model. These extension can also be added to the dual-point model in a straightforward way.

Popularity. The basic LME models have only limited means of expressing the popularity of a song. By adding a separate "popularity boost" b_i to each song s_i, the resulting transition model

$$\Pr(p^{[i]}|p^{[i-1]}) = \frac{e^{-\Delta(p^{[i]}, p^{[i-1]})^2 + b_i}}{\sum_j e^{-\Delta(s_j, p^{[i-1]})^2 + b_j}} \quad (12)$$

can separate the effect of a song's popularity from the effect of its similarity in content to other songs. This can normalize the resulting embedding space with respect to popularity, and it is easy to see that training the popularity scores b_i as part of Eq. (12) does not substantially change the optimization problem.

User Model. The popularity score is a simple version of a preference model. In the same way, more complex models of song quality and user preference can be included as well. For example, one can add a matrix factorization model to explain user preferences independent of the sequence context, leading to the following transition model.

$$\Pr(p^{[i]}|p^{[i-1]}, u) = \frac{e^{-\Delta(p^{[i]}, p^{[i-1]})^2 + A(p^{[i]})^T B(u)}}{\sum_j e^{-\Delta(s_j, p^{[i-1]})^2 + A(s_j)^T B(u)}} \quad (13)$$

Analogous to models like in [7], $A(s)$ is a vector describing song s and $B(u)$ is a vector describing the preferences of user u.

Semantic Tags. Many songs have semantic tags that describe genre and other qualitative attributes of the music. However, not all songs are tagged, and tags do not follow a standardized vocabulary. It would therefore be desirable to embed semantic tags in the same Euclidean space as the songs, enabling the computation of (semantic) distances between tags, as well as between tags and (untagged) songs. This can be achieved by modeling the prior distribution of the location of song s based on its tags $T(s)$ in the following way.

$$\Pr(X(s)|T(s)) = \mathcal{N}\left(\frac{1}{|T(s)|} \sum_{t \in T(s)} M(t),\ \frac{1}{2\lambda} I_d\right) \quad (14)$$

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. Again, simultaneously optimizing song embeddings $X(s)$ and tag embeddings $M(t)$ does not substantially change the optimization problem during training. This extended embedding model for songs and tags is described in more detail in [12].

Observable Features. Some features may be universally available for all songs, in particular features derived from the audio signal via automated classification. Denote these observable features of song s as $O(s)$. We can then learn a positive-semidefinite matrix W similar to [10], leading to the following transition model.

$$\Pr(p^{[i]}|p^{[i-1]}) = \frac{e^{-\Delta(p^{[i]}, p^{[i-1]})^2 + O(p^{[i]})^T W O(p^{[i-1]})}}{\sum_j e^{-\Delta(s_j, p^{[i-1]})^2 + O(s_j)^T W O(p^{[i-1]})}} \quad (15)$$

Long-Range Dependencies. A more fundamental problem is the modeling of long-range dependencies in playlists. While it is straightforward to add extensions for modeling closeness to some seed song – either during training, or at the time of playlist generation as discussed in Section 4 – modeling dependencies beyond n-th order Markov models is an open question. However, submodular diversification models from information retrieval (e.g. [20]) may provide interesting starting points.

4. GENERATING PLAYLISTS

From a computational perspective, generating new playlists is very straightforward. Given a seed location in the embedding space, a playlist is generated through repeated sampling from the transition distribution. From a usability perspective, however, there are two problems.

First, how can the user determine a seed location for a playlist? Fortunately, the metric nature of our models gives many opportunities for letting the user specify the seed location. It can be either a single song, the centroid of a set of songs (e.g. by a single artist), or a user may graphically select a location through a map similar to the one in Figure 3. Furthermore, we have shown in other work [12] how songs and social tags can be jointly embedded in the metric space, making it possible to specify seed locations through keyword queries for semantic tags.

Second, the playlist model that is learned represents an average model of what constitutes a good playlists. Each particular user, however, may have preferences that are different from this average model at any particular point in time. It is therefore important to give the user some control over the playlist generation process. Fortunately, our model allows a straightforward parameterization of the transition distribution. For example, through the parameters α, β and γ in the following transition distribution

$$\Pr(p^{[i]}|p^{[i-1]}, p^{[0]}) = \frac{e^{-\alpha\Delta(p^{[i]},p^{[i-1]})^2+\beta b_i-\gamma\Delta(p^{[i]},p^{[0]})^2}}{Z(p^{[i-1]}, p^{[0]}, \alpha, \beta, \gamma)}, \quad (16)$$

the user can influence meaningful and identifiable properties of the playlists that get generated. For example, by setting α to a value that is less than 1, the model will take larger steps. By increasing β to be greater than 1, the model will focus on popular songs. And by setting γ to a positive value, the playlists will tend to stay close to the seed location. It is easy to imagine other terms and parameters in the transition distribution as well.

To give an impression of the generated playlists and the effects of the parameters, we provide an online demo at `http://lme.joachims.org`.

5. SOLVING THE OPTIMIZATION PROBLEMS

In the previous section, the training problems were formulated as the optimization problems in Eq. (5) and (8). While both have a non-convex objective, we find that the stochastic gradient algorithm described in the following robustly finds a good solution. Furthermore, we propose a heuristic for accelerating gradient computations that substantially improves runtime.

5.1 Stochastic Gradient Training

We propose to solve optimization problems (5) and (8) using the following stochastic gradient method. We only describe the algorithm for the dual-point model, since the algorithm for the single-point model is easily derived from it.

We start with random initializations for U and V. We also calculate a matrix T whose elements T_{ab} are the number of transitions from the s_a to s_b in the training set. Note that this matrix is sparse and always requires less storage than the original playlists. Recall that we have defined $\Delta_2(s_a, s_b)$ as the song divergence $||U(s_a) - V(s_b)||_2$ and $Z_2(s_a)$ as the dual-point partition function $\sum_{l=1}^{|\mathcal{S}|} e^{-\Delta_2(s_a,s_l)^2}$. We can now equivalently write the objective in Eq. (8) as

$$L(D|U, V) = \sum_{a=1}^{|\mathcal{S}|}\sum_{b=1}^{|\mathcal{S}|} T_{ab}\, l(s_a, s_b) - \Omega(V, U) \quad (17)$$

where $\Omega(V, U)$ is the regularizer and $l(s_a, s_b)$ is the "local" log-likelihood term that is concerned with the transition from s_a to s_b.

$$l(s_a, s_b) = \quad -\Delta_2(s_a, s_b)^2 - \log(Z_2(s_a)) \quad (18)$$

Denoting with $1^{[x=y]}$ the indicator function that returns 1 if the equality is true and 0 otherwise, we can write the derivatives of the local log-likelihood terms and the regularizer as

$$\frac{\partial l(s_a, s_b)}{\partial U(s_p)} = 1^{[a=p]} 2\left[-\overrightarrow{\Delta}_2(s_a, s_b) + \frac{\sum_{l=1}^{|\mathcal{S}|} e^{-\Delta_2(s_a,s_l)^2}\overrightarrow{\Delta}_2(s_a, s_l)}{Z_2(s_a)}\right]$$

$$\frac{\partial l(s_a, s_b)}{\partial V(s_q)} = 1^{[b=q]} 2\overrightarrow{\Delta}_2(s_a, s_b) - 2\frac{e^{-\Delta_2(s_a,s_q)^2}\overrightarrow{\Delta}_2(s_a, s_q)}{Z_2(s_a)}$$

$$\frac{\partial\Omega(V, U)}{\partial U(s_p)} = 2\lambda U(s_p) - 2\nu\overrightarrow{\Delta}_2(s_p, s_p)$$

$$\frac{\partial\Omega(V, U)}{\partial V(s_p)} = 2\lambda V(s_p) + 2\nu\overrightarrow{\Delta}_2(s_p, s_p)$$

where we used $\overrightarrow{\Delta}_2(s, s')$ to denote the vector $V(s) - U(s')$.

We can now describe the actual stochastic gradient algorithm. The algorithm iterates through all songs s_p in turn and updates the exit vectors for each s_p by

$$U(s_p) \leftarrow U(s_p) + \frac{\tau}{N}\left[\sum_{b=1}^{|\mathcal{S}|} T_{pb}\frac{\partial l(s_p, s_b)}{\partial U(s_p)} - \frac{\partial\Omega(V, U)}{\partial U(s_p)}\right]. \quad (19)$$

For each s_p, it also updates the entry vector for each possible transition (s_p, s_q) via

$$V(s_q) \leftarrow V(s_q) + \frac{\tau}{N}\left[\sum_{b=1}^{|\mathcal{S}|} T_{pb}\frac{\partial l(s_p, s_b)}{\partial V(s_q)} - \frac{\partial\Omega(V, U)}{\partial V(s_q)}\right]. \quad (20)$$

τ is a predefined learning rate and N is the number of transitions in training set. Note that grouping the stochastic gradient updates by exit songs s_p as implemented above is advantageous, since we can save computation by reusing the partition function in the denominator of the local gradients of both $U(s_p)$ and $V(s_q)$. More generally, by storing intermediate results of the gradient computation, a complete iteration of the stochastic gradient algorithm through the full training set can be done in time $O(|\mathcal{S}|^2)$. We typically run the algorithm for $T = 100$ or 200 iterations, which we find is sufficient for convergence.

5.2 Landmark Heuristic for Acceleration

The $O(|\mathcal{S}|^2)$ runtime of the algorithm makes it too slow for practical applications when the size of $\mathcal{S}$ is sufficiently large. The root of the problem lies in the gradient computation, since for every local gradient one needs to consider the transition from the exit song to all the songs in $\mathcal{S}$. This leads to $O(|\mathcal{S}|)$ complexity for each update steps. However, considering all songs is not really necessary, since most songs are not likely targets for a transition anyway. These songs contribute very little mass to the partition function and excluding them will only marginally change the training objective.

We therefore formulate the following modified training problem, where we only consider a subset C_i as possible successors for s_i.

$$L(D|U, V) = \sum_{a=1}^{|\mathcal{S}|}\sum_{s_b\in C_a} T_{ab}\, l(s_a, s_b) - \Omega(V, U) \quad (21)$$

This reduces the complexity of a gradient step to $O(|C_i|)$. The key problem lies in identifying a suitable candidate set C_i for each s_i. Clearly, each C_i should include at least most of the likely successors of s_i, which lead us to the following landmark heuristic.

We randomly pick a certain number (typically 50) of songs and call them landmarks, and assign each song to the nearest landmark. We also need to specify a threshold $r \in [0, 1]$. Then for each s_i, its direct successors observed in the training set are first added to the subset C_i^r, because these songs are always needed to compute the local log-likelihood. We keep adding songs from nearby landmarks to the subset, until ratio r of the total songs has been included. This defines the final subset C_i^r. By adopting this heuristic, the gradients of the local log-likelihood become

$$\frac{\partial l(s_a,s_b)}{\partial U(s_p)} = 1^{[a=p]} 2 \left[-\overrightarrow{\Delta}_2(s_a,s_b) + \frac{\sum_{s_l \in C_p^r} e^{-\Delta_2(s_a,s_l)^2} \overrightarrow{\Delta}_2(s_a,s_l)}{Z^r(s_a)} \right]$$

$$\frac{\partial l(s_a,s_b)}{\partial V(s_q)} = 1^{[b=q]} 2 \overrightarrow{\Delta}_2(s_a,s_b) - 2 \frac{e^{-\Delta_2(s_a,s_q)^2} \overrightarrow{\Delta}_2(s_a,s_q)}{Z^r(s_a)},$$

where $Z^r(s_a)$ is the partition function restricted to C_a^r, namely $\sum_{s_l \in C_a^r} e^{-\Delta_2(s_a,s_l)^2}$. Empirically, we update the landmarks every 10 iterations[1], and fix them after 100 iterations to ensure convergence.

5.3 Implementation

We implemented our methods in C. The code is available online at `http://lme.joachims.org`.

6. EXPERIMENTS

In the following experiments we will analyze the LME in comparison to n-gram baselines, explore the effect of the popularity term and regularization, and assess the computational efficiency of the method.

To collect a dataset of playlists for our empirical evaluation, we crawled *Yes.com* during the period from Dec. 2010 to May 2011. Yes.com is a website that provides radio playlists of hundreds of stations in the United States. By using the web based API[2], one can retrieve the playlists of the last 7 days for any station specified by its genre. Without taking any preference, we collect as much data as we can by specifying all the possible genres. We then generated two datasets, which we refer to as *yes_small* and *yes_big*. In the small dataset, we removed the songs with less than 20, in the large dataset we only removed songs with less than 5 appearances. The smaller one is composed of $3,168$ unique songs. It is then divided into into a training set with $134,431$ transitions and a test set with $1,191,279$ transitions. The larger one contains $9,775$ songs, a training set with $172,510$ transitions and a test set with $1,602,079$ transitions. The datasets are available for download at `http://lme.joachims.org`.

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 the 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. One should note that the division of training and test set is done so that each song appears at least once in the training set. This was done to exclude the case of encountering a new song when doing testing, which any method would need to treat as a special case and impute some probability estimate.

[1]A iteration means a full pass on the training dataset.

[2]`http://api.yes.com`

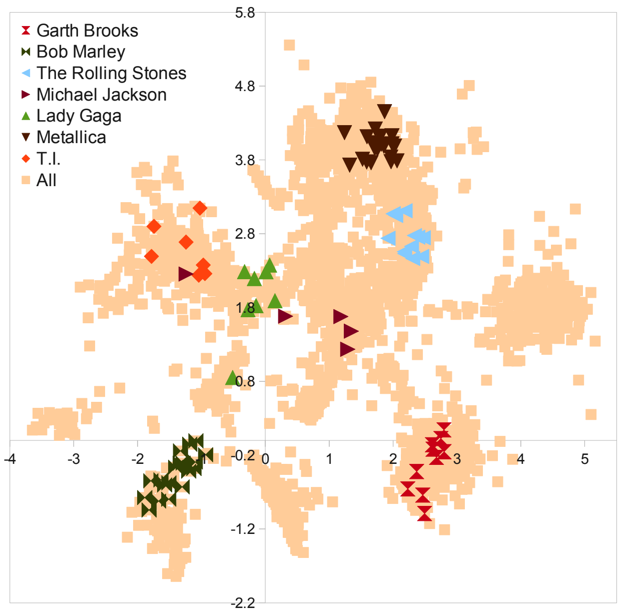

Figure 3: Visual representation of an embedding in two dimensions with songs from selected artists highlighted

6.1 What do embeddings look like?

We start with giving a qualitative impression of the embeddings that our method produces. Figure 3 shows the two-dimensional single-point embedding of the *yes_small* dataset. Songs from a few well-known artists are highlighted to provide reference points in the embedding space.

First, it is interesting to note that songs by the same artist cluster tightly, even though our model has no direct knowledge of which artist performed a song. Second, logical connections among different genres are well-represented in the space. For example, consider the positions of songs from Michael Jackson, T.I., and Lady Gaga. Pop songs from Michael Jackson could easily transition to the more electronic and dance pop style of Lady Gaga. Lady Gaga's songs, in turn, could make good transitions to some of the more dance-oriented songs (mainly collaborations with other artists) of the rap artist T.I., which could easily form a gateway to other hip hop artists.

While the visualization provides interesting qualitative insights, we now provide a quantitative evaluation of model quality based on predictive power.

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

We first compare our models against baseline methods from Natural Language Processing. We consider the following models.

Uniform Model. The choices of any song are equally likely, with the same probability of $1/|\mathcal{S}|$.

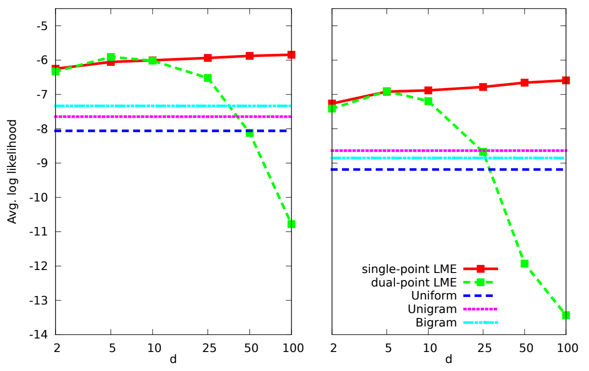

Figure 4: Single/Dual-point LME against baseline on *yes_small*(left) and *yes_big*(right). d is the dimensionality of the embedded space.

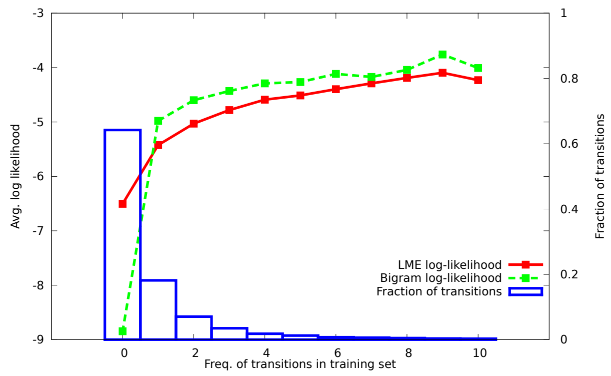

Figure 5: Log likelihood on testing transitions with respect to their frequencies in the training set on *yes_small*

Unigram Model. Each song s_i is sampled with probability $p(s_i) = \frac{n_i}{\sum_j n_j}$, where n_i is the number of appearances of s_i in the training set. $p(s_i)$ can be considered as the popularity of s_i. Since each song appears at least once in the training set, we do not need to worry about the possibility of $p(s_i)$ being zero in the testing phase.

Bigram Model. Similar to our models, the bigram model is also a first-order Markov model. However, transition probabilities $p(s_j|s_i)$ are estimated directly for every pair of songs. Note that not every transition from s_i to s_j in the test set also appears in the training set, and the corresponding $p(s_i|s_j)$ will just give us minus infinity log likelihood contribution when testing. We adopt the Witten-Bell smoothing [6] technique to solve this problem. The main idea is to use the transition we have seen in the training set to estimate the counts of the transitions we have not seen, and then assign them nonzero probabilities.

We train our LME models without heuristic on both *yes_small* and *yes_big*. The resulting log-likelihood on the test set is reported in Figure 4, where d is the dimensionality of the embedding space. Over the full range of d the single-point LME outperforms the baselines by at least one order 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 single-point LME over the baselines is even larger for *yes_big*.

The dual-point model performs equally well for models with low dimension, but shows signs of overfitting for higher dimensionality. We will see in Section 6.4 that regularization can mitigate this problem.

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 models. The following section analyzes in more detail where the conventional bigram model fails, while the single-point LME shows no signs of overfitting.

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

We now explore in more detail why the LME model outperforms the conventional bigram model. In particular, we explore the extent to which 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 about 64 percent of the test transitions were not at all observed in our training set.

For both the single-point LME and the bigram model on the small dataset, Figure 5 shows the log-likelihood of the test transitions conditioned on how often that transition was observed in the training set. The bar graph illustrates what percentage of test transitions had that given number of occurrences 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.

6.4 What are the effects of regularization?

We now explore whether additional regularization as proposed in Section 3.3 can further improve performance.

For the single-point model on *yes_small*, Figure 6 shows a comparison between the norm-based regularizer (R1) and the unregularized models across dimensions 2, 5, 10, 25, 50 and 100. For each dimension, the optimal value of λ was selected out of the set $\{0.0001, 0.001, 0.01, 0.1, 1, 10, 20, 50, 100, 500, 1000\}$. It can be seen that the regularized models offer no substantial benefit over the unregularized model. We conjecture that the amount of training data is already sufficient to estimate the (relatively small) number of parameters of the single-point model.

Figure 7 shows the results for dual-point models using three modes of regularization. R1 denotes models with $\nu = 0$, R2 denotes models with $\lambda = 0$, and R3 denotes models trained with $\nu = \lambda$. Here, the regularized models consistently outperform the unregularized ones. Starting from dimensionality 25, the improvement of adding regularization is drastic, which saves the dual-point model from being unusable for high dimensionality. It is interesting to note the effect of R2, which constrains the exit and entry points for each song to be near each other. Effectively, this squeezes

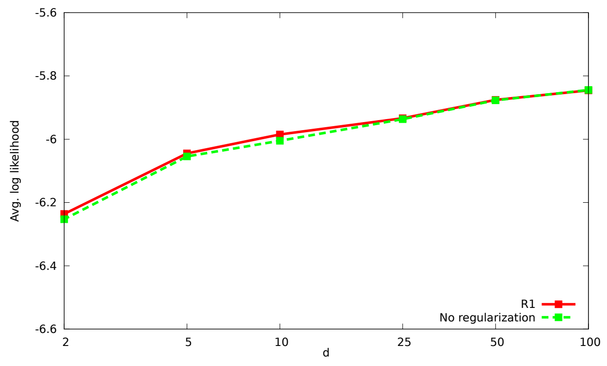

Figure 6: Effect of regularization for single-point model on *yes_small*

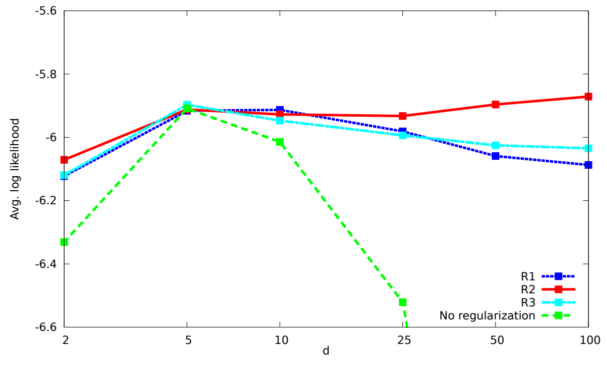

Figure 7: Effect of regularization for dual-point model on *yes_small*

the distance between the two points, bringing the dual-point model closer to the single-point model.

6.5 How directional are radio playlists?

Since the single-point model appears to perform better than the dual-point model, it raises the question of how important directionality is in playlists. We therefore conducted the following experiment. We train the dual-point model as usual for $d = 5$ on *yes_small*, but then reverse all test transitions. The average log-likelihood (over 10 training runs) on the reversed test transition is -5.960 ± 0.003, while the log-likelihood of the test transitions in the normal order is -5.921 ± 0.003. While this difference is significant according to a binomial sign test (i.e. the reversed likelihood was indeed worse on all 10 runs), the difference is very small. This provides evidence that radio playlists appear to not have many directional constraints. However, playlists for other settings (e.g. club, tango) may be more directional.

6.6 What is the effect of modeling popularity?

As discussed in Section 3.4, an added term for each song can be used to separate popularity from the geometry of the resulting embedding. In Figure 8, a comparison of the popularity-augmented model to the standard model (both with single-point) on the two datasets is shown. Adding the popularity terms substantially improves the models for low-dimensional embeddings. Even though the term adds only one parameter for each song, it can be viewed as adding

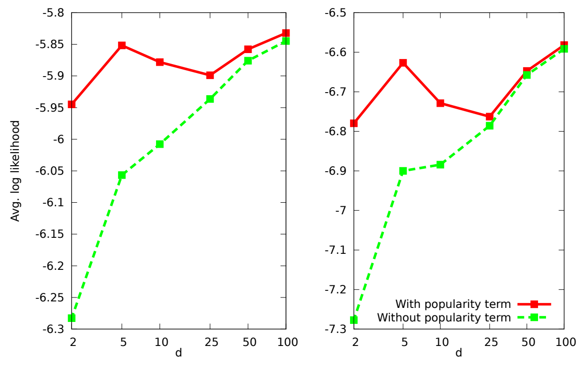

Figure 8: Effect of popularity term on model likelihood in *yes_small* (left) and *yes_big* (right)

as much expressive power as dozens of additional spatial parameters per song.

6.7 How does the landmark heuristic affect model quality?

We take the single-point model with $d = 5$ without regularization as an example in this part. We list the CPU time per iteration and log-likelihood on both datasets in Table 1 and Table 2. The landmark heuristic significantly reduces the training iteration time to what is almost proportional to r. However, for low r we see some overhead introduced by building the landmark data structure. The heuristic yields results comparable in quality to models trained without the heuristic when r reaches 0.3 on both datasets. It even gets slightly better than the no-heuristic method for higher r. This may be because we excluded songs that are very unlikely to be transitioned to, resulting in some additional regularization.

r	CPU time/s	Test log-likelihood
0.1	3.08	-6.421977
0.2	3.81	-6.117642
0.3	4.49	-6.058949
0.4	5.14	-6.043897
0.5	5.79	-6.048493
No heuristic	11.37	-6.054263

Table 1: CPU time and log-likelihood on *yes_small*

r	CPU time/s	Test log-likelihood
0.1	27.67	-7.272813
0.2	34.98	-7.031947
0.3	42.01	-6.925095
0.4	49.33	-6.897925
0.5	56.88	-6.894431
No heuristic	111.36	-6.917984

Table 2: CPU time and log-likelihood on *yes_big*

6.8 Does our method capture the coherency of playlists?

We designed the following experiment to see whether our method captures the coherency of playlists. We train our

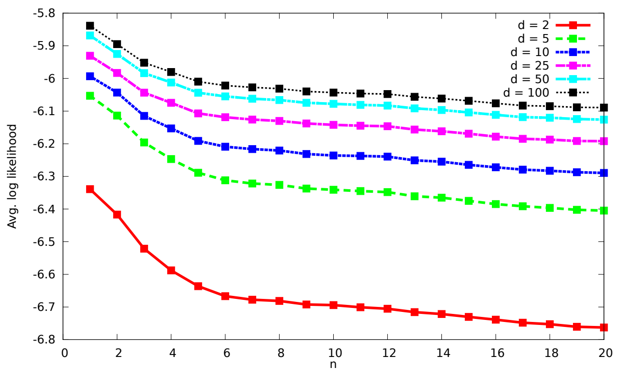

Figure 9: n-hop results on *yes_small*

model on the 1-hop transitions in training dataset, which is the same as what we did before. However, the test is done on the n-hop transitions (consider the current song and the nth song after it as a transition pair) in the test dataset. The experiments was run on *yes_small* for various values of d without regularization. Results are reported in Figure 9.

One can observe that for all values of d, the log-likelihood consistently decreases as n increases. As n goes up to 6 and above, the curves flatten out. This is evidence that our method does capture the coherency of the playlists, since songs that are sequentially close to each other in the playlists are more likely to form a transition pair.

7. CONCLUSIONS

We presented a new family of methods for learning a generative model of music playlists using existing playlists as training data. The methods do not require content features about songs, but automatically embed songs in Euclidean space similar to a collaborative filtering method. Our approach offers substantial modeling flexibility, including the ability to represent song as multiple points, to make use of regularization for improved robustness in high-dimensional embeddings, and to incorporate popularity of songs, giving users more freedom to steer their playlists. Empirically, the LME outperforms smoothed bigram models from natural language processing and leads to embeddings that qualitatively reflect our intuition of music similarity.

8. ACKNOWLEDGMENTS

This work was funded in part by NSF Awards IIS-0812091, IIS-0905467, and IIS-1217686.

9. REFERENCES

[1] N. Aizenberg, Y. Koren, and O. Somekh. Build your own music recommender by modeling internet radio streams. In *Proceedings of the 21st international conference on World Wide Web*, pages 1–10. ACM, 2012.

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

[3] T. F. Cox and M. A. Cox. *Multidimensional scaling.* Chapman and Hall, 2001.

[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. Hsu, S. Kakade, and T. Zhang. A spectral algorithm for learning hidden markov models. *Arxiv preprint arXiv:0811.4413*, 2008.

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

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

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

[9] F. Maillet, D. Eck, G. Desjardins, and P. Lamere. Steerable playlist generation by learning song similarity from radio station playlists. In *International Conference on Music Information Retrieval (ISMIR)*, 2009.

[10] B. McFee and G. R. G. Lanckriet. Metric learning to rank. In *ICML*, pages 775–782, 2010.

[11] B. McFee and G. R. G. Lanckriet. The natural language of playlists. In *International Conference on Music Information Retrieval (ISMIR)*, 2011.

[12] J. L. Moore, S. Chen, T. Joachims, and D. Turnbull. Learning to embed songs and tags for playlist prediction. http://www.joachims.org/publications/moore_etal_12a.pdf, April 2012.

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

[14] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[15] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 811–820. ACM, 2010.

[16] L. Song, B. Boots, S. Siddiqi, G. Gordon, and A. Smola. Hilbert space embeddings of hidden markov models. 2010.

[17] D. Tingle, Y. Kim, and D.Turnbull. Exploring automatic music annotation with "acoustically-objective" tags. In *ACM International Conference on Multimedia Information Retrieval*, 2010.

[18] C. Wang and D. Blei. Collaborative topic modeling for recommending scientific articles. In *SIGKDD*, 2011.

[19] 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.

[20] Y. Yue and T. Joachims. Predicting diverse subsets using structural SVMs. In *International Conference on Machine Learning (ICML)*, pages 271–278, 2008.

[21] E. Zheleva, J. Guiver, E. Mendes Rodrigues, and N. Milić-Frayling. Statistical models of music-listening sessions in social media. In *Proceedings of the 19th international conference on World wide web*, pages 1019–1028. ACM, 2010.