A Bag of Systems Representation for Music Auto-Tagging

Katherine Ellis, Emanuele Coviello, Antoni B. Chan, and Gert Lanckriet, *Senior Member, IEEE*

***Abstract*—We present a content-based automatic tagging system for music that relies on a high-level, concise "Bag of Systems" (BoS) representation of the characteristics of a musical piece. The BoS representation leverages a rich dictionary of musical codewords, where each codeword is a generative model that captures timbral and temporal characteristics of music. Songs are represented as a BoS histogram over codewords, which allows for the use of traditional algorithms for text document retrieval to perform auto-tagging. Compared to estimating a single generative model to directly capture the musical characteristics of songs associated with a tag, the BoS approach offers the flexibility to combine different generative models at various time resolutions through the selection of the BoS codewords. Additionally, decoupling the modeling of audio characteristics from the modeling of tag-specific patterns makes BoS a more robust and rich representation of music. Experiments show that this leads to superior auto-tagging performance.**

***Index Terms*—Audio annotation and retrieval, bag of systems, content-based music processing, dynamic texture model, music information retrieval.**

I. Introduction

As physical media has evolved towards digital content, recent technologies have changed the way users discover music. Millions of songs are instantly available via online music players or personal listening devices, and new songs are constantly being created. With such a large amount of data, particularly in the "long tail" of relatively unknown music, it becomes a challenge for listeners to discover new songs that align with their musical tastes. Hence there is a growing need for automated algorithms to explore and recommend musical content.

Many recommendation systems rely on semantic tags, which are words or short phrases that describe musically meaningful concepts such as genres, instrumentation, mood and usage. By bridging the gap between acoustic semantics and human semantics, tags provide a concise description of a musical piece. Tags can then be leveraged for semantic search based on transparent textual queries, such as "mellow acoustic rock", or for playlist generation based on semantic similarity to a query song. A variety of tag-based retrieval systems use annotations provided by expert musicologists [1], [2] or social services [3], and work well when provided enough annotations. However, scaling these systems to the size and needs associated with modern music collections is not practical, due to the cost of human labor and the cold start problem (i.e., the fact that songs that are not annotated cannot be retrieved, and more popular songs (in the short-head) tend to be annotated more thoroughly than unpopular songs (in the long-tail))[4]. Therefore, it is desirable to develop intelligent algorithms that automatically annotate songs with semantic tags based on the song's acoustic content. In this paper we present a new approach to content-based auto-tagging, which we build on a novel, high-level descriptor of songs that uses a vocabulary of musically meaningful codewords, each of which is a generative model that captures some prototypical textures or dynamical patterns of audio content.

A good deal of previously proposed approaches follow a "fixed" recipe—start from a collection of annotated songs, represented by a sequence of audio feature vectors, and estimate a series of statistical models, one for each tag, to learn which acoustic patterns are most predictive for each tag. Then, the tag models are used to annotate new songs—the audio features of a new song are compared to each tag model, and the song is annotated with the subset of tags whose statistical models best fit the song's audio content.

A few solutions rely on generative models, which posit that data points are randomly generated samples from a fully specified probabilistic model. This is in contrast to a discriminitive approach, which models only the conditional distribution of a tag conditioned on the observed audio data. After selecting a base model, i.e., a particular type and time scale of generative model, these approaches fine-tune an instance of this base model for each tag, through maximum likelihood estimation, to capture musical characteristics that are common to songs associated with the tag. The choice of base model is usually driven by the type of audio characteristics one intends to capture with the tag models. For example, Turnbull *et al.* [5] use Gaussian mixture models (GMMs) over a "bag of features" (BoF) representation of songs as a base model, where each audio feature represents the timbre over a short snippet of audio. Coviello *et al.* [6] use the more sophisticated dynamic texture mixture (DTM) models (i.e., linear dynamical systems), to explicitly leverage the temporal dynamics (e.g., rhythm, beat, tempo) present in short audio fragments, i.e., sequences of audio features.

Our work is motivated by the observation that these direct generative approaches have some inherent limitations. First, their modeling power is determined by the choice of *base model* (e.g., GMM, DTM, etc.) and the choice of *time scale* for audio feature extraction (e.g., length of snippets, duration of

Manuscript received December 14, 2012; revised April 30, 2013; accepted August 03, 2013. Date of publication August 21, 2013; date of current version October 24, 2013. This work was supported by Google, Inc. The work of E. Coviello and G. Lanckriet was supported by Yahoo!, Inc., the Sloan Foundation, KETI under the PHTM program, and National Science Foundation Grants CCF-0830535 and IIS-1054960. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Laurent Daudet.

K. Ellis, E. Coviello, and G. Lanckriet are with the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093 USA (e-mail: kellis@ucsd.edu; ecoviell@ucsd.edu; gert@ece.ucsd.edu).

A. B. Chan is with the Department of Computer Science, City University of Hong Kong, Kowloon Tong, Hong Kong (e-mail: abchan@cityu.edu.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASL.2013.2279318

fragments). Alternative choices will result in tag models that focus on complementary characteristics of the music signal, and no particular set of choices may be uniformly better, for all tags, than any other. For example, in [6], Coviello *et al.* reported that while the DTM models they propose outperformed GMM models (as proposed by Turnbull *et al.* in [5]) on *average*, this superior DTM performance is not consistently observed for all tags. Indeed, each method was best on a subset of tags, suggesting that different fundamental design choices are best suited to model different groups of tags. In addition, fixing a single time scale may be suboptimal, as acoustic patterns common to different tags may unfold over different time scales. A second limitation of the direct generative methodology is that fitting tag models, in particular complex ones such as DTMs, involves estimating many parameters. This may result in non-reliable estimates for tags associated with few example songs in the annotated collection.

To address these limitations, we propose a "Bag of Systems" (BoS) representation of music, which *indirectly* uses generative models to represent tag-specific characteristics. The BoS representation is a rich, high-level descriptor of musical content that uses generative models as musical codewords. Similar to the bag of words (BoW) representation commonly used in text retrieval [7] which represents textual documents as histograms over a vocabulary of English words, the BoS approach represents songs as histograms over counts of musical codewords. Given a vocabulary of musical codewords (which we call a BoS codebook), a song's BoS histogram is derived by "counting" the occurrences of each codeword in the song through probabilistic inference. Once songs are represented as a BoS histogram, a variety of supervised learning algorithms (e.g., those generally employed with BoW representations in the text mining community) can be leveraged to implement a music auto-tagger over the BoS histograms of songs.

The BoS representation enables combining the benefits of different types of generative models over various time scales in a single auto-tagger. In this paper we demonstrate that a codebook which uses a combination of Gaussian models and dynamic texture models that operate over various time scales as BoS codewords improves the performance of the auto-tagger relative to codebooks that are formed with codewords from only one base model.

The BoS codebook is similar to the classical vector-quantized (VQ) codebook of prototypical feature vectors, but contains richer and more complex codewords. A Gaussian codeword in the BoS representation captures the same information contained in a VQ codeword, i.e., the mean feature vector, but includes additional information about the variance of the cluster that is not accounted for by VQ. Additionally, the use of DT codewords in the BoS representation adds information about the temporal dynamics of sequences of feature vectors that a VQ representation cannot capture.

Another key advantage of the BoS representation is that it decouples modeling *music* from modeling *tags*. First, a vocabulary of BoS codewords that model typical characteristics of musical audio is learned from some large, representative collection of songs, which need not be annotated. With a suitable collection, these BoS codewords will be rich descriptors that can represent a wide variety of songs. Once a BoS representation for music has been obtained, auto-tagging reduces to learning recurring patterns in the BoS histograms of songs associated with each tag. The advantage is that relatively simple tag models (i.e., relative to the sophisticated generative models used at the codeword level) may suffice to capture tag-specific characteristics in the BoS histograms, while still leveraging the full descriptiveness of the codebook. In particular, estimating simpler statistical models is less prone to over-fitting and, thus, more likely to provide robust estimates for tags with few example songs in the annotated collection (used for estimating tag models). We demonstrate this advantage through experiments later in this paper.

This paper expands on results described in a previous conference publication [8], including an expanded discussion of algorithmic details and a more extensive experimental evaluation. In particular, we analyze in more depth the method we use to generate BoS codebooks, with a comparison to three alternative methods of codebook generation. We include more experiments dealing with parameter selection, e.g., the effect of the codebook size, the size of codebook song set, and the histogram smoothing parameter. We include experiments using additional time scales of base models. We also include for comparison two additional baseline methods beyond the direct generative HEM-DTM and HEM-GMM methods shown in the conference publication — one that uses VQ codebooks and one that averages probabilities given by the direct generative approaches across different time scales and models. Additionally, we include more experiments that investigate the composition of the codebook song set, using combinations of labeled and unlabeled data.

The remainder of this paper is organized as follows. In Section II we overview related work. Then in Section III we delve into the details of the BoS representation of music, including the generative models used as codewords (III-A), and algorithms for learning a BoS codebook (III-B) and for representing songs using the resulting BoS codebook (III-C). Music annotation and retrieval with BoS histograms is discussed in Section IV. After introducing the music datasets used in our experiments in Section V, we present experiments that demonstrate the effectiveness of the BoS representation for music auto-tagging in Section VI.

II. Related Work

In recent years content-based auto-tagging has seen increasing interest as a result of the growing amount of online music and the impossibility of manual labeling. Music auto-tagging has been approached from a variety of perspectives, both discriminative and generative, as well as a few unsupervised methods. Generative models previously used in auto-taggers include Gaussian mixture models (GMMs) [5], [9], hidden Markov models (HMMs) [10]–[12], hierarchical Dirichlet processes (HDPs) [13], a codeword Bernoulli average model (CBA) [14] and dynamic texture mixture models (DTMs) [6]. Previously proposed auto-taggers that rely on a discriminative framework employ algorithms such as multiple-instance learning [15], stacked SVMs [16], boosting [17], nearest-neighbor [10], logistic regression [18], locally-sensitive hashing [19] and temporal pooling with multilayer perceptrons [20]. In [21], the authors approach music auto-tagging as a low rank matrix factorization problem. Alternative approaches add information from non-audio data, such as a hybrid music

recommender system that combines usage and content data [22] and a multimodal music recommendation algorithm that combines information from music content, music context, and user context [23].

Recent work in music auto-tagging has focused on developing approaches that can take into account information at different time scales. For example, one approach uses boosting to combine features extracted at different time scales by using trees as weak classifiers [24]. Another approach integrates features computed from shorter frames over the duration of a longer analysis window, before applying a classifier [25]. Additionally, Sturm *et al.* [26] found that combining MFCCs over multiple time scales to create a unified feature vector improved performance in automatic musical instrument identification. Multiscale spectrotemporal features are used to discriminate speech from non-speech audio in [27]. In [28], the authors use an unsupervised deep convolutional representation for genre recognition and artist identification, demonstrating an improvement over raw MFCC features. In [29], scattering representations of MFCCs were used for genre classification.

Existing auto-tagging algorithms for music that employ a codebook representation of songs have only focused on vector-quantized (VQ) codebooks of raw audio features [14], [18]. One of these approaches combines multiple VQ codebooks, learned from the same set of features but with slightly different initial conditions, to reduce the variance in codebook construction [30]. In VQ codebooks, each codeword is a prototypical feature vector, while in BoS codebooks, each codeword is a generative model that describes sets or sequences of features vectors; this allows for the capture of higher-level and richer musical characteristics.

The BoS approach provides a way to combine discriminative and generative components into a single learning framework — while the modeling power of the BoS representation is achieved through generative models used as codewords, once songs are represented as BoS histograms, any standard supervised learning algorithm, discriminative or generative, can be used to learn tag models. A related approach that merges discriminative and generative learning is the use of kernels to integrate the generative models within a discriminative learning paradigm, for example through probability product kernels (PPK) [31].

A BoS approach has been used for the classification of videos [32]–[34], and a similar idea has inspired anchor modeling for speaker identification [35].

III. The Bag of Systems Representation of Music

Analogous to the bag-of-words representation of text documents, the BoS approach represents songs with respect to a codebook, where generative models are used in lieu of words. These generative models compactly characterize typical audio features and musical dynamics in songs. A song is then modeled as a composition of these codewords, just as a text document is composed of words, and a BoS histogram is constructed by counting the occurrences of each codeword in the song through probabilistic inference (see Fig. 1).

In this section, we establish a BoS representation for music. We first discuss the (generative) base models from which we will derive BoS codewords (Section III-A). We choose Gaussian and dynamic texture (DT) models to capture prototypical timbral content and temporal dynamics, respectively, although it should be noted that the BoS representation is not restricted to these choices of generative models. We then present several approaches to learn a dictionary of codewords from a collection of representative songs (Section III-B). Finally, we describe how to map songs to the BoS codebook using probabilistic inference (Section III-C).

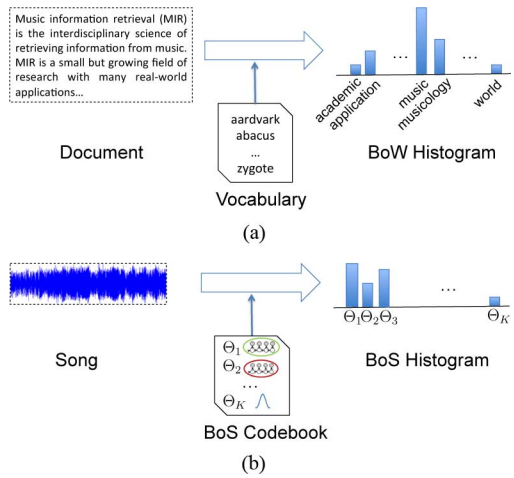

Fig. 1. (a) Bag of Words modeling process: a document is represented as a histogram over word counts. (b) Bag of Systems modeling process: a song is represented as a histogram over musical codewords, where each codeword is a generative model that captures timbral and temporal characteristics of music.

A. Generative Models as Codewords

To construct a *rich* codebook that can model diverse aspects of musical signals, we consider two different base models, the Gaussian model and the dynamic texture (DT) model [36]. Each captures distinct musical characteristics.

In particular, the audio content of a song is first represented by a sequence of audio feature vectors (e.g., MFCCs, which capture timbre) $\mathcal{Y} = \{y_1, \ldots, y_T\}$, extracted from equally spaced, half-overlapping time windows of length η (which determines the time resolution of the features), where T depends on the duration of the song and the sampling process. Gaussian codewords will capture average timbral information in *unordered* portions of $\mathcal{Y}$; DT codewords will model characteristic temporal dynamics in *ordered* subsequences of $\mathcal{Y}$. Below, we discuss each base model in more detail. To further increase the diversity of the codebook, we will consider the above base models at different time resolutions η as well.

1) Gaussian Models: A Gaussian codeword models the average timbral characteristics of the bag-of-features representation of a sequence of audio feature vectors, without taking into account the actual ordering of audio features. In particular, the model assumes each vector in the bag-of-features is generated by a multivariate Gaussian $\mathcal{N}(\mu, \Sigma)$, where μ and Σ are the mean and the covariance matrix, respectively. Hence the codeword is parameterized by $\Theta = \{\mu, \Sigma\}$.

2) Dynamic Texture Models: A dynamic texture (DT) codeword captures timbre as well as explicit temporal dynamics of an audio fragment (i.e., a sequence of audio feature vectors) by explicitly modeling the sequential ordering of the audio feature vectors.

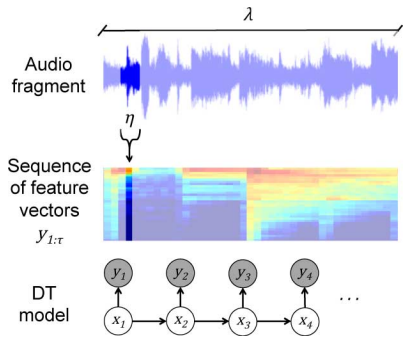

Fig. 2. A dynamic texture model represents a fragment of audio of length λ. We first compute a sequence of τ feature vectors, each computed from a window of length η. The DT models each feature vector as an observed variable y_t in a linear dynamical system.

Specifically, a DT treats a sequence $y_{1:\tau}$ of τ audio feature vectors as the output of a linear dynamical system (LDS):

$$x_t = Ax_{t-1} + v_t, \tag{1}$$

$$y_t = Cx_t + w_t + \bar{y}, \tag{2}$$

where the random variable $y_t \in \mathbb{R}^m$ encodes the timbral content (audio feature vector) at time t, and a lower dimensional hidden variable $x_t \in \mathbb{R}^n$ encodes the dynamics of the observations over time. The codeword is specified by parameters $\Theta = \{A, Q, C, R, \mu, S, \bar{y}\}$, where the state transition matrix $A \in \mathbb{R}^{n\times n}$ encodes the evolution of the hidden state x_t over time, $v_t \sim \mathcal{N}(0, Q)$ is the driving noise process, the observation matrix $C \in \mathbb{R}^{m\times n}$ encodes the basis functions for representing the observations y_t, $\bar{y}$ is the mean of the observation vectors, and $w_t \sim \mathcal{N}(0, R)$ is the observation noise. The initial condition is distributed as $x_1 \sim \mathcal{N}(\mu, S)$. We note that the columns of the observation matrix C can be interpreted as the principal components (or basis functions) of the audio feature vectors over time. Hence, each audio feature vector can be represented as a linear combination of principal components, with corresponding weights given by the current hidden state. In this way, the DT can be interpreted as a time-varying PCA representation of an audio feature vector time series.

Fig. 2 illustrates the underlying generative process of a DT modeling a sequence of feature vectors. The time scale of a DT is determined by the duration λ of an audio-fragment, and the length η of a window from which feature vectors are computed. Note that the duration λ is directly determined by η and the number of feature vectors in a sequence, τ, by $\lambda = \eta/2(\tau + 1)$, since we compute feature vectors from half-overlapping windows.

B. Codebook Generation

We are interested in compiling a *rich* codebook, which effectively quantizes the space of music-fragment models. The codebook should have a high representational power for music, i.e., contain a diverse set of codewords that can capture many different musical characteristics. First, in order to consider diverse aspects of the musical signal (e.g., timbre in short snippets of audio, or more complex timbral and dynamic patterns in longer fragments), we choose M different base models (i.e., type of generative model and time scale). We learn the codebook by learning groups of codewords, where each codeword in a group is derived from a certain base model.

More formally, we define the subset of the codebook consisting of codewords derived from the m^{th} base model as $\mathcal{V}_m = \{\Theta_i\}_{i=1}^{K_m}$, where K_m is the number of codewords derived from base model m. Hence for a given song $\mathcal{Y}$, the m^{th} subset of codewords is associated with a particular feature vector representation $\mathcal{Y}_m$ of the song at the specified time scale. For codeword subset m with a Gaussian base model, each codeword $\Theta \in \mathcal{V}_m$ takes the form $\Theta = \{\mu, \Sigma\}$, and song $\mathcal{Y}$ is represented as a bag of audio feature vectors $\mathcal{Y}_m = \{y_1, \ldots, y_{T_m}\}$, extracted from windows of length η_m. For codeword subset n with a DT base model, each codeword $\Theta \in \mathcal{V}_n$ takes the form $\Theta = \{A, Q, C, R, \mu, S, \bar{y}\}$, and song $\mathcal{Y}$ is represented as a bag of audio fragments $\mathcal{Y}_n = \{y^1_{1:\tau_n}, \ldots, y^{T_n}_{1:\tau_n}\}$. Audio fragments are sequences of τ_n feature vectors, each feature extracted from a window of length η_n. The length of an audio fragment is defined as λ_n and the step size between successive audio fragments is ν_n.

In what follows, we assume that all codeword subsets are the same size, i.e., $K_1 = K_2 = \ldots = K_M = \tilde{K}$. The union of all these subsets of codewords is the BoS codebook, $\mathcal{V} = \bigcup_{m=1}^{M} \mathcal{V}_m = \{\Theta_i\}_{i=1}^{K}$, where $K = M\tilde{K}$ is the total number of codewords in the codebook.

In practice, to compile a codebook we select a collection of representative songs $\mathcal{X}_c$. Then, based on this collection, we obtain some codewords for each of the M base models. In particular, we will learn the codewords in each codeword subset m *independently*, by first generating the corresponding representation $\mathcal{Y}_m$ for each song $\mathcal{Y} \in \mathcal{X}_c$, and then estimating the parameters for a set of $\tilde{K}$ codewords of that base model to adequately span these audio representations.

A natural approach to this estimation problem is to use the EM algorithm to directly learn a mixture of $\tilde{K}$ codewords (of base model m) from *all* audio data — i.e., the combined audio data from all the songs in $\mathcal{X}_c$. However, this becomes expensive as $\tilde{K}$ and the amount of audio data increases. Therefore, we also investigate alternative methods of codebook generation, which are expected to increase computational efficiency by learning fewer (than $\tilde{K}$) codewords at once and/or learning from (small) subsets of all audio data in $\mathcal{X}_c$.

In particular, we will consider four different procedures for learning $\tilde{K}$ codewords for subset m: (1) learn *each* of the $\tilde{K}$ codewords independently from *very small* (non-overlapping) subsets of all available audio data (e.g., from a fragment of a song in $\mathcal{X}_c$), (2) learn q small mixtures of a *few* ($K_s = \tilde{K}/q$) codewords independently from *small* (but slightly larger, non-overlapping) subsets of all available audio data (e.g., from individual songs in $\mathcal{X}_c$), (3) learn *all* $\tilde{K}$ codewords simultaneously from *all* available audio data in $\mathcal{X}_c$, either by using *standard* EM, or (4) with recourse to a more efficient, *hierarchical* EM (HEM) algorithm. Each of these methods is depicted graphically in Fig. 3. In Section VI-A we experimentally evaluate each codebook generation method, leading us to settle upon the second, song-based method of codebook generation, as the best combination of efficient learning time and good performance.

In what follows, we detail each codebook generation method, applied to learning a subset of codewords $\mathcal{V}_m$ for a single base model m. To generate a codebook from M base models, we

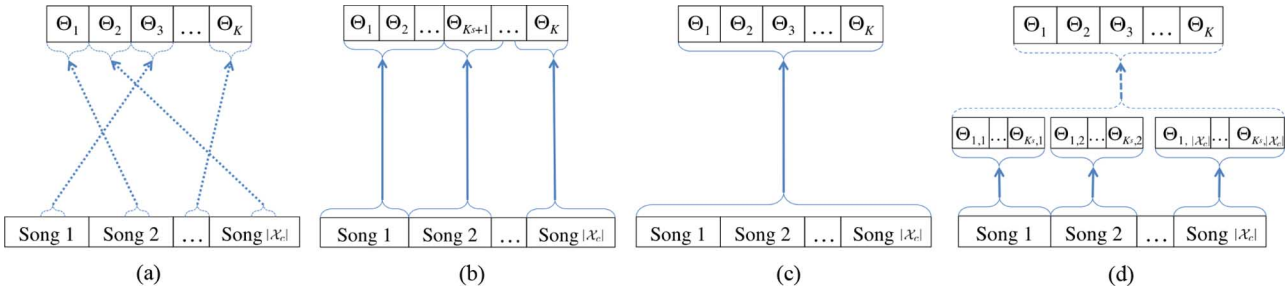

Fig. 3. Learning a BoS Codebook $\mathcal{V} = \{\Theta_i\}_{i=1}^{K}$ by (a) modeling random audio fragments with a single instance of the model, (b) modeling each song with a small mixture model, (c) modeling all available audio data with a large mixture model using standard EM, or (d) using a more efficient, hierarchical EM (HEM) algorithm. A solid arrow corresponds to applying EM, a dashed arrow to applying HEM, and a dotted arrow to direct estimation (see Section III-B.1).

repeat the procedure for each base model and pool the resulting codeword subsets.

1) Fragment-Based Procedure: An *individual codeword* $\Theta \in \mathcal{V}_m$ is learned directly from a *very small* subset of all available audio data in $\mathcal{X}_c$. In particular, to learn a Gaussian codeword, a song is selected uniformly at random from the codebook song set $\mathcal{X}_c$, represented appropriately as $\mathcal{Y}_m$, and a few (sequential) audio feature vectors are (uniformly) randomly chosen from $\mathcal{Y}_m$. The parameters of Θ are then directly estimated by computing the mean and the (diagonal) covariance matrix of these feature vectors. To learn a DT codeword, again a song is selected uniformly at random from the codebook song set $\mathcal{X}_c$, represented appropriately as $\mathcal{Y}_m$, and an audio fragment $y_{1:\tau_m}$ is (uniformly) randomly chosen from $\mathcal{Y}_m$. The parameters of Θ are directly estimated from the audio fragment, using an approximate and efficient algorithm based on principal component analysis for DT codewords [36]. The process is repeated until the desired number of codewords ($\tilde{K}$) has been collected. With complexity $O(\tilde{K})$, this is the least computationally expensive method of codebook generation, as each codeword is learned individually and efficiently from a small amount of data (as shown in Fig. 3(a)). Additionally, since each codeword is learned independently, the learning processes can easily be parallelized, reducing the complexity to $O(\tilde{K}/N_p)$, where N_p is the number of processors available.

2) Song-Based Procedure: Mixtures of a *few* codewords are learned from *individual songs* in the codebook song set $\mathcal{X}_c$ and are then pooled together to form the codeword set $\mathcal{V}_m$. More specifically, each song in $\mathcal{X}_c$ (represented appropriately as $\mathcal{Y}_m$) is modeled with a Gaussian mixture model (GMM) or dynamic texture mixture (DTM) model $\{\pi_j, \Theta_j\}_{j=1}^{K_s}$ with K_s mixture components, where the Θ_j's are Gaussian or DT components and the π_j's are the corresponding mixture weights. These mixture models are learned using the EM algorithm (EM-GMM [37] or EM-DTM [38] for Gaussian mixture models or dynamic texture mixture models, respectively). We then aggregate the mixture components (ignoring the mixture weights[1]) from each song-level model to form $\mathcal{V}_m = \{\Theta_i\}_{i=1}^{\tilde{K}}$, where $\tilde{K} = K_s|\mathcal{X}_c|$.

[1]The alternative to ignoring mixture weights would be to use the mixture weights when building the BoS histograms for each song — so a more common codeword, with a high mixture weight, is more likely to be assigned to represent a portion of audio than a rare codeword, which would have a low mixture weight. However, it makes more sense assign to a snippet of audio the codeword that *best* matches, even if it is a rare codeword, to create the most precise representation of the song. In fact, these rare codewords may have more discriminative power than more common codewords, which may occur across many tags.

This approach has a complexity of $O(dFK_s|\mathcal{X}_c|) = O(dF\tilde{K})$, where d is the average number of iterations for each run of the EM algorithm and F is the average number of data samples used as input to the EM algorithm. In the case of Gaussian codewords, F is the average number of audio feature vectors per song, while in the case of DT codewords, F is the average number of audio fragments per song. In comparison to the fragment-based procedure, the song-based procedure makes use of the entire codebook song set, and provides a regularization effect in learning the codewords by smoothing over an entire song rather than using a single fragment for estimation. As Fig. 3(b) shows, each song is associated with a few codewords, from the corresponding song model. Since song models can be learned in parallel, the complexity can be reduced to $O(dFK_s|\mathcal{X}_c|/N_p)$, where again N_p is the number of processors available.

3) Collection-Based Procedure: All the codewords in $\mathcal{V}_m$ are learned *jointly*, as one large mixture model, from all available audio data in the codebook song set $\mathcal{X}_c$. In particular, for Gaussian codewords, we pool the bag-of-features representations of all the songs in $\mathcal{X}_c$, and use this as input to the EM-GMM algorithm to learn a mixture model $\{\pi_i, \Theta_i\}_{i=1}^{\tilde{K}}$, where each Θ_i parameterizes a Gaussian and π_i is the corresponding mixture weight. For DT codewords, we pool the bag-of fragments representations of all the songs in $\mathcal{X}_c$, and use this as input to the EM-DTM algorithm to learn a mixture model $\{\pi_i, \Theta_i\}_{i=1}^{\tilde{K}}$, where each Θ_i parameterizes a DT and π_i is the corresponding mixture weight. Each individual mixture component becomes a codeword in $\mathcal{V}_m$ (mixture weights are discarded). Therefore, the number of mixture components corresponds to $\tilde{K}$ (the desired number of codewords in $\mathcal{V}_m$). The computational complexity of this procedure is $O(dN\tilde{K}) = O(dF|\mathcal{X}_c|\tilde{K})$, where $N = F|\mathcal{X}_c|$ is the number of samples used as input to the EM algorithm. This approach has a factor of $|\mathcal{X}_c|$ higher time complexity than the song-based procedure. However, the complexity can be improved by subsampling the inputs to the EM algorithm, i.e., using $N < F|\mathcal{X}_c|$. The EM algorithm usually requires storing in memory (RAM) all available audio data, and hence scales poorly to large codebook song sets. Even though a parallel implementation of EM could effectively alleviate the high memory requirements (at the likely modest cost of globally coordinating the local computations), the cumulative CPU time requirements would be still elevated because each iteration would still need to process the entire data. Fig. 3(c) shows the collection based procedure.

4) Hierarchical Collection-Based Procedure: Similar to the previous approach, this procedure learns a mixture model with $\tilde{K}$ components from all available audio data in $\mathcal{X}_c$, but now using a more efficient two-stage procedure. First, a series of mixture models are estimated from small subsets of the data, using EM. Then, after aggregating all mixture components, the resulting (very large) mixture is reduced to a mixture with $\tilde{K}$ components using the hierarchical EM algorithm (HEM) [33], [39]. Specifically, a large collection of codewords $\mathcal{V}_m^{(b)}$ of size $\tilde{K}^{(b)}$ is learned from $\mathcal{X}_c$ with the song-based procedure, using K'_s mixture components per song. Then, the $\tilde{K}^{(b)} = K'_s|\mathcal{X}_c|$ codewords in $\mathcal{V}_m^{(b)}$ are clustered to form a codeword set $\mathcal{V}_m$ of size $\tilde{K} < \tilde{K}^{(b)}$ using the HEM algorithm for GMMs [39] or the HEM algorithm for DTMs [33]. In this case, the HEM algorithm takes as input a large mixture model $\{1/\tilde{K}^{(b)}, \Theta_i^{(b)}\}_{i=1}^{\tilde{K}^{(b)}}$ (weighing all mixture components equally), and produces a reduced mixture model with fewer components $\{\pi_j, \Theta_j\}_{j=1}^{\tilde{K}}$, where each component Θ_j is a *novel codeword* that groups, i.e., clusters, some of the original codewords. As before, each individual mixture component in the output model becomes a codeword.

The complexity is $O(dFK'_s|\mathcal{X}_c| + d'\tilde{K}^{(b)}\tilde{K})$, where d' is the average number of iterations of the HEM algorithm. Assuming $F \approx \tilde{K}$ and $d' \approx d$, this becomes $O(d'|\mathcal{X}_c|(F + \tilde{K})K'_s)$, saving a factor $\tilde{K}/K'_s$ compared to the previous, direct collection-based procedure. Furthermore, since learning the song models in the first stage can be parallelized, the complexity can be further reduced to $O(dFK'_s|\mathcal{X}_c|/N_p + d'\tilde{K}^{(b)}\tilde{K})$. As before, N_p is the number of processors available. Fig. 3(d) shows the two stages of this method of codebook generation.

C. Representing Songs With the Codebook

Once a codebook is available, a song $\mathcal{Y}$ is represented by a BoS histogram $\mathbf{b} \in \mathbb{R}^K$, where $b[i]$ is the weight of codeword i in the song. To count the number of "occurrences" of a given codeword $\Theta_i \in \mathcal{V}_m$ in the song, we start from the appropriate song representation, $\mathcal{Y}_m$, at the appropriate time scale. At various time points t in the song (e.g., every ν_m seconds), we compare the likelihood of Θ_i to the likelihood of all other codewords derived from the same codeword subset, i.e., all $\Theta \in \mathcal{V}_m$ (since likelihoods are only comparable between similar models with the same time scale). We count an occurrence of Θ_i at t if it has the highest likelihood of all the codewords in $\mathcal{V}_m$ given the audio data $\mathbf{y}^t$, i.e., $\Theta_i = \arg\max_{\Theta \in \mathcal{V}_m} P(\mathbf{y}^t|\Theta)$. For GMM codewords, $\mathbf{y}^t$ is a single audio feature vector, extracted from a window of width η_m starting at t, while for DTM codewords, $\mathbf{y}^t$ is a sequence of τ_m such feature vectors, extracted at intervals of $\eta_m/2$.

Finally, codeword counts are normalized to frequencies to obtain the BoS histogram $\mathbf{b}$ for song $\mathcal{Y}$:

$$b[i] = \frac{1}{M|\mathcal{Y}_m|} \sum_{\mathbf{y}^t \in \mathcal{Y}_m} \mathbb{1}[\Theta_i = \arg\max_{\Theta \in \mathcal{V}_m} P(\mathbf{y}^t|\Theta)]. \quad (3)$$

We first normalize within each codeword subset — by the number of fragments, $|\mathcal{Y}_m|$, in $\mathcal{Y}_m$ — and then between subsets — by the number of base models, M.

Equation (3) is derived from the standard term frequency representation of a document, where each audio fragment is replaced by its closest codeword. However, this can become unstable when a fragment has multiple codewords with (approximately) equal likelihoods, which is more likely to happen as the size of the codebook increases. To counteract this effect, we generalize Equation (3) to support the assignment of multiple codewords at each point in the song. We introduce a smoothing parameter $k \in \{1, 2, \ldots, |\mathcal{V}_m|\}$. In particular, we assign the k most likely codewords (within one subset) to each portion of audio. The softened histogram is then constructed as:

$$b[i] = \frac{1}{M|\mathcal{Y}_m|} \sum_{\mathbf{y}^t \in \mathcal{Y}_m} \frac{1}{k} \mathbb{1}[\Theta_i \in \arg\max_{\Theta \in \mathcal{V}_m}{}^k P(\mathbf{y}^t|\Theta)], \quad (4)$$

where the additional normalization factor of $1/k$ ensures that b is still a valid multinomial for $k > 1$. The $\arg\max^k$ is formally defined for a function f over a discrete set U as $\{x : \nexists\, Y \subset U$ such that $x \notin Y, |Y| = k$, and $\forall z \in Y, f(z) \geq f(x)\}$.

IV. Music Annotation and Retrieval Using the Bag-of-Systems Representation

Once a BoS codebook $\mathcal{V}$ has been generated and songs represented by BoS histograms, a content-based auto-tagger may be obtained based on this representation — by modeling the characteristic codeword patterns in the BoS histograms of songs associated with each tag in a given vocabulary. In this section, we formulate annotation and retrieval as a multiclass multi-label classification of BoS histograms and discuss the algorithms used to learn tag models.

A. Annotation and Retrieval With BoS Histograms

Formally, assume we are given a training dataset $\mathcal{X}_t$, i.e., a collection of songs annotated with semantic tags from a vocabulary $\mathcal{T}$. Each song s in $\mathcal{X}_t$ is associated with a BoS histogram $\mathbf{b}_s = (b_s[1], \ldots, b_s[K])$ which describes the song's acoustic content with respect to the BoS codebook $\mathcal{V}$. The song s is also associated with an annotation vector $\mathbf{c}_s = (c_s[1], \ldots, c_s[|\mathcal{T}|])$ which expresses the song's semantic content with respect to $\mathcal{T}$, where $c_s[i] = 1$ if s has been annotated with tag $w_i \in \mathcal{T}$, and $c_s[i] = 0$ otherwise. In short, a training dataset is a collection of histogram-annotation pairs $\mathcal{X}_t = \{(\mathbf{b}_s, \mathbf{c}_s)\}_{s=1}^{|\mathcal{X}_t|}$.

Given a training set $\mathcal{X}_t$, we estimate a series of tag-level models that capture the statistical regularities in the BoS histograms representing the subsets of songs in $\mathcal{X}_t$ associated with each tag in $\mathcal{T}$, using standard text-mining algorithms. Given the BoS histogram representation of a novel song, $\mathbf{b}$, we can then resort to the previously trained tag-level models to compute the relevance of each tag in $\mathcal{T}$ to the song. In this work, we consider two algorithms with a probabilistic interpretation, which output posterior probabilities $p(w_i|\mathbf{b})$ that a tag w_i applies to a song with BoS histogram $\mathbf{b}$. We then aggregate the posterior probabilities to form a semantic multinomial (SMN) $\mathbf{p} = (p_1, \ldots, p_{|\mathcal{T}|})$, characterizing the relevance of each tag to the song ($p_i \geq 0\ \forall i$ and $\sum_{i=1}^{|\mathcal{T}|} p_i = 1$).

Annotation involves selecting the most representative tags for a new song, and hence reduces to selecting the tags with the highest entries in the SMN $\mathbf{p}$. Retrieval consists of rank ordering a set of songs $\mathcal{S} = \{s_1, s_2 \ldots s_R\}$ according to their relevance

to a query. When the query is a single tag w_i from $\mathcal{T}$, we define the relevance of a song to the tag by p_i, and hence retrieval consists of ranking the songs in the database based on the i^{th} entry in their SMN.

B. Learning Tag Models From BoS Histograms

The BoS histogram representation of songs is amenable to a variety of annotation and retrieval algorithms. In this work, we investigate one generative algorithm, Codeword Bernoulli Average (CBA), and one discriminative algorithm, multiclass kernel logistic regression (LR).

1) Codeword Bernoulli Average: The CBA model proposed by Hoffman *et al.* [14] is a generative process that models the conditional probability that a tag applies to a song. Hoffman *et al.* define CBA based on a vector quantized (VQ) codebook representation of songs. For our work, we use instead the BoS representation. CBA defines a collection of binary random variables $y_{ji} \in \{0, 1\}$, which determine whether or not tag w_i applies to song j, with a two-step generative process. First, a codeword $z_{ji} \in \{1, \ldots, K\}$ is chosen according to the distribution given by the song's BoS histogram $\mathbf{b}_j$, i.e.,

$$p(z_{ji} = l|\mathbf{b}_j) = b_j[l]. \tag{5}$$

Then, a value for y_{ji} is chosen from a Bernoulli distribution with parameter β_{li}, which represents the probability of tag word w_i given codeword l:

$$\begin{aligned} p(y_{ji} = 1|z_{ji}, \boldsymbol{\beta}) &= \beta_{z_{ji}i}, \\ p(y_{ji} = 0|z_{ji}, \boldsymbol{\beta}) &= 1 - \beta_{z_{ji}i}. \end{aligned} \tag{6}$$

We use the author's code [14] to fit the CBA model, i.e., learn the parameters $\boldsymbol{\beta}$. To obtain the SMN of a novel song with BoS histogram $\mathbf{b}$, we compute the posterior probabilities $p(y_i = 1|\mathbf{b}, \beta) = p_i$ under the estimated CBA model, and normalize $\mathbf{p} = (p_1, \ldots, p_K)$ by its L_1 norm.

2) Multiclass Logistic Regression: For each tag, logistic regression defines a linear classifier with a probabilistic interpretation by fitting a logistic function to BoS histograms associated and not associated with the tag:

$$P(w_i|\mathbf{b}, \beta_i, \alpha_i) = \frac{1}{1 + \exp\left(\beta_i^T \mathbf{b} + \alpha_i\right)}. \tag{7}$$

Kernel logistic regression [40] finds a linear classifier after applying a non-linear transformation to the data, $\varphi : \mathbb{R}^d \to \mathbb{R}^{d_\varphi}$. The feature map φ is indirectly defined via a kernel function $\kappa(\mathbf{a}, \mathbf{b}) = \langle \varphi(a), \varphi(b) \rangle$.

In our experiments, we use the histogram intersection kernel [41], which is defined by the kernel function:

$$\kappa(\mathbf{a}, \mathbf{b}) = \sum_j \min(a[j], b[j]), \tag{8}$$

where $\mathbf{a}$ and $\mathbf{b}$ are BoS histograms. In our implementation we use the software package Liblinear [42] to learn an L_2-regularized logistic regression model for each tag using the "one-vs-the rest" approach. We select the regularization parameter C by performing 4-fold cross-validation on the training set. Given the BoS histogram of a new song $\mathbf{b}$, we collect the posterior probabilities $p(w_i|\mathbf{b})$ and normalize to obtain the song's SMN.

V. Music Data

In this section we introduce the two music datasets used in our experiments, along with the audio features and the base models used for the codewords.

A. CAL500 Dataset

The CAL500 [5] dataset consists of 502 Western popular songs from 502 different artists. Each song-tag association has been evaluated by at least 3 humans, using a vocabulary of 149 tags, including genres, instruments, vocal characteristics, emotions, acoustic characteristics, and use cases. CAL500 provides binary annotations that can be safely considered as hard labels, i.e., $c[i] = 1$ when a tag i applies to the song and 0 when the tag does not apply. We restrict our experiments to the 97 tags with at least 30 example songs. CAL500 experiments use 5-fold cross-validation where each song appears in the test set exactly once.

B. CAL10K Dataset

The CAL10K dataset [43] is a collection of over ten thousand songs from 4,597 different artists, weakly labeled (i.e., song annotations may be incomplete) from a vocabulary of over 500 tags. The song-tag associations are mined from Pandora's website. We restrict our experiments to the 55 tags in common with CAL500.

C. Codeword Base Models and Audio Features

We choose a priori five different base models for codewords: two time scales of Gaussians (G_1 and G_2) and three time scales of DTs (DT_1, DT_2 and DT_3). These time scales were chosen by referring to previous work [5], [6], and with the goal of spanning a reasonable range of scales that occur in music. For example, codewords from model class DT_1 will capture dynamics that unfold over a few milliseconds to while codewords from model class DT_3 will capture dynamics that unfold in the time span of several seconds. We will experimentally determine the best base models and combinations of models from these five choices. For the Gaussian codewords, we use as audio features the first 13 Mel-frequency cepstral coefficients (MFCCs) appended with first and second instantaneous derivatives (MFCC deltas). This results in 39-dimensional feature vectors [44]. For the DT codewords, we model 34-bin Mel-frequency spectral features.[2] We note that these features represent timbre, but not other facets of music such as rhythm or tonality.

To motivate the difference in features between Gaussian and DT codewords, note that Mel-frequency cepstral coefficients (MFCCs) use the discrete cosine transform (DCT) to decorrelate the bins of a Mel-frequency spectrum. In Section III-A.2 we noted that the DT model can be viewed as a time-varying PCA representation of the audio feature vectors. This suggests that we can represent the Mel-frequency spectrum over time as the output of the DT model. In this case, the columns of the observation matrix (a learned PCA matrix) are analogous to the DCT basis functions, and the hidden states are the coefficients (analogous to the MFCCs). The advantage of learning the PCA representation, rather than using the standard DCT basis, is that

[2]We use 30 Mel-frequency bins to compute the MFCC features, which provides a similar spectral resolution to the 34-bin Mel-frequency features.

TABLE I
DESCRIPTION OF BASE MODELS USED IN EXPERIMENTS: TWO TIME SCALES OF GAUSSIANS AND THREE TIME SCALES OF DTS

Codeword Base Model	Window length (η)	Fragment length (λ)	Fragment step (ν)
G_1	46 ms	—	—
G_2	93 ms	—	—
DT_1	12 ms	726 ms	145 ms
DT_2	93 ms	5.8 s	1.16 s
DT_3	93 ms	11.6 s	2.32 s

different basis functions (matrices) can be learned to best represent the particular codeword, and different codewords can focus on different frequency structures. Also, note that since the DT explicitly models the temporal evolution of the audio features, we do not need to include their instantaneous derivatives (as in the MFCC deltas).

In both cases, feature vectors are extracted from half-overlapping windows of audio. The window and fragment length for each codeword base model are specified in Table I. These choices were intended to cover a few sufficiently distinct time resolutions, and the step size between fragments is relatively small in order to increase the number of fragments we can extract from each song.

VI. EXPERIMENTAL EVALUATION

In this section we present experimental results on music annotation and retrieval using the BoS representation. Annotation performance is measured using mean per-tag precision (P), mean per-tag recall (R) and mean per-tag F-score. Retrieval performance is measured using area under the receiver operating characteristic curve (AROC), mean average precision (MAP), and precision at 10 (P10) averaged over all one-tag queries. We refer the reader to Turnbull *et al.* [5] for a detailed definition of these metrics.

A. Design Choices

The combination of four codebook generation methods, five base models of codewords and two BoS histogram annotation algorithms creates a large number of possibilities for the implementation of the BoS framework. Along with the choice of codebook size K and histogram smoothing parameter k, it becomes difficult to present an exhaustive comparison of all possibilities. We instead evaluate each design parameter independently while keeping all other parameters fixed, which leads us to select (1) the song-based procedure for codebook generation, (2) a combination of three codeword base models (DT_1, DT_2, and G_1; see Table I), and (3) logistic regression to annotate BoS histograms. Appendix I discusses each of these design choices in more detail. For each design choice, alternatives are compared by measuring the annotation and retrieval performance of a series of BoS auto-taggers. In the experiments in this section, reported metrics are the result of five-fold cross-validation where tag model training and testing are done using LR on CAL500.

1) EXP-1: Codebook Generation Methods: We first compare the four methods of codebook generation presented in Section III-B. We restrict our attention to codebooks consisting of a single base model (G_1 only or DT_2 only) and use a small codebook size K (because for the collection-based codebook learning procedures, learning time quickly becomes prohibitive with larger values of K). Additional parameter settings for this experiment can be found in Appendix I.

TABLE II
EXP-1: LEARNING A BOS CODEBOOK WITH FOUR DIFFERENT METHODS: FRAGMENT-BASED (FRAG), SONG-BASED (SONG) COLLECTION-BASED (COLL) AND HIERARCHICAL (HIER). $K = 128$ CODEWORDS ARE LEARNED FROM A CODEBOOK SONG SET OF $|\mathcal{X}_c| = 64$ SONGS. BOS HISTOGRAMS USE SMOOTHING PARAMETER $k = 10$, AND ARE ANNOTATED WITH LR. SPEEDUP IS REPORTED RELATIVE TO THE SONG-BASED METHOD

Method	Speedup	Annotation			Retrieval		
		P	R	F-Score	AROC	MAP	P10
Codeword Base Model G_1							
Frag	81.3	0.370	0.217	0.217	0.689	0.423	0.439
Song	1	**0.390**	**0.229**	**0.230**	0.701	0.437	0.449
Coll	0.015	0.382	0.228	0.229	**0.704**	**0.441**	**0.452**
Hier	0.875	0.376	0.227	0.227	0.697	0.435	0.447
Codeword Base Model DT_2							
Frag	54.2	0.381	0.229	0.225	0.695	0.433	0.456
Song	1	**0.411**	**0.246**	**0.247**	0.712	**0.457**	**0.479**
Coll	0.036	0.398	0.243	0.242	**0.709**	0.455	0.471
Hier	0.070	0.390	0.236	0.230	0.705	0.447	0.463

Annotation and retrieval results for these experiments are reported in Table II. We find that learning codewords with the song-based procedure produces the best combination of performance and efficiency. While highly efficient, learning codewords at the fragment level gives sub-optimal performance. The collection-based procedure (with standard EM) provides good performance in terms of annotation and retrieval, but training times are an order of magnitude higher than the song-based method. Moreover, the results in Table II are for relatively small codebooks ($K = 128$) — as the codebook size increases, the computational cost of the EM algorithm becomes even more prohibitive. In Appendix I we analyze the codebooks produced by each method in more detail, in particular investigating the diversity and spread of the learned codewords using appropriate pairwise distances and 2-D visualization. Hence, for the remainder of experiments codebooks are learned by the song-based method.

2) EXP-2: Codebook Size: In this experiment we investigate the effect of the codebook size, K, on annotation and retrieval performance. Fig. 4 plots the retrieval performance (MAP) as a function of the codebook size, for values of K ranging from 50 to 4800. Further details about the parameter settings for this experiment are provided in Appendix I. We find that once the codebook has reached a certain size, adding more codewords has a negligible effect on performance, and in fact for DT codewords performance eventually degrades as the codebook becomes large. We see an optimal range for K between 800 and 3200, corresponding to an optimal range for K_s between 2 and 8. Based on these results we choose for future experiments $\tilde{K} = 1600$ (corresponding to $K_s = 4$) for each codeword group, for codebooks learned from CAL500. We note however, that these results are for a single base model, and we see in preliminary experiments that when codewords from a variety of different base models are combined, we can use a larger codebook without saturating performance (see Section VI-A). We also note that these results are based on a fixed codebook song set size $|\mathcal{X}_c| = 400$.

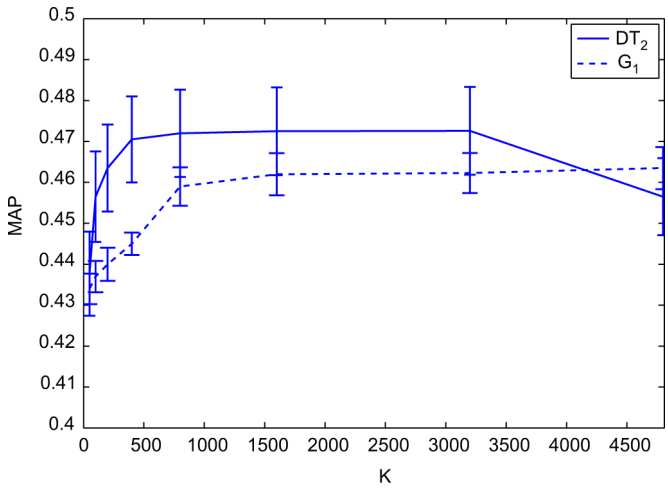

Fig. 4. EXP-2: Retrieval performance (MAP) as a function of BoS codebook size K. Each codebook is generated with a song-based approach. For a fixed codebook song set $\mathcal{X}_c$, the codebook size is varied by varying the number of codewords learned per song, K_s. The error bars indicate the standard error over the five cross-validation folds of CAL500.

TABLE III
EXP-3: RESULTS USING BOS CODEBOOKS WITH CODEWORDS FROM VARIOUS BASE MODELS. ALL CODEBOOKS HAVE $K = 4800$ CODEWORDS, WITH VARYING NUMBER OF CODEWORD BASE MODELS, M, CODEWORDS PER BASE MODEL, $\tilde{K}$, AND SIZE OF SONG MODEL, K_s. BOS HISTOGRAMS USE SMOOTHING PARAMETER $k = 10$, AND ARE ANNOTATED USING LR

Base Models	Annotation			Retrieval		
	P	R	F-Score	AROC	MAP	P10
DT_2	0.418	0.253	0.258	0.724	0.469	0.491
$DT_{1,2}$	0.424	0.255	0.263	0.729	0.475	0.499
DT_2,G_1	**0.439**	0.258	0.264	0.735	0.479	0.500
$DT_{1,2}$,G_1	0.433	**0.263**	**0.267**	**0.744**	**0.489**	**0.506**
$DT_{1,2,3}$,G_1	0.431	0.260	0.265	0.739	0.483	0.504
$DT_{1,2,3}G_{1,2}$,	0.428	0.261	0.266	0.739	0.483	0.505

We found in preliminary experiments that with a larger codebook song set, we can keep K_s in a similar range, allowing for larger codebooks before performance deteriorates. In particular, when using the CAL10K dataset as the codebook song set, we set $K_s = 2$ (see Section VI-B.2).

3) EXP-3: Codeword Base Models: One of the key advantages of the BoS framework is that it allows one to leverage codewords from different base models (i.e., model types and time scales) to produce a rich codebook. In this set of experiments, we investigate the extent to which using multiple base models is useful.

We evaluate various combinations of the codeword base models defined in Table I, adjusting the number of codewords learned from each base model, $\tilde{K}$, to obtain a fixed codebook size K. We begin with a codebook with all codewords of base model DT_2, which was the best performing individual codeword base model in preliminary experiments, and compare it with codebooks containing additional base models of codewords. Additional parameter settings can be found in Appendix I.

Results are presented in Table III. We find that using multiple types of generative models with different time scales in a codebook is beneficial. Compared to a codebook of DT_2 codewords only, adding Gaussian codewords[3] (i.e., G_1) or DT codewords on a shorter time scale[4] (i.e., DT_1) leads to a significant improvement in performance. In particular, the best performance is achieved when all of the above base models are combined: DT_1, DT_2 and G_1. This is a significant improvement in performance over the DT_2, G_1 codebook.[5] While Fig. 4 shows that adding more codewords of the *same* base model to a codebook of size 1600 saturates and eventually deteriorates performance, Table III illustrates that adding codewords of a *different base model* can further improve performance. This confirms the codebook is *enriched* with codewords that model different aspects of the musical signal.

We notice, however, that adding a third set of DT codewords no longer improves performance. We speculate this is because the DT_2 and DT_3 base models use feature vectors sampled at the same time resolution, η; they differ in the number of feature vectors, τ, in the fragments they model, which is twice as long for DT_3 as DT_2. The hidden state of a DT model evolves as a first order Markov process, with the evolution from state x_{t-1} to x_t defined by the transition matrix A. While a different time resolution can significantly affect A, the model (and the corresponding codewords) does not explicitly encode the fragment length, τ. The latter only defines the length of the input audio fragments used in the learning stage of the algorithm. Using larger values of τ means using longer audio fragments, resulting in more smoothed DT models. Hence the codewords in DT_2 and DT_3 may capture more or less the same dynamics, although the codewords in DT_3 may be more smoothed than those in DT_2.[6]

Similarly, adding a second set of Gaussian codewords provides no improvement in performance. Indeed, since Gaussians do not model temporal dynamics, the only time scale parameter to tune is the window length (i.e., η), varying which determines the amount of audio that is averaged over to produce timbral features. Therefore, features extracted at the time scale for G_2 are essentially more smoothed versions of the features extracted for G_1 (with slightly different frequency bands), and the codewords capture much of the same timbral information.

4) EXP-4: BoS Histogram Smoothing Parameter k: In this experiment we investigate the effect of the BoS histogram smoothing parameter k on annotation and retrieval performance. Fig. 5 illustrates a few example BoS histograms with varying values for k for a song from CAL500. When building BoS histograms, we assign the k most likely codewords (of each base model) to each portion of audio. Hence when a smaller value of k is used, songs tend to be represented by only a few different codewords, leading to more peaked histograms, while a larger value of k creates smoother histograms.

Details of the experimental setup are provided in Appendix I. Results are plotted in Fig. 6 as a function of k. We find that the performance is fairly stable for a wide range of values for k, reaching a peak between $k = 5$ and $k = 10$. The performance drops off gradually for large k and more steeply for small k. This leads us to select $k = 10$ as the BoS histogram smoothing parameter for further experiments.

[3]Paired t-test of MAP score on 97 tags: $t(96) = 6.43, p < 0.001$

[4]Paired t-test of MAP score on 97 tags: $t(96) = 9.08, p < 0.001$

[5]Paired t-test of MAP scores on 97 tags: $t(96) = 5.38, p < 0.001$

[6]See Chan and Vasconcelos [38] for more details.

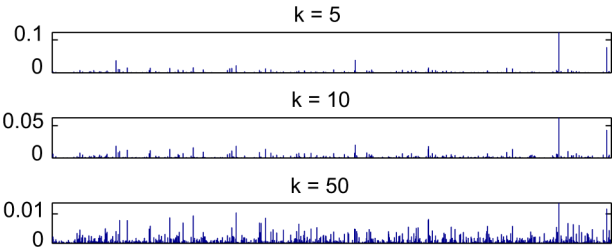

Fig. 5. BoS histograms with smoothing parameter $k = 5$, 10 and 50 for Guns N' Roses' *November Rain*. The smoothing parameter k determines the number of codewords to be assigned at each point in the song, so using a smaller value of k will assign only a few different codewords, leading to more peaked histograms, while a larger value of k creates smoother histograms. Note the different y-axis scales.

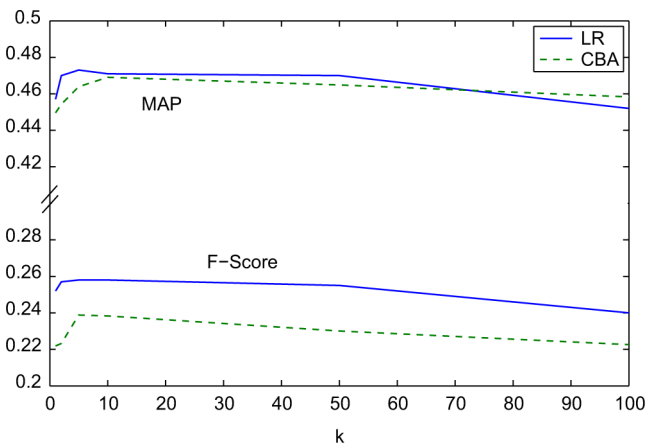

Fig. 6. EXP-4: Annotation (F-score) and retrieval (MAP) performance of the BoS approach, using LR on the CAL500 dataset, as a function of the smoothing parameter k.

We also notice throughout these (and following) experiments that LR tends to outperform CBA as an annotation algorithm, and hence LR is our annotation algorithm of choice.

B. BoS Performance

In this section we present experiments that demonstrate the effectiveness of the BoS representation for music annotation and retrieval, using the design choices made in the previous section. We compare the BoS approach to several baseline autotaggers on the CAL500 and CAL10K datasets.

1) Results on CAL500: We run annotation and retrieval experiments on CAL500 using two BoS auto-taggers based on a codebook of size $K = 4800$ which combines codeword base models G_1, DT_1 and DT_2, each with $\tilde{K} = 1600$ and obtained with the song-based method ($K_s = 4, |\mathcal{X}_c| = 400$) and histogram smoothing parameter $k = 10$. One auto-tagger uses LR to annotate BoS histograms and the second uses CBA — BoS-LR and BoS-CBA, respectively. For LR, the regularization parameter C is chosen by 4-fold cross-validation on the training set. All reported metrics are the result of five-fold cross-validation, where for each split of the data the whole training set is used as the codebook song set (i.e., $\mathcal{X}_c = \mathcal{X}_t$).

We compare our BoS-CBA and BoS-LR auto-taggers with three state-of-the-art baselines for automatic music tagging. The first two are based on hierarchically trained GMMs (HEM-GMM) [5] and hierarchically trained DTMs (HEM-DTM) [6], respectively. Note that the HEM-GMM and HEM-DTM approaches each leverage one base model from among those selected for our BoS codewords, but use it to directly represent tag-specific distributions on low-level audio feature spaces. In particular, the HEM-GMM auto-tagger learns tag-level GMM distributions with $K = 16$ mixture components over the same audio feature space as G_1. The HEM-DTM fits tag-level DTM distributions with $K = 16$ mixture components over the same space of audio fragments as used for DT_2. The hyperparameters for these methods were set to those used in previous works [5], [6].

TABLE IV
BoS Codebook Performance on CAL500, Using CBA and LR, Compared to Gaussian Tag Modeling (HEM-GMM), DTM Tag Modeling (HEM-DTM), Averaging of GMM and DTM Tag Models (HEM-AVG) and VQ Codebooks With LR

	Annotation			Retrieval		
	P	R	F-Score	AROC	MAP	P10
HEM-GMM	0.374	0.205	0.213	0.686	0.417	0.425
HEM-DTM	**0.446**	0.217	0.264	0.708	0.446	0.460
HEM-AVG	0.445	0.219	0.251	0.719	0.458	0.474
VQ-LR	0.413	0.243	0.255	0.717	0.456	0.480
BoS-CBA	0.378	0.262	0.248	0.738	0.482	0.505
BoS-LR	0.433	**0.263**	**0.267**	**0.744**	**0.489**	**0.506**

The third baseline, HEM-AVG, averages (on a tag-by-tag basis, by directly averaging the SMNs) the predictions of three HEM auto-taggers: two versions of HEM-DTM (based on DT_1 and DT_2, respectively) and HEM-GMM (based on G_1).

The fourth baseline (VQ-LR) is based on a VQ representation of songs [18]. Each song is represented as a Bag-of-Words (BoW) histogram by vector-quantizing its MFCC features with a VQ codebook of size $K = 2048$, and annotation is implemented using L_2-regularized logistic regression. In preliminary experiments we found that, similarly to the BoS representation, using a softened BoW histogram results in superior performance, and consequently we used a smoothing parameter of $k = 5$ to build VQ histograms. These hyperparameters were tuned using cross-validation over the CAL500 dataset, using the same methodology as for the BoS hyperparameters.

Results for these experiments are reported in Table IV. The BoS auto-taggers based on LR and CBA outperform the other approaches on all metrics except precision, where HEM-DTM performs best. HEM-AVG outperforms HEM-DTM and HEM-GMM in retrieval metrics, indicating that leveraging multiple time scales and models is beneficial in general, but BoS outperforms HEM-AVG, which demonstrates the additional modeling power the BoS framework provides.

In addition, we also notice that tagging BoS histograms with LR leads to higher performance than CBA. This is not surprising, as LR is a discriminative algorithm which is typically better suited when training on small, hard-labeled datasets such as CAL500, whereas CBA is a generative algorithm.

We plot precision-recall curves for each of these auto-taggers in Fig. 7. First, we notice that BoS-LR achieves the best precision-recall trade-off point, and has the best precision for intermediate and large values of recall. However, at low levels of recall, HEM-DTM has the highest precision — meaning some DTM tag models may be finely tuned to capture certain patterns associated with a tag, at the risk of not generalizing to all occurrences of that tag. We also note that BoS-LR has a higher precision than BoS-CBA at all levels of recall.

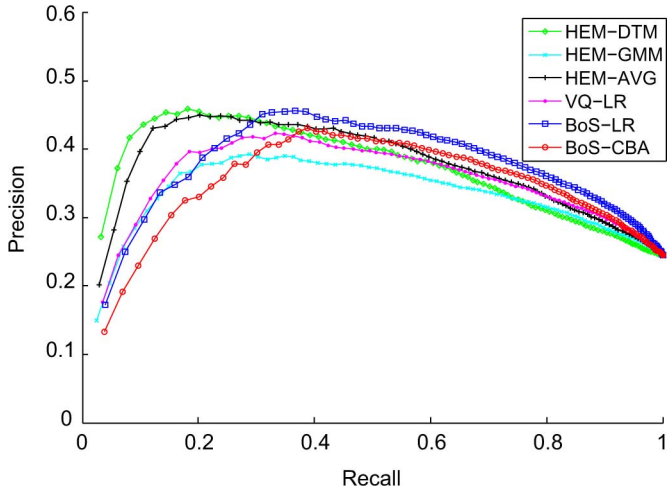

Fig. 7. Precision-recall curves for various auto-taggers on CAL500. HEM-DTM dominates at low recall, but BoS outperforms HEM-DTM at higher recall.

TABLE V
TOP-10 RETRIEVED SONGS FOR "FEMALE LEAD VOCALS". SONGS WITH FEMALE LEAD VOCALS ARE MARKED IN BOLD

Rank	HEM-GMM	
1	Joram	"Solipsism"
2	**Shakira**	**"The one"**
3	Dennis Brown	"Tribulation
4	The Replacements	"Answering machine"
5	Lambert, Hendricks & Ross	"Gimme that wine"
6	**Ray Charles**	**"Hit the road Jack"**
7	**ABBA**	**"S.O.S."**
8	Tim Rayborn	"Yedi tekrar"
9	Everly Brothers	"Take a message to Mary"
10	**Sheryl Crow**	**"I shall believe"**
Rank	**HEM-DTM**	
1	**Sheryl Crow**	**"I shall believe"**
2	Louis Armstrong	"Hotter than that"
3	Joram	"Solipsism"
4	**The Mamas and the Papas**	**"Words of love"**
5	**Jewel**	**"Enter from the east"**
6	**Shakira**	**"The one"**
7	**ABBA**	**"S.O.S."**
8	**Aimee Mann**	**"Wise up"**
9	Panacea	"Dragaicuta"
10	**Ike and Tina Turner**	**"River deep mountain high"**
Rank	**BoS-LR**	
1	**ABBA**	**"S.O.S."**
2	**The Mamas and the Papas**	**"Words of love"**
3	**Shakira**	**"The one"**
4	**Sheryl Crow**	**"I shall believe"**
5	**Spice Girls**	**"Stop"**
6	**Cyndi Lauper**	**"Money changes everything"**
7	**Aimee Mann**	**"Wise up"**
8	**Ike and Tina Turner**	**"River deep mountain high"**
9	Bryan Adams	"Cuts like a knife"
10	**Jewel**	**"Enter from the east"**

In order to provide a qualitative sense of how the BoS autotagger performs for retrieval, Table V shows the top-10 retrieval results for the query "female lead vocals", for the three autotaggers HEM-GMM, HEM-DTM and BoS-LR.

2) Results on CAL10K: In order to analyze the performance of the BoS approach when dealing with larger, weakly labeled music collections, which are more representative of web-scale applications, we train both the codebook and the annotation models on CAL10K. Specifically, we subsample from CAL10K a codebook song set $\mathcal{X}_c$ consisting of one song from each artist (i.e., $|\mathcal{X}_c| = 4{,}597$), and use the song-based method with $K_s = 2$ to compile a BoS codebook, resulting in $\tilde{K} = 9{,}194$ codewords from each base model. (Because the codebook song set is much larger in this experiment than in previous CAL500 experiments, we find that it can support much larger codebook sizes.) Similar to the CAL500 experiments, we use three base models of codewords, G_1, DT_1 and DT_2, leading to $K = 3\tilde{K} = 27{,}582$ codewords overall. As CAL10K is not well suited for evaluation purposes, (due to the weakly labeled nature of its annotations the absence of a tag in a song's annotations does not generally imply that it does not apply to the song), we train BoS tag models on all of CAL10K, and reliably evaluate them on CAL500. Our experiments are limited to the 55 tags that CAL10K and CAL500 have in common. We fix the hyperparameters to the best setting found through cross-validation on CAL500 (see Section VI-B.1), i.e., BoS histogram smoothing parameter $k = 10$ and LR regularization trade-off $C = 1$. We compare the performance of BoS codebooks to the HEM-GMM, HEM-DTM, and VQ-LR autotaggers.

TABLE VI
BOS CODEBOOK PERFORMANCE FOR TRAINING TAG MODELS ON CAL10K AND EVALUATING ON CAL500, USING CBA AND LR, COMPARED TO GAUSSIAN TAG MODELING (HEM-GMM), DTM TAG MODELING (HEM-DTM), AND VQ CODEBOOKS WITH LR

	Annotation			Retrieval		
	P	R	F-Score	AROC	MAP	P10
HEM-GMM	0.297	0.404	0.264	0.714	0.350	0.315
HEM-DTM	0.289	0.391	0.259	0.702	0.354	0.314
VQ-LR	0.295	0.412	0.263	0.703	0.347	0.315
BoS-CBA	0.310	**0.495**	0.295	0.756	0.414	**0.361**
BoS-LR	**0.329**	0.479	**0.312**	**0.759**	**0.416**	**0.361**

Annotation and retrieval results reported in Table VI demonstrate that the BoS approach outperforms direct tag modeling also using larger, weakly annotated collections for training. In addition, we notice that BoS-CBA catches up with BoS-LR on several performance metrics. This is a consequence of the weakly-annotated nature of CAL10K, which makes the employment of generative models (such as CBA) more appealing relative to discriminative models (such as LR).

VII. DISCUSSION

In this section we look at the performance of the BoS autotagger in more detail, substantiating the claims we made in the introduction about the advantages of the BoS framework. Namely, that decoupling modeling music from modeling tags makes the BoS system more robust for tags with few training examples (Section VII-A), that combining multiple types and time scales of generative models improves performance (VII-B) and that incorporating unlabeled songs when building a codebook is advantageous (VII-C).

A. Decoupling Modeling Music From Modeling Tags

As anticipated in the introduction, we expect that the BoS approach will be particularly robust for tags with fewer training examples when compared to traditional generative algorithms, because the relatively simpler tag models used in the BoS approach are less prone to overfitting. To demonstrate this, we analyze performance on subsets of CAL500 tags defined by their maximum cardinality α, i.e., subsets $\{w \in \mathcal{T} \mid |w| < \alpha\}$, where

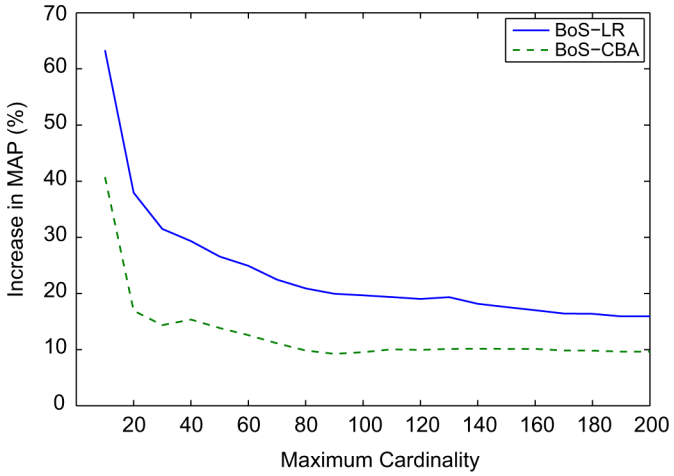

Fig. 8. Retrieval performance (MAP) of the BoS approach, relative to HEM-DTM, as a function of maximum tag cardinality. For each point in the graph, the set of all CAL500 tags is restricted to those associated with a number of songs that is at most the abscissa value. Experiments are run on the resulting (reduced) data set as described in Section VI-B.1, and the performance metrics are averaged over each tag subset.

the cardinality $|w|$ of a tag w is defined as the number of examples in the data set that are labeled with the tag.

We report retrieval performance for both the BoS approaches and the direct generative approach with DTM models (HEM-DTM), using the experimental setup in Section VI-B.1. In Fig. 8 we plot the relative improvement in retrieval performance (MAP score relative to the MAP score of HEM-DTM) for both BoS auto-taggers as a function of maximum tag cardinality. The BoS approach achieves the greatest improvement for tags with few training examples. Since the BoS approach decouples modeling music from modeling tags, simple tag models (with few tunable parameters) can leverage the full descriptive power of a rich BoS codebook while avoiding the risk of over-fitting small training sets.

B. Combining Codeword Base Models

A key advantage of the BoS framework is that it enables combining different types of generative models over various time scales in a single auto-tagger. We expect that different tags will be better modeled by different model types and time scales. Coviello *et al.* discuss this in [6], noting that DTM models are more suitable for tags with characteristic temporal dynamics (e.g., tempo, rhythm, etc.)), while some other tags are well characterized by timbre alone and thus are better suited for GMMs. A framework such as BoS, which allows the combination of model types and time scales, can therefore enable good performance over these different kinds of tags by learning to use codewords from the most suitable base model class in each tag model.

To illustrate this, Table VII compares the annotation (F-score) performance of a BoS-LR autotagger based on four codebooks, using: (1) base model DT_1 only (2) base model DT_2 only (3) base model G_1 only, and (4) base models DT_1, DT_2 and G_1, for a few tags in the CAL500 dataset.

DT codewords prove suitable for tags with significant temporal structure, such as vocal characteristics and emotions such as "angry" and "aggressive". Instruments such as electric guitar and synthesizer also perform well, perhaps because these instruments exhibit some temporal signature that is captured in the model. We can infer that tags that perform better with DT_1 are characterized by dynamics that unfold more quickly than tags that perform better with DT_2, e.g., "fast" (recall that DT_1 models fragments of 726 ms, at a resolution of 12 ms per feature vector, while DT_2 models fragments of 5.8 s, at a resolution of 93 ms per feature vector). Additionally, as the performance averaged over all the tags is a bit higher with DT_2 codewords, we gather that the longer time scale is a better "catch-all" to model a wide range of tags. Other tags, such as "low energy" and "mellow" seem to have little in the way temporal dynamics to model, and are best described by timbre information alone.

We note that the BoS codebook combining all three base models often leads to comparable or higher performance to the highest performing individual base model codebook.

TABLE VII
COMPARISON OF PER-TAG F-SCORES FOR BOS AUTOTAGGERS BASED ON FOUR CODEBOOKS: (1) BASE MODEL DT_1 ONLY (2) BASE MODEL DT_2 ONLY (3) BASE MODEL G_1 ONLY AND (4) BASE MODELS DT_1, DT_2 AND G_1

Tag	DT_1	DT_2	G_1	$\mathrm{DT}_{1,2}$,G_1
DT_1 is the best-performing base model				
angry	0.42	0.32	0.34	0.43
drum machine	0.45	0.40	0.42	0.47
electric guitar	0.28	0.27	0.18	0.29
electronica	0.56	0.48	0.50	0.62
fast	0.38	0.36	0.17	0.38
DT_2 is the best-performing base model				
aggressive	0.31	0.40	0.35	0.47
classic rock	0.45	0.48	0.40	0.49
emotional vocals	0.26	0.37	0.25	0.30
female lead vocals	0.70	0.72	0.45	0.70
synthesizer	0.32	0.37	0.29	0.38
G_1 is the best-performing base model				
going to sleep	0.29	0.33	0.40	0.40
low energy	0.55	0.51	0.56	0.56
mellow	0.37	0.35	0.42	0.49
positive	0.26	0.29	0.32	0.33
tender	0.51	0.49	0.52	0.57
average	0.253	0.258	0.245	0.267

C. Incorporating Unlabeled Songs in the Codebook Song Set

An additional advantage of the BoS framework is that the codebook can be learned without an annotated corpus of audio data. This suggests that the codebook can be enriched by expanding the codebook song set to include potentially large, unannotated music collections. To illustrate this intuition, we compare a variety of BoS auto-taggers, each of which differ in the codebook song set $\mathcal{X}_c$.

We start by forming three different codebook song sets by aggregating subsets of CAL10K and CAL500: (1) $\mathcal{X}_c = \mathrm{C400}$, the codebook song set corresponding to the training set for each cross-validation split of CAL500 (as in Section VI-B.1), (2) $\mathcal{X}_c = \mathrm{R400}$, a codebook song set that is a subset of 400 randomly selected songs from the CAL10K collection and (3) $\mathcal{X}_c = \mathrm{A5K}$, the codebook song set that is a subset of the CAL10K dataset consisting of one song from each of the 4,597 artists (as in Section VI-B.2). Note that, since we are using only $\mathcal{X}_t = \mathrm{C400}$ as the training set in these experiments, the CAL10K

TABLE VIII
BOS CODEBOOK PERFORMANCE ON CAL500 USING CODEBOOKS LEARNED FROM VARIOUS CODEBOOK SONG SETS. BOS HISTOGRAMS USE SMOOTHING PARAMETER $k = 10$ AND ARE ANNOTATED WITH LR

Codebook Song Set ($\mathcal{X}_c$)	Codebook Size (K)	Annotation			Retrieval		
		P	R	F-Score	AROC	MAP	P10
C400	4800	0.433	0.263	0.267	**0.744**	0.489	0.506
R400	4800	0.427	0.260	0.266	0.739	0.484	0.505
A5K	27,582	0.438	0.268	0.273	**0.744**	0.491	**0.515**
C400 ∪ R400	9600	0.441	0.266	0.270	0.740	0.488	0.511
C400 ∪ A5K	29,982	**0.444**	**0.273**	**0.283**	0.743	**0.494**	0.511

dataset, and thus R400 and A5K, can be considered as collections of unlabeled data that could be used to form a codebook song set. In addition, we consider codebooks derived from the union of C400 with R400 and A5K, respectively. This is a practical scenario — an annotated music collection is augmented with additional unlabeled songs to build a richer codebook.

Details of the experimental setup can be found in Appendix III. Annotation and retrieval results are reported in Table VIII. We notice that the best performances are obtained when a richer codebook is produced from combining the training set (i.e., C400) with a much larger collection of unlabeled music, corresponding to $\mathrm{C400} \cup \mathrm{A5K}$ in Table VIII. Adding only a small set of unlabeled music to the codebook song set (i.e., $\mathrm{C400} \cup \mathrm{R400}$) does not lead to substantial improvements over using only the training set.

We have shown before (in Section VI-A) that as we learn increasingly more codewords from the same collection, performance flattens and then deteriorates. However, if we increase the size of our codebook song set, the added information allows us to extract larger codebooks that are effectively enriched. This results in a slight improvement of performance.

In addition, we notice that when codebook song sets are equally sized, using the training collection as the codebook song set (C400) outperforms using a different collection (R400). This result may be affected by the fact that CAL500 is mostly pop music, while CAL10K has many other genres (classical, jazz, etc.), so codewords learned from CAL10K may not be as relevant for CAL500. However, growing the size of the collection of unlabeled songs (A5K) allows it to catch up with, and in fact outperform, the C400 codebooks. The ability to leverage a large codebook compiled off-line could, for example, be useful for an application that learns personalized retrieval models. Such a system should be based on a BoS codebook large and varied enough to represent well the music various users are interested in. Personalized tag models can then be learned from a smaller, user-specific collection, perhaps located on the user's personal device. As demonstrated in Section VI-B.1, these simple tag models can be estimated reliably even when the collection of songs associated with a specific tag is relatively small, as might be the case when training on a user's personal collection of music.

VIII. CONCLUSION

We have presented a semantic auto-tagging system for music that leverages a rich "bag of systems" representation based on generative modeling. The BoS representation allows for the integration of the descriptive qualities of various generative models of musical content with different time resolutions into a single histogram descriptor of a song. This approach improves performance over traditional generative modeling approaches to auto-tagging, which directly model tags using a single type of generative model. It also proves significantly more robust for tags with few training examples, and can be learned from a representative collection of songs which need not be annotated. We have shown the BoS representation to be effective for training annotation models on two very different datasets, one small and strongly annotated and one larger and weakly annotated, demonstrating its robustness under different scenarios.

APPENDIX A
DETAILS OF DESIGN CHOICES EXPERIMENTS

This Appendix provides the details of the parameter settings and design for the experiments in Section VI-A.

EXP-1: Codebook Generation Methods: In this experiment we learn codebooks using each of the four methods of codebook generation presented in Section III-B.

For each codebook generation method and each cross-validation split of CAL500, we construct one codebook based on G_1 only and a second codebook based on DT_2 only. We learn codebooks of size $K = 128$ (We choose small K because for the collection-based codebook learning procedures, learning time quickly becomes prohibitive with larger values of K). We select a small (in order to keep K small with $K_s > 1$) random subset of songs from the training set of CAL500 to form a codebook song set of size $|\mathcal{X}_c| = 64$. For the fragment-based method, each codeword is directly estimated from a randomly selected subset of audio data, consisting of 100 (sequential) feature vectors for Gaussian codewords and a fragment of $\tau = 125$ feature vectors for DT codewords. For the song-based method we set $K_s = 2$, and for the hierarchical collection-based method we set $K'_s = 4$. For the collection-based method we subsample the input data to alleviate the high cost in computational time of the EM algorithm. For the Gaussian codebook we randomly subsample 250,000 input feature vectors and for the DT codebook we randomly subsample 5000 input fragments ($\approx 30\%$ of the data in both cases).

We build BoS histograms with smoothing parameter $k = 10$ and use LR for annotation and retrieval. Tag model training and testing are done on CAL500, using five-fold cross-validation.

EXP-2: Codebook Size: In this experiment we investigate the effect of the codebook size, K, on annotation and retrieval performance. For each cross-validation split of CAL500, we use the song-based method to compile codebooks using the training set $\mathcal{X}_t$ as the codebook song set $\mathcal{X}_c$ (Now that we have settled on the song-based method of codebook generation, we can efficiently build larger codebooks from a larger codebook song set). We learn one codebook from base model DT_2 and a separate codebook from base model G_1. We vary the size of the codebook, K, from 50 to 4800 codewords. As the number of songs in the codebook song set is fixed[7], i.e., $|\mathcal{X}_c| = 400$, we vary the codebook size $K = K_s|\mathcal{X}_c|$ by changing K_s, i.e., the number of codewords learned per song. When learning codebooks consisting of fewer than 400 codewords, we group multiple songs to

[7]Actual sizes of the five training splits are either 401 or 402 songs, we subsample to 400.

TABLE IX
OVERVIEW OF EXPERIMENTS TO DETERMINE DESIGN CHOICES. EXP-1 EVALUATES FOUR DIFFERENT CODEBOOK GENERATION PROCEDURES. EXP-2 EVALUATES THE EFFECT OF THE CODEBOOK SIZE, K. EXP-3 INVESTIGATES DIFFERENT NUMBERS AND COMBINATIONS OF BASE MODELS IN A CODEBOOK. EXP-4 EVALUATES THE EFFECT OF THE HISTOGRAM SMOOTHING PARAMETER k

	EXP-1	EXP-2	EXP-3	EXP-4
$\mathcal{X}_c$	$\subset$ CAL500	$\subset$ CAL500	$\subset$ CAL500	$\subset$ CAL500
$\mathcal{X}_t$	$\subset$ CAL500	$\subset$ CAL500	$\subset$ CAL500	$\subset$ CAL500
$\lvert\mathcal{X}_c\rvert$	64	400	400	400
$\lvert\mathcal{X}_t\rvert$	$\approx$ 400	$\approx$ 400	$\approx$ 400	$\approx$ 400
K	128	50-4800	4800	1600
M	1	1	1-5	1
k	10	10	10	1-100

TABLE X
EXP-3: ALL CODEBOOKS HAVE $K = 4800$ CODEWORDS, WITH VARYING NUMBER OF CODEWORD BASE MODELS, M, CODEWORDS PER BASE MODEL, $\tilde{K}$, AND SIZE OF SONG MODEL, K_s

Base Models	M	$\tilde{K}$	K_s
DT_2	1	4800	12
$DT_{1,2}$	2	2400	6
DT_2,G_1	2	2400	6
$DT_{1,2}$,G_1	3	1600	4
$DT_{1,2,3}$,G_1	4	1200	3
$DT_{1,2,3}G_{1,2}$,	5	960	3

learn a single codeword. For example, for a codebook with 100 codewords, we learn each codeword from the combined audio data of four songs. In this case, $K_s = 1$ and the effective size of the codebook song set is $|\mathcal{X}_c'| = 100$.

BoS histograms use $k = 10$ and annotation and retrieval is done with LR. Tag model training and testing are done on CAL500, using five-fold cross-validation.

EXP-3: Codeword Base Models: In this experiment, we investigate codebooks combining multiple base models.

For each cross-validation split of CAL500, we use the song-based method to compile codebooks from all the songs in the training set (i.e., $\mathcal{X}_c = \mathcal{X}_t$). We evaluate various combinations of the codeword base models defined in Table IX, adjusting the number of codewords learned from each base model, $\tilde{K}$, to obtain a fixed codebook size K. We fix $K = 4800$ (in preliminary experiments, we found that when combining codewords of multiple base models, a larger codebook size is necessary to allow for a sufficient number of codewords from each base model). Accordingly, for a fixed size codebook song set ($|\mathcal{X}_c| = 400$) we vary the number of codewords $\tilde{K}$ of each base model by varying K_s. Table X specifies each of these parameter choices used to learn the codebooks in these experiments.

We use $k = 10$ to build BoS histograms and LR for annotation and retrieval. Tag model training and testing are done on CAL500, using five-fold cross-validation.

EXP-4: Bos Histogram Smoothing Parameter k: In this experiment we investigate the effect of the BoS histogram smoothing parameter k on annotation and retrieval performance.

For each cross-validation split of CAL500, we use the song-based method ($K_s = 4$) to compile codebooks of size $K = 1600$ from the songs in the training set (i.e., $\mathcal{X}_c = \mathcal{X}_t$). We use base model DT_2. We build BoS histograms with values of k ranging from 1 to 100, and perform annotation and retrieval with both LR and CBA. Tag model training and testing are done on CAL500, using five-fold cross-validation.

TABLE XI
MEAN PAIRWISE DISTANCE BETWEEN CODEWORDS LEARNED WITH FOUR DIFFERENT METHODS. WE COMPUTE AN APPROPRIATE MEAN PAIRWISE DISTANCE BETWEEN CODEWORDS — FOR GAUSSIAN CODEBOOKS, WE COMPUTE THE AVERAGE EUCLIDEAN DISTANCE BETWEEN THE MEANS OF EACH PAIR OF GAUSSIAN CODEWORDS; FOR DT CODEBOOKS, WE COMPUTE THE AVERAGE MARTIN DISTANCE BETWEEN EACH PAIR OF DT CODEWORDS

Codeword Class	Codebook Generation Method	Mean Pairwise Distance
G_1	Fragment-based	337
	Song-based	2173
	Collection-based	6085
	Hierarchical	3486
DT_2	Fragment-based	3.4
	Song-based	5.7
	Collection-based	8.9
	Hierarchical	6.5

APPENDIX B
COMPARISON OF CODEBOOK GENERATION METHODS

We expand the analysis of the four codebook generation methods presented in Section III-B: (1) learning each of the codewords independently from fragments of a song in $\mathcal{X}_c$, (2) learning small mixtures independently from individual songs in $\mathcal{X}_c$, (3) learning all codewords simultaneously from all available audio data in $\mathcal{X}_c$ using standard EM, and (4) learning all codewords simultaneously from all available audio data in $\mathcal{X}_c$ using the hierarchical EM (HEM) algorithm. In particular, we would like to gain more insight into the diversity of codewords generated with each method.

First, we compute the average of (an appropriate) pairwise distance between codewords in a codebook. This quantity provides some information about the quality of the codebook independent of the annotation and retrieval experiments. A larger mean pairwise distance between codewords indicates that codewords are less similar to each other, and hence have a greater variety of representative power, than codewords with smaller pairwise distances. For Gaussian codebooks, we compute the average Euclidean distance between the means of each pair of Gaussian codewords; for DT codebooks, we compute the average Martin distance [38] between each pair of DT codewords. We compute these pairwise distances on the codebooks learned in the experiments in Section VI-A.1, reported in Table XI. We see that the collection-based method (with standard EM) produces the codebooks with the largest mean distance, followed by the hierarchical collection-based method and the song-based method. The fragment-based method gives rise to the smallest distances between codewords.

Second, we visualize the span of the codebook for each of the codebook generation methods by computing a 2-D embedding (via the t-SNE method [39]) of the Gaussian codeword centers. This embedding is shown in Fig. 9. While the 2-D embedding and mean pairwise distance capture slightly different information about the spread of the codewords (The cost function

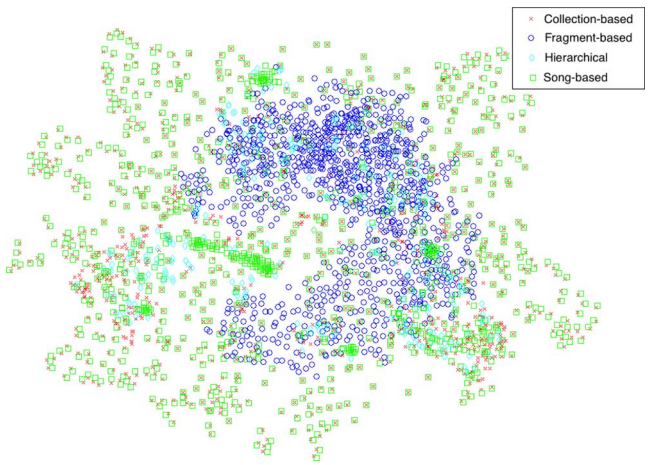

Fig. 9. 2D visualization of Gaussian codeword centers generated using the four different methods of codebook generation. The fragment-based method produced codewords that are more tightly clustered than the other methods, and showed correspondingly low performance in annotation and retrieval experiments (See Table XI).

used in t-SNE focuses on retaining local structure of the data, and, as such, is relatively insensitive to outlier structure; the mean pairwise distance, on the other hand, will be affected more significantly by outliers), they agree that the fragment-based method produces codewords with the lowest diversity, and the collection-based method produces codewords with the highest diversity.

We observe that once we get reasonably well-spread codewords (i.e., with the song-based method), we obtain good performance in terms of annotation and retrieval (see Section VI-A.1, Table X); further increasing the average distance between codewords (i.e., using collection-based or hierarchical methods) does not necessarily lead to a further improvement in annotation and retrieval performance. In fact, we might speculate that it is advantageous to have a higher density sampling of codewords modeling certain portions of the audio space that are most commonly found in music, leading to BoS representations that are better suited to discriminate between similar songs.

This analysis of codeword spread reinforces our intuition that codebooks with higher diversity between codewords perform better for annotation and retrieval, and supports our decision to use the song-based method of codeword generation for future experiments.

Appendix C
Details of Unlabeled Songs Experiment

This Appendix provides the details of the parameter settings and design for the experiments in Section VII-C.

Three different codebook song sets are used in this experiment: (1) $\mathcal{X}_c = \mathrm{C400}$, the codebook song set corresponding to the training set for each cross-validation split of CAL500 (as in Section VI-B.1), (2) $\mathcal{X}_c = \mathrm{R400}$, a codebook song set that is a subset of 400 randomly selected songs from the CAL10K collection and (3) $\mathcal{X}_c = \mathrm{A5K}$, the codebook song set that is a subset of the CAL10K dataset consisting of one song from each of the 4,597 artists (as in Section VI-B.2). We compare codebooks learned from each of these codebook song sents individually, in addition to codebooks derived from the union of C400 with R400 and A5K, respectively.

We use the song-based method to learn codebooks, combining codewords from base models DT_1, DT_2, and G_1. We use $K_s = 4$ for the C400 and R400 codebook, leading to $\tilde{K} = 1600$ codewords per base model, and codebooks of size $K = 4800$, and $K_s = 2$ for the A5K codebook, leading to $\tilde{K} = 9194$ codewords per base model, and codebooks of size $K = 27{,}582$. For the codebook song set $\mathcal{X}_c = \mathrm{C400} \cup \mathrm{R400}$, we use $K_s = 4$ to build a codebook of size $K = 9600$, and for the codebook song set $\mathcal{X}_c = \mathrm{C400} \cup \mathrm{A5K}$, we use $K_s = 2$ to build a codebook of size $K = 29{,}982$. We build BoS histograms with $k = 10$ and perform annotation and retrieval using LR.

Acknowledgment

The authors would like to thank L. Barrington, B. McFee, the editor and reviewers for their helpful feedback.

References

[1] J. S. W. Glaser, T. Westergren, and J. Kraft, "Consumer item matching method and system," U.S. patent 7,003,515, 2006.
[2] S. Clifford, "Pandora's long strange trip," *Inc.com*, 2007.
[3] M. Levy and M. Sandler, "A semantic space for music derived from social tags," in *Proc. ISMIR*, 2007, pp. 411–416.
[4] P. Lamere and O. Celma, "Music recommendation tutorial notes," *ISMIR Tutorial*, pp. 2–19, Sep. 2007.
[5] D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet, "Semantic annotation and retrieval of music and sound effects," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 16, no. 2, pp. 467–476, Feb. 2008.
[6] E. Coviello, A. Chan, and G. Lanckriet, "Time series models for semantic music annotation," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 19, no. 5, pp. 1343–1359, Jul. 2011.
[7] D. Aldous, I. Ibragimov, and J. Jacod, "Exchangeability and related topics," in *Ser. Lecture Notes in Mathematics*. Berlin/Heidelberg: Springer, 1985, vol. 1117, pp. 1–198.
[8] K. Ellis, E. Coviello, and G. Lanckriet, "Semantic annotation and retrieval of music using a bag of systems representation," in *Proc. ISMIR*, 2011, pp. 723–728.
[9] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Trans. Speech Audio Process.*, vol. 10, no. 5, pp. 293–302, Jul. 2002.
[10] E. Pampalk, A. Flexer, and G. Widmer, "Improvements of audio-based music similarity and genre classification," in *Proc. ISMIR*, 2005, pp. 628–633.
[11] J. Reed and C. Lee, "A study on music genre classification based on universal acoustic models," in *Proc. ISMIR*, 2006, pp. 89–94.
[12] E. Coviello, A. Chan, and G. Lanckriet, "The variational hierarchical EM algorithm for clustering hidden Markov models," *Adv. Neural Inf. Process. Syst.*, pp. 413–421, 25, 2012.
[13] M. Hoffman, D. Blei, and P. Cook, "Content-based musical similarity computation using the hierarchical Dirichlet process," in *Proc. ISMIR*, 2008, pp. 349–354.
[14] M. Hoffman, D. Blei, and P. Cook, "Easy as CBA: A simple probabilistic model for tagging music," *Proc. ISMIR*, pp. 369–374, 2009.
[15] M. Mandel and D. Ellis, "Multiple-instance learning for music information retrieval," in *Proc. ISMIR*, 2008, pp. 577–582.
[16] S. Ness, A. Theocharis, G. Tzanetakis, and L. Martins, "Improving automatic music tag annotation using stacked generalization of probabilistic svm outputs," in *Proc. ACM MULTIMEDIA*, , 2009, pp. 705–708.
[17] D. Eck, P. Lamere, T. Bertin-Mahieux, and S. Green, "Automatic generation of social tags for music recommendation," *Adv. Neural Inf. Process. Syst.*, 2007.
[18] B. Xie, W. Bian, D. Tao, and P. Chordia, "Music tagging with regularized logistic regression," in *Proc. ISMIR*, 2011, pp. 711–716.
[19] M. Casey, C. Rhodes, and M. Slaney, "Analysis of minimum distances in high-dimensional musical spaces," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 16, no. 5, pp. 1015–1028, Jul. 2008.

[20] P. Hamel, S. Lemieux, and Y. Bengio, "Temporal pooling and multi-scale learning for automatic annotation and ranking of music audio," in *Proc. ISMIR*, 2011, pp. 729–734.
[21] Y. Panagakis, C. Kotropoulos, and G. Arce, "Non-negative multilinear principal component analysis of auditory temporal modulations for music genre classification," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 18, no. 3, pp. 576–588, Mar. 2010.
[22] M. A. Domingues, F. Gouyon, A. M. Jorge, J. P. Leal, J. Vinagre, L. Lemos, and M. Sordo, Int. J. of Multimedia Inform. Retrieval "Combining usage and content in an online recommendation system for music in the long-tail," vol. 1 [Online]. Available: http://link.springer.com/content/pdf/10.1007%2Fs13735–012-0025–1
[23] M. Schedl and D. Schnitzer, "Hybrid retrieval approaches to geospatial music recommendation," in *Proc. 35th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval (SIGIR)*, Dublin, Ireland, Jul. 1, 2013.
[24] R. Foucard, S. Essid, and M. Lagrange, "Multi-scale temporal fusion by boosting for music classification," in *Proc. ISMIR*, 2011, pp. 663–668.
[25] C. Joder, S. Essid, and G. Richard, "Temporal integration for audio classification with application to musical instrument classification," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 17, no. 1, pp. 174–186, Jan. 2009.
[26] B. Sturm, M. Morvidone, and L. Daudet, "Musical instrument identification using multiscale Mel-frequency cepstral coefficients," in *Proc. EUSIPCO*, 2010, no. 1, pp. 477–481.
[27] N. Mesgarani, M. Slaney, and S. A. Shamma, "Discrimination of speech from nonspeech based on multiscale spectro-temporal modulations," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 14, no. 3, pp. 920–930, May 2006.
[28] H. Lee, Y. Largman, P. Pham, and A. Y. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," *Adv. Neural Inf. Process. Syst.*, 2009.
[29] J. Andén and S. Mallat, "Multiscale scattering for audio classification," in *Proc. ISMIR*, 2011.
[30] Z. Fu, G. Lu, K. M. Ting, and D. Zhang, "Music classification via the bag-of-features approach," *Pattern Recognition Lett.*, vol. 32, no. 14, pp. 1768–1777, 2011.
[31] T. Jebara, R. Kondor, and A. Howard, "Probability product kernels," *J. Mach. Learn. Res.*, vol. 5, pp. 819–844, 2004.
[32] A. Ravichandran, R. Chaudhry, and R. Vidal, "View-invariant dynamic texture recognition using a bag of dynamical systems," in *Proc. IEEE CVPR*, 2009, pp. 1651–1657.
[33] A. Chan, E. Coviello, and G. Lanckriet, "Clustering dynamic textures with the hierarchical EM algorithm," in *Proc. IEEE CVPR*, 2010, pp. 2022–2029.
[34] E. Coviello, A. Mumtaz, A. Chan, and G. Lanckriet, "Growing a bag of systems tree for fast and accurate classification," in *Proc. IEEE CVPR*, 2012, pp. 1979–1986.
[35] D. Sturim, D. Reynolds, E. Singer, and J. Campbell, "Speaker indexing in large audio databases using anchor models," in *Proc. IEEE ICASSP*, 2001, pp. 429–432.
[36] G. Doretto, A. Chiuso, Y. N. Wu, and S. Soatto, "Dynamic textures," *Int. J. Comput. Vis.*, vol. 51, no. 2, pp. 91–109, 2003.
[37] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. R. Statist. Soc. B*, vol. 39, pp. 1–38, 1977.
[38] A. B. Chan and N. Vasconcelos, "Modeling, clustering, and segmenting video with mixtures of dynamic textures," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 5, pp. 909–926, Jul. 2008.
[39] N. Vasconcelos and A. Lippman, "Learning mixture hierarchies," *Adv. Neural Inf. Process. Syst.*, 1998.
[40] T. Hastie, R. Tibshirani, and J. Friedman, *Elements Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York, NY, USA: Springer, 2009.
[41] M. Swain and D. Ballard, "Color indexing," *Int. J. Comput. Vis.*, vol. 7, no. 1, pp. 11–32, 1991.
[42] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *J. Mach. Learn. Res.*, vol. 9, pp. 1871–1874, 2008.
[43] D. Tingle, Y. E. Kim, and D. Turnbull, "Exploring automatic music annotation with acoustically-objective tags," in *Proc. MIR. ACM*. New York, NY, USA: ACM, 2010, pp. 55–62.
[44] L. Rabiner and B. H. Juang, *Fundamentals of Speech Recognition*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1993.

Katherine Ellis received the B.S. degree in Electrical Engineering from the University of Southern California in 2010 and the M.S. degree in Electrical and Computer Engineering from the University of California, San Diego (UCSD) in 2012. She is currently working toward her Ph.D. in Electrical and Computer Engineering at UCSD. Her research interests are in machine learning and applications to music information retrieval, and activity recognition.

Emanuele Coviello received his Bachelor degree in Information Engineering, and his Masters degree in Telecommunication Engineering from the Université degli Studi di Padova, Italy, in 2006 and 2008, respectively. He is currently a senior Ph.D. Candidate in Electrical and Computer Engineering, at the University of California, San Diego (UCSD). Emanuele is an expert in machine learning and multi-media content analysis. He received the Premio Guglielmo Marconi Junior Award in 2009, from the Guglielmo Marconi Foundation, and the Yahoo! Key Scientific Challenges Award in 2010, for his work on content-based analysis and tagging of music videos.

Antoni B. Chan received the B.S. and M.Eng. degrees in electrical engineering from Cornell University, Ithaca, NY, in 2000 and 2001, respectively, and the Ph.D. degree in electrical and computer engineering from the University of California, San Diego (UCSD), San Diego, in 2008. From 2001 to 2003, he was a Visiting Scientist with the Vision and Image Analysis Laboratory, Cornell University, Ithaca, NY, and in 2009, he was a Postdoctoral Researcher with the Statistical Visual Computing Laboratory, UCSD. In 2009, he joined the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong, as an Assistant Professor. His research interests include computer vision, machine learning, pattern recognition, and music analysis. Dr. Chan was the recipient of an NSF IGERT Fellowship from 2006 to 2008, and an Early Career Award in 2012 from the Research Grants Council of the Hong Kong SAR, China.

Gert Lanckriet received the MS degree in electrical engineering from the Katholieke Universiteit Leuven, Belgium, in 2000 and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 2001 and 2005, respectively. In 2005, he joined the Department of Electrical and Computer Engineering, University of California, San Diego, where he heads the Computer Audition Laboratory. His research focuses on the interplay of convex optimization, machine learning, and signal processing, with applications in computer audition and music information retrieval. He was awarded the SIAM Optimization Prize in 2008 and is the recipient of a Hellman Fellowship, an IBM Faculty Award, an NSF CAREER Award, and an Alfred P. Sloan Foundation Research Fellowship. In 2011, MIT Technology Review named him one of the 35 top young technology innovators in the world (TR35). He is a senior member of the IEEE.